END OF FILE – Ebook C/C++

http://endoffile.umforum.net (raidcall: 5762625)
Author: hfts315 (www.facebook.com/hacker.fts315)
Greetz: mmxm, f.ramon, susp3it0@virtual, eof member, entre outros
27/09/2013 – 15/11/2013 (sem revisão)

Ola membros do forum End Of File seja bem vindos ao ebook sobre programação c/c++, nesse ebook irei ensinar sobre programação basica da linguagem c/c++ e também sobre algumas coisas de niveis basicos a avançados, não vamos nos aprofundar em logica de programação embora vamos ver um pouco sobre ela para facilitar na criação dos algorítimos, esse ebook sera atualizado sempre que houver oportunidade e sempre nas paginas finais eu estarei listando as alterações que foram feitas para facilitar aos antigos leitores, não é permitido a alteração desse ebook sem autorização do autor também não é permitido a comercialização desse ebook, porem sera totalmente permitido a sua distribuição pela internet ou outros meios de forma não comercial.

nesse ebook abordaremos os seguintes tópicos diferença entre linguagem de programação e linguagem da máquina, diferença entre linguagem compilada e a linguagem interpretada, compilação por uma IDE ou compilador, manuseio do terminal ou prompt, diferença das interfaces CLI e GUI, estrutura básica da linguagem c/c++, diferença de entrada e saída de dados, bibliotecas padrões da linguagem c/c++, variáveis e constantes, vetores e matrizes, ponteiros e alocação dinâmica, estrutura condicional e laços, diretivas de pré-processamento, conversão de dados e tipos, manipulação de arquivos, manipulação do tempo, orientação a objeto da linguagem c++, APIs do sistema windows e linux entre outras coisas.

Indice

1.0 – Introdução da linguagem C/C++

A linguagem de programação C é uma das linguagens mais antigas e mais usada ate hoje, seja por programadores amadores a hackers, é uma linguagem muito poderosa e muito fácil achar compiladores para diversas plataformas e arquiteturas, essa linguagem serve para uso geral deis de um “hello world” na tela, uma página web por CGI, acender um led ou ate criar um sistema de inteligência artificial (por favor não criem a skynet, é sério), a linguagem C foi criada pelo Dennis Ritchier na década de 70 e foi escrita na linguagem assembly, em 89 a linguagem teve uma padronização pela ANSI chamada c89 que serviu para evitar incompatibilidade entre um compilador e outro, e futuramente surgiram novas padronizações adicionando novas bibliotecas e constantes, mesmo com uma padronização nem todos os compiladores segue esse padrão podendo gerar incompatibilidade entre compiladores que não segue esse padrão, uma das vantagens da linguagem C++ que ela é uma linguagem orientada a objeto (POO) diferente da linguagem C que não é orientada a objeto, os códigos da linguagem C pode ser compilado facilmente em um compilador C++ já os códigos da linguagem C++ não pode ser compilado em um compilador que é apenas para linguagem C, outra vantagem que sintaxe usada na linguagem C/C++ e muito parecida com outra linguagem mais recente entre elas C#, Java, Perl, Php entre outras, então quando alguém aprende a programar em C/C++ automaticamente vai saber o básico dessas outras linguagens, a linguagem C/C++ é uma linguagem compilada e existem compiladores para maioria dos sistemas operacionais embora muitas funções e bibliotecas só funcione em determinado sistema ou plataforma, outra vantagem de programar em C/C++ que ela é uma linguagem antiga então existem milhares de bibliotecas e códigos na internet para estudos, a linguagem C/C++ que segue o padrão ANSI é case-sensitive ou seja faz diferença escrever maiúsculo ou minúsculo, o compilador vai ler o código de cima para baixo da esquerda para direita.

1.1 – Linguagem de programação e linguagem da máquina

Muitos leigos usa um programa e mal sabe explicar o que é e como funciona um, um programa ou um software é uma sequencia de códigos na linguagem da máquina feito para executar tarefas sobre um sistema operacional, quando baixamos um programa na internet baixamos na verdade uma sequencia de códigos conhecida como binário (executável), esse binário são procedimentos que sistema operacional deve fazer (criar uma janela ou botoes, fica esperando eventos, ler e executar processo etc), dizemos quando um programa é alto nível quando ele funciona sobre uma camada acima de uma outra aplicação como por exemplo sistema operacional, e quando o programa é baixo nível quando ele funciona abaixo de determinada aplicação (também é comum ouvir que um programa de alto nível é mais perto do ser humano e um de baixo nível é mais perto da máquina), ate agora descobrimos que um programa é binário ou um código na linguagem da máquina e a linguagem da máquina é uma sequencia de códigos que sistema operacional vai executar, já um código de linguagem de programação é uma sequencia de códigos escrito pelo programador que será convertido para linguagem da máquina e essa conversão é chamada compilação, em uma compilação não é gerado o executável final é gerado um arquivo objeto e depois é linkado gerando o executável final, nem todas as linguagens de programação é compilada algumas existe um executável chamado interpretador que ler o código e executa esse procedimento, a vantagem das linguagens interpretadas que se tiver um interpretador para outras plataformas o script pode ser executado sem dificuldade diferente das linguagens compiladas que teria que compilar de novo para gerar o executável daquela plataforma, existe linguagens que pode ser compilada e interpretada ao mesmo tempo como por exemplo o java mesmo gerando o executável é necessário a máquina virtual para ler o executável (a vantagem do java que e o executável e multiplataforma embora seja extramente lento comparado as outras linguagens)

1.2 – Compiladores e IDE

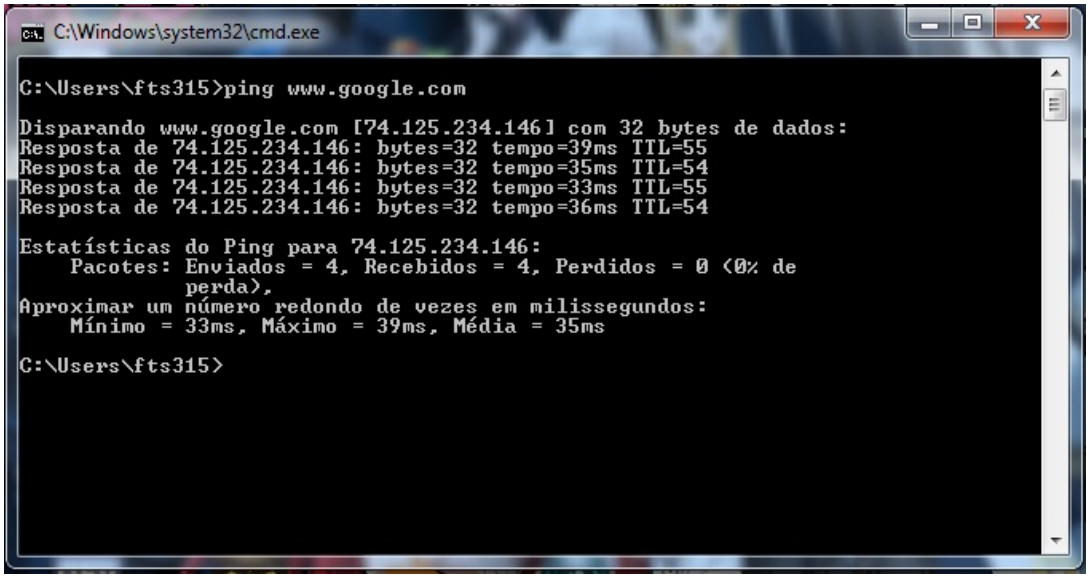
Como já sabemos os compiladores converte o código da linguagem de programação para linguagem da máquina, existem vários tipos de compiladores para linguagem c/c++ entre eles gcc, g++, borland c++, turbo c++, entre outros tipos de compiladores, além dos compiladores existem as IDE que são ambiente de programação, elas contem o compilador e outras ferramentas que ajuda o desenvolvedor podendo ser um editor de texto próprio que pode destacar os códigos, um debugger e checagem de sintaxe, ate desenvolvimento gráfico como janelas e botões, alguns exemplos de IDE para linguagem C/C++ seria o dev c++, code blocks, anjuta, borland c++ builder, microsoft visual studio, eclipse, netbeans, alguns desse compiladores e IDE são freewares outros são comerciais, abaixo podemos ver alguns sites onde pode ser baixado alguns compiladores e IDE

http://www.bloodshed.net/download.html (Dev C++)
http://www.codeblocks.org/downloads/binaries (Code Blocks)
http://www.anjuta.org/downloads (Anjuta)
http://www.superdownloads.com.br/download/72/borland-c-compiler/ (Borland c++)
http://www.eclipse.org/downloads/ (Eclipse)
https://netbeans.org/downloads/ (netbeans)
http://www.microsoft.com/visualstudio/ptb/downloads (microsoft visual studio)

1.3 – Interfaces CLI e GUI

As interfaces CLI são programas por base comando, ou seja programas que usa o terminal, embora seja raro um usuário final usar esses tipos de programas para um programador é crucial saber programar e usar tais programas, os programas com interface CLI é muito mais rápido e também gasta menos memória que programas com interfaces gráficas, a grande desvantagem que usuário final vai ter que saber manipular esses programas, para os usuários finais quanto mais fácil for o uso do programa melhor, porém quanto mais fácil o uso do programa para o usuário final mais

trabalhoso para o programador criá-lo, abaixo vemos uma imagem de um programa com interface CLI o ping do windows.



as interfaces gráficas são chamadas interface GUI, é mais comum você baixar um programa que tenha uma interface GUI do que CLI, os programas com interface GUI dá um visual melhor para programa se seu programa vai ser comercial, ou seja você pretende vender ele é recomendo fazer ele com interface GUI para chamar mais atenção para o usuário final, na criação de uma interface GUI os programadores deve estudar API gráfica como por exemplo GTK, abaixo temos uma imagem de um programa com interface GUI.

existem outros tipos de interfaces entre elas a de voz VUI que não abordarei nesse ebook.

2.0 – Terminal Windows e Linux

Quem olha a tela do prompt do windows pela primeira vez acaba falando algo do gênero “Mais que ….. é essa '-' ”, alguns nem acredita que isso era um sistema operacional a duas décadas atrás, mais é verdade era um sistema operacional antes mesmo do windows 3x o famoso e quase morto MSDOS, é raro achar alguém que use esse sistema ainda embora o uso do cmd também conhecido por prompt ou terminal seja bastante comum entre programadores e hackers, o que temos que entender do terminal que ele funciona a base de comando, ali podemos copiar, mover, deletar, criar

arquivo, ver diretório, executar programa, praticamente qualquer coisa que faríamos no sistema operacional no modo gráfico, o terminal do window e do linux são um pouco parecido mais tem sua diferença no terminal do windows os diretórios são separador por contra barra (\) já no linux por barra (/), no terminal do windows é mostrado no final o símbolo de maior (>) no do linux é a permissão do usuário logado que pode variar entre cifrão e sharps ($ ou #) indicando usuário normal ou root, no terminal do windows antes do último símbolo indica o diretório que ele está apontando no caso do linux antes do diretório que ele tá apontando tem uma arroba (@) e o usuário logado e a máquina, no terminal do windows os comandos podem ser separador por duas vezes o simbolo “e” comercial (&&) já no linux são por ponto e vírgula (;)

```
C:\users\fts315>
```

Terminal Windows

```
fts315@skynet /home/fts315$
```

Terminal Linux

2.1 – Mexendo com terminal do windows

A manipulação do terminal permite agilizar tarefas no sistema operacional ou simplesmente execução de programas por comando, no windows existe um terminal chamado prompt de comando esse terminal pode ser acessado apertando no botao “iniciar” do windows, depois em “Todos os programas”, “acessórios”, “prompt de comando”, ou simplesmente escrever “cmd.exe” em “pesquisar programa e arquivos” depois apertar enter ai vai abrir uma tela preta que é o terminal (esse modo só funciona no windows 7), você também poderia apertar “winkey + r” no desktop vai abrir uma janela chamada executar nela é só digitar “cmd.exe” e apertar em “ok”, a ultima forma de abrir o prompt é abrir ele manualmente na pasta dele que seria “meu computador”, a unidade que está instalado o sistema operacional (maioria das vezes na unidade c:\), diretório windows e depois diretório system32, e la dentro tem o executável cmd.exe (c:\windows\system32\cmd.exe), com o prompt aberto podemos ver que é algo assim

```
Microsoft Windows [versão 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. Todos os direitos reservados.

C:\Users\fts315>
```

a primeira parte escrita “Microsoft Windows … direitos reservado” podemos ignorar isso a única coisa importante ali é o diretório atual “C:\Users\fts315>” no caso o prompt esta apontando para o diretório que “fts315” (maioria das vezes o prompt aponta para a pasta do usuario logado), esse diretório “fts315” está dentro do diretório “Users” e o diretório “Users” esta dentro da unidade “c:\”, para executar comandos no prompt basta digitar o comando e apertar enter, só é possível executar programas pelo nome que esta dentro do diretório atual ou em uma variavel de ambiente se não é necessário digitar o endereço completo, o primeiro comando que vamos aprender será o “DIR” para exibir todos os arquivos e diretorio dentro do diretorio atual

```
C:\Users\fts315> dir

 Pasta de C:\Users\fts315

02/11/2013  05:45    <DIR>          .
02/11/2013  05:45    <DIR>          ..
17/08/2013  23:45    <DIR>          Contacts
```

```
08/11/2013  05:02    <DIR>          Desktop
02/11/2013  05:52    <DIR>          Documents
08/11/2013  00:04    <DIR>          Downloads
17/08/2013  23:45    <DIR>          Favorites
08/11/2013  05:24        14.155.776 NTUSER.DAT

C:\Users\fts315>
```

no caso ele listo mais arquivos e diretórios aqui porém eu coloquei apenas esses, podemos ver que ele mostra a data e hora que os programas foram criados, se o arquivo é um diretorio ou não caso seja ele mostra <DIR>, caso seja um arquivo ele mostra o tamanho em bits e por fim o nome do arquivo ou do diretório, também podemos listar o diretório pelo endereço

```
C:\Users\fts315> dir c:\

 Pasta de c:\

10/06/2009  19:42                24 autoexec.bat
29/09/2013  01:15    <DIR>          Borland
10/06/2009  19:42                10 config.sys
22/09/2013  06:41    <DIR>          cygwin
18/08/2013  01:32    <DIR>          Dev-Cpp
07/09/2013  02:07    <DIR>          Drivers
07/09/2013  02:07    <DIR>          Users

C:\Users\fts315>
```

para entrar em um diretório usamos o comando CD seguido do nome do diretório, no caso eu estou no diretório do usuário la dentro tem um diretório chamado Desktop que é nada mais nada menos que a área de trabalho daquele usuário

```
C:\Users\fts315> cd Desktop
C:\Users\fts315\Desktop>
```

também é possivel ir para determinado diretório apenas com endereço

```
C:\Users\fts315> cd c:\users
C:\Users>
```

para voltar um diretorio podemos usar CD e dois pontos

```
C:\Users\fts315\Desktop> cd ..
C:\Users\fts315>
```

para executar ou abrir um programa ou arquivo basta digitar nome do programa caso esteja no mesmo diretório que ele

```
C:\Users\fts315\Desktop> eof.exe
```

também e possível executar pelo endereço completo caso não esteja na mesma pasta

```
C:\Users> C:\Users\fts315\Desktop\eof.exe
```

porem o método acima pode travar esse terminal enquanto o programa estiver em execução para evitar isso podemos usar o comando START antes para usar outro prompt para aquele programa

```
C:\Users\fts315\Desktop> start eof.exe
```

caso o programa ou diretorio tenha espaço coloque entre aspas (isso serve para os outros comandos também)

```
C:\Users\fts315\Desktop> start "forum eof.exe"
```

para criar um diretorio podemos usar o comando MD seguido do nome do diretorio

```
C:\Users\fts315\Desktop> md fts
```

para remover um diretorio podemos usar o comando RD

```
C:\Users\fts315\Desktop> rd fts
```

para deletar um arquivo podemos usar DEL

```
C:\Users\fts315\Desktop> del fts.exe
```

para ler um arquivo usamos TYPE

```
C:\Users\fts315\Desktop> type fts.txt
```

para copiar um arquivo ou diretório podemos usar o comando COPY seguido do arquivo a ser copiado e do local para onde ele vai

```
C:\Users\fts315\Desktop> copy fts.txt c:\
```

existem muito outros comandos alem dos citados

2.2 – Mexendo com terminal do linux

A manipulação do terminal do linux é parecida com a do windows só mudando alguns comandos alem disso existe muito mais comandos para o linux do que no windows, embora exista o comando dir no linux é recomendado usar o comando ls

```
fts315@skynet:~/Desktop$ ls
amsn.desktop      fts.txt                RaidCall.desktop
script
fts315@skynet:~/Desktop$
```

podemos colocar a sintaxe -l para mostrar a permissão, o dono do arquivo, grupo, tamanho etc

```
fts315@skynet:~/Desktop$ ls -l
lrwxrwxrwx 1 fts315 fts315    36 Mai  3  2013 amsn.desktop -> /usr/share/applications/
-rw-r--r-- 1 fts315 fts315   297 Out 24 13:15 fts.txt
-rwxr-xr-x 1 fts315 fts315   259 Mai  9  2013 RaidCall.desktop
drwxr-xr-x 2 fts315 fts315   760 Out 15 06:21 script
fts315@skynet:~/Desktop$
```

o comando cd é a mesma coisa do windows

```
fts315@skynet:~/Desktop$ cd script
fts315@skynet:~/Desktop/script$
```

para voltar diretório com cd usamos dois pontos igual o windows

```
fts315@skynet:~/Desktop/script$ cd ..
fts315@skynet:~/Desktop$
```

para criar um diretório usamos mkdir seguido do nome do diretorio

```
fts315@skynet:~/Desktop$ mkdir fts
```

para deletar um diretorio usamos rmdir seguido do nome do diretorio

```
fts315@skynet:~/Desktop$ rmdir fts
```

para deletar um arquivo usamos rm

```
fts315@skynet:~/Desktop$ rm fts
```

para criar um arquivo usamos o comando touch

```
fts315@skynet:~/Desktop$ touch fts.txt
```

para copiar determinado arquivo usamos o comando cp

```
fts315@skynet:~/Desktop$ cp fts.txt script/fts.txt
```

para mover determinado arquivo usamos o comando mv

```
fts315@skynet:~/Desktop$ mv fts.txt script/fts.txt
```

para executar determinado programa usamos ponto e barra antes

```
fts315@skynet:~/Desktop$ ./fts
```

para ler determinado arquivo podemos usar o cat

```
fts315@skynet:~/Desktop$ cat fts.txt
```

para ficar com acesso root usamos o comando su porem esse comando pode pedir senha de root, se a senha estiver certo sera modificado o símbolo $ pelo #

```
fts315@skynet:~/Desktop$ su
Senha:
root@skynet:/home/fts315/Desktop#
```

também podemos usar o su para mudar de usuario, para isso basta digitar o usuario depois do su

```
fts315@skynet:~/Desktop$ su hack
Senha:
hack@skynet:/home/fts315/Desktop$
```

para conseguir acesso root temporário para determinado comando podemos usar o sudo seguido do comando

```
fts315@skynet:~/Desktop$ sudo rd fts.txt
```

podemos mudar a permissão com o comando chmod seguido da sintaxe -c e o codigo da permissão e por fim o arquivo (não ensinarei sobre permissão mesmo porque isso é um ebook de C/C++ e não de linux), no exemplo abaixo eu coloco a permissão rwxr-xr-x para o arquivo fts.txt

```
fts315@skynet:~/Desktop$ chmod -c 755 fts.txt
```

para montar uma unidade ou partição podemos usar o comando mount seguido da unidade depois uma pasta onde sera montado (dependendo do caso é necessario estar como root)

```
fts315@skynet:~/Desktop$ mount /dev/sdb1 /home/fts315/Desktop/hd2
```

para desmontar usamos o umount

```
fts315@skynet:~/Desktop$ umount /home/fts315/Desktop/hd2
```

3.0 – Sintaxe C/C++

Todo programa em C existe uma função principal e na maioria das vezes é a função é a main, nessa função vamos ter que definir um tipo de retorno e um argumento só que antes de tudo criamos um arquivo de texto com extensão .c e os programas C++ usa extensão .cpp, esse arquivo não pode ter formatação de texto então ele deve ser criado com um editor de texto puro tipo o notepad do window ou kwrite do linux ou pelas IDE, o tipo de retorno vem antes do nome da função no caso essa função principal necessita retornar um tipo inteiro então usamos int seguido do nome main, usamos parênteses para definir o argumento que será void (tipo vazio), por fim usamos chaves para criar o bloco também chamado de escopo

```
int main(void){}
```

a linguagem C/C++ não diferencia espaço e nem quebra de linhas o exemplo anterior e o próximo seria a mesma coisa para compilador

```
int                    main    (    void    )
{
}
```

sempre de uma quebra de linha a cada final de uma função e sempre de um espaço a cada novo escopo isso deixa o código mais organizado e legível isso se chama edentação ou recuo, ate agora criamos a função main definimos o argumento e o tipo de retorno mais não definimos o retorno, para isso usamos a palavra return seguido do valor de retorno que será 0, e para terminar usamos ponto e vírgula indicando fim do comando, esse ponto e vírgula é necessário sem ele o compilador vai achar que o próximo comando faz parte desse e vai dá erro na hora de compilar

```
int main(void)
{
    return 0;
}
```

quando o compilador achar o return ele automaticamente vai terminar a função mesmo se tiver outras coisas em baixo, para compilar caso você esteja usando uma IDE procure o botao ou menu (no caso o dev c++ você pode apertar F9), caso você queira compilar pelo terminal localize o arquivo e digite gcc seguido da source no caso vou compilar sintaxe03.c, depois usamos o comando -o seguido do nome do executável que será gerado (caso de um erro de um erro de não reconhecido como comando interno ou externo possivelmente seu compilador não está nas variáveis de ambiente ou você está usando outro tipo de compilador, outra coisa para compilar programas c++ pelo terminal usamos g++ no lugar de gcc)

```
C:\users\fts315\desktop> gcc sintaxe03.c -o exemplo.exe
```

depois de gerar o executável basta abrir ele, você deve ter notado que ele fecho isso porque ele não faz nada é um programa inútil kkkk.

3.1 – Bibliotecas da linguagem C/C++

Uma das vantagens da programação que você não precisa criar tudo já existem bibliotecas de funções prontas que é só declarar e usar, na linguagem C/C++ para declarar uma biblioteca header usamos a diretiva #include seguido da biblioteca entre o símbolo menor e maior, boa parte das vezes declaramos uma biblioteca no começo da source entre as primeiras linhas, existem diversas bibliotecas padrões que servem para diversos procedimentos entre eles mostrar algo na tela manipular eventos do sistema operacional, manipular texto, manipular o tempo, entre outras coisas, as bibliotecas mais usadas na linguagem C são “stdio.h” para manipular funções de entrada e saída de dados, “stdlib.h” tem diversas funções do sistema, “string.h” para manipular textos, “ctype.h” para manipular caracteres, time.h para manipular o tempo, stdbool.h para manipular lógica booleana, “windows.h” para manipulação de API no sistema operacional windows, “unistd.h” para manipulação de determinadas API do sistema linux/unix, “winsock2.h” para manipulação de rede do sistema windows, “sys/socket.h” para manipulação de rede do sistema linux, “dirent.h” para manipulação dos diretórios do sistema windows, “sys/dir.h” para manipulação dos diretórios do sistema linux, “sys/types.h” para manipulação de alguns tipos de dados no sistema linux, na linguagem C++ é possível usar essas bibliotecas porém existem bibliotecas que só funciona na linguagem C++ como por exemplo a “iostream” que seria uma melhoria da stdio.h para linguagem C++

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    return 0;
}
```

essas bibliotecas na maioria das vezes está em um diretório do compilador mais é possível colocar elas no mesmo diretório da source porém teria que colocar entre aspas duplas em vez de menor e maior na hora de declarar

```
#include <stdio.h>
#include "eof.h"

int main(void)
{
    return 0;
}
```

existem outros tipos bibliotecas além das header um exemplo são as DLL, quando uma biblioteca é compilada junto gerando o executável chamamos ela de biblioteca estática, e quando ela é chamada em tempo de execução (run-time) chamamos de biblioteca dinâmica, também seria possível criar suas próprias bibliotecas para reaproveitar parte do código, mais para frente vamos ver melhor isso

3.2 – Comentários

É sempre bom comentar o seu código para facilitar na leitura, os comentários também são usado nas primeiras linhas para descrever o código ou informação como o autor ou descrição do codigo, quando o compilador acha um comentário ele ignora aquele parte do código, um comentário tambem pode servir para excluir temporariamente um trecho do código assim permitindo você debugar, para comentar usamos duas vezes a barra (//) para comentar a linha toda, o que estiver depois vai ser o comentário

```
//forum: end of file
#include <stdio.h> //isso é uma biblioteca

//função principal
int main(void)
{
   // return 0; esse return foi excluido
}
```

além desse comentário de uma linha podemos comentar várias linhas com barra asterisco e para finalizar asterisco barra (/* e */), esse tipo de comentário pode ser usado dentro dos códigos sem afetá-los

```
/*
forum: end of file
ebook: c/c++
*/
```

```
#include <stdio.h>

int main(/* isso é um tipo vazio */ void )
{
    return /* vai retornar o numero 0 */ 0;
}
```

algumas IDE permite comentário de destaque, esses tipos de comentários são parecido com os comentários de várias linhas porém tem um asterisco a mais (/** e **/), algumas IDE permite destacar alguns parágrafos dentro desse tipo de comentário usando algumas representações como por exemplo o @Author: para destacar o autor do código

```
/**
@Author: end of file
**/

#include <stdio.h>

int main(void)
{
  return 0;
}
```

3.3 – Diferença da linguagem C e C++

A sintaxe da linguagem C/C++ são iguais a diferença que a linguagem C++ tem coisas novas como orientação a objeto, passagem de argumento de classes para as funções, alocação dinâmica de memória pelo operador new e liberação da memória pelo operador delete, umas estruturas novas como namespace e class, as constantes true e false não precisar da biblioteca stdbool.h, alguns tipos de dados novos como string, também é comum usar extensão “.c” para linguagem C e “.cpp” para linguagem C++

4.0 – Saída de dados (output)

Quando dizemos entrada (input) ou saida (output) estamos enviando ou recebendo dados de um ponto A para o ponto B, no caso temos que definir o marco zero que na maioria das vezes é próprio programa, ou seja uma saída de dados poderia ser algo que sai do programa para o monitor, ou do programa para impressora, ou do programa para o HD ou memória, já a entrada de dados é o oposto seria para dentro do programa podendo ser algo que escrevemos no teclado, movimento do mouse, ou alguma coisa do HD, esse termo input e output também é usado em outras áreas como por exemplo manutenção se você perguntar para técnico para ele definir oque é um teclado e um monitor ele certamente vai dizer que o monitor é um periférico de saída e o teclado um periférico de entrada, o termo usado para os dados enviado ou recebido são chamado de fluxo, stream ou buffer, e os textos na programação se chama string, uma string é um conjunto de caracteres que seria o mesmo que dizer que uma palavra é um conjunto de letras, toda string deve ficar entre aspas para o compilador não confundir ela com uma variável, a saída de dados mais usada será para o monitor um exemplo de função que mostra algo na tela é o printf e puts da biblioteca stdio.h, na linguagem c++ podemos usar o cout da biblioteca iostream, para usar o printf ou puts temos que declarar a biblioteca stdio.h, depois dentro do escopo da função principal usamos a função e passamos como argumento a nossa string que será mostrada na tela e por fim usamos ponto e vírgula para finalizar

```
#include <stdio.h>

int main(void)
{
    printf("usando o printf");
    return 0;
}
```

o uso da função puts é a mesma coisas que o printf

```
#include <stdio.h>

int main(void)
{
    puts("usando o puts");
    return 0;
}
```

também poderíamos usar várias vezes o printf ou puts, ou os dois juntos

```
#include <stdio.h>

int main(void)
{
    printf("usando o printf");
    printf("usando a segunda vez o printf XD");
    puts("usando o puts")
    return 0;
}
```

a diferença entre o puts e o printf, que o puts mostra apenas string já o printf permite formatar a saída para mostra números e outros tipos de dados, para saida de dados para monitor de um único carácter podemos usar putchar da biblioteca stdio.h

```
#include <stdio.h>

int main(void)
{
    putchar('f');
    putchar('t');
    putchar('s');
    return 0;
}
```

para usar o cout da linguagem c++ temos que colocar um namespace std antes da função ficando std::cout e usar duas vezes o menor seguido da string

```
#include <iostream>
```

```
int main(void)
{
    std::cout << "usando o cout";
    return 0;
}
```

também seria possível usar o printf ou puts na linguagem C++, porém é recomendado o uso da biblioteca cstdio no lugar de stdio.h

```
#include <cstdio>

int main(void)
{
    printf("cstdio e igual a stdio.h");
    puts("porem cstdio nao funciona na linguagem C so em C++");
    return 0;
}
```

4.1 – Evitando da janela fechar

Se você usa windows deve ter percebido que o programa executo e depois fecho automaticamente para evitar isso temos que pausar o terminal, para isso existem várias funções uma delas é o getch da biblioteca conio.h, essa é uma das funções de entrada de dados do teclado, para usar basta declarar sem nenhum argumento assim

```
#include <stdio.h>
#include <conio.h>

int main(void)
{
    printf("quero ver se voce fecha agora janela!!!");
    getch();
    return 0;
}
```

outra função é o system da biblioteca stdlib.h, para usar basta passar como argumento a string “pause”

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    printf("hello world!!!");
    system("pause");
    return 0;
}
```

4.2 – Formatação de string

Quando usamos duas vezes o printf ele mostras as duas strings na mesma linha, para mostrar em

linhas diferentes temos que usar um código de escape que represente a quebra de linha, os códigos de escapes começa com contra barra seguida de um carácter, para a quebra de linha usamos contra barra mais o carácter “n” (\n)

```
#include <stdio.h>

int main(void)
{
   printf("\n\n");
   printf("era uma vez uma gato com 7 vidas\n");
   printf("passo um carro 4x4 por cima dele,\n e ele morreu FIM!!!\n\n");
   return 0;
}
```

para representamos aspas usamos contra barra mais o aspas (\”),

```
#include <stdio.h>

int main(void)
{
   printf("fato - todos que fazem software faz \"programa\" \nmais nem todos que
fazem \"programa\" faz software");
   return 0;
}
```

para tabulação usamos contra barra mais o carácter “t” (\t), um bom uso para isso é criar banner para seu programa

```
#include <stdio.h>

int main(void)
{
   printf("======================\n");
   printf("\t EOF\n");
   printf("======================\n\n");
   return 0;
}
```

para emitir um som (beep) usamos contra barra mais o carácter “a” (\a), esse som sera emitido da placa mae e não da caixa de som

```
#include <stdio.h>

int main(void)
{
   printf("\a programa ligado");
   printf("\a\a programa desligado");
   return 0;
}
```

para representar o símbolo de porcentagem usamos duas vezes porcentagem (%%)

```
#include <stdio.h>

int main(void)
{
   printf("hoje eu estou 100%% bebado (zueira to nao)");
   return 0;
}
```

também existe outro tipo de código de escape para atribuição de variáveis na string, esse código de escape é o símbolo de porcentagem seguido do carácter que representa o tipo de dados que será mostrado, %d ou %i tipo inteiro, %f tipo float, %lf tipo double, %c tipo char, %s tipo string, %e ou %E representação do número em forma expoente, %x ou %X representação do número em forma hexadecimal, %p endereço de memória, %o representação do número em forma octal, além desses existem outros códigos de escape.

```
#include <stdio.h>

int main(void)
{
    printf("decimal: %d \n",315);
    printf("hexadecimal: %x \n",315);
    printf("octal: %o \n",315);
    printf("float: %f",3.15);
    return 0;
}
```

4.3 – Variáveis e constantes

Uma variável é uma alocação de memória que permite um armazenamento de dados na memória do computador, os dados armazenado nessas variáveis podem ser modificado ao decorrer do programa, as variáveis pode ter qualquer nome contando que seja nomes que comece com letras de “a” ate “z” ou underline, sem espaço e podendo ser maiúsculas ou minúsculas, porém por padrão é recomendado colocar todas as variáveis minúsculas e as constantes em maiúsculas para aumentar a legibilidade, podemos criar uma variável em 3 lugares diferente sendo eles dentro do escopo de uma função criada essas são as variáveis locais como por exemplo dentro da função principal (main), no argumento de uma função criada como por exemplo onde seria o void da função main, ou no lado de fora para as funções globais um exemplo depois dos includes, as variáveis só podem ser acessadas pelos códigos abaixo delas em termo lógico você só pode acessar uma variável depois de criar ela, as variáveis locais só pode ser acessadas dentro do escopo onde foram criadas e as variáveis globais pode ser acessada em qualquer escopo, para criar uma variável basta colocar o tipo de dado seguida do nome da variável, abaixo vemos um exemplo de 3 variáveis de tipo int

```
#include <stdio.h>

int var_global;

int main(int var_argumento)
{
    int var_local;
    return 0;
```

```
}
```

os tipos padrões da linguagem C/C++ são variáveis de tipo int que permite armazenar números inteiros, números quebrados são as variáveis tipo float, números quebrados grandes são variáveis tipo double, variáveis tipo carácter são as char, as strings na verdade são array do tipo char vamos aprender como armazenar elas mais para frente em vetores e matrizes a linguagem c++ existe o tipo string embora algumas funções costuma da erro com ele

```
#include <stdio.h>

int main(void)
{
    int inteiro;
    float quebrado;
    double quebrado_gigante;
    char caracter;
    return 0;
}
```

para atribuirmos os dados para as variáveis colocamos o nome da variável o símbolo de igual e o dado que vai ser armazenado

```
#include <stdio.h>

int main(void)
{
    int endoffile;
    float fts;
    endoffile = 315;
    fts = 3.15;
    return 0;
}
```

podemos usar o printf para exibir a variável colocando um vírgula depois da string e depois a variável

```
#include <stdio.h>

int main(void)
{
    int endoffile;
    float fts;
    endoffile = 315;
    fts = 3.15;
    printf("inteiro = %d \n",endoffile);
    printf("float = %f",fts);;
    return 0;
}
```

tambem poderia fazer assim

```
#include <stdio.h>

int main(void)
{
    int endoffile;
    float fts;
    endoffile = 315;
    fts = 3.15;
    printf("inteiro = %d \nfloat = %f",endoffile,fts);
    return 0;
}
```

uma variável pode mudar o valor ao decorrer do programa bastando atribuir o novo valor para ela

```
#include <stdio.h>

int main(void)
{
    int endoffile;
    endoffile = 315;
    printf("valor = %d",endoffile);
    endoffile = 100;
    printf("valor = %d",endoffile);
    return 0;
}
```

também e possível armazenar em uma variável o valor de outra variável

```
#include <stdio.h>

int main(void)
{
    int endoffile;
    int fts;
    fts = 315;
    endoffile = fts;
    printf("valor = %d",fts);
    return 0;
}
```

é possível iniciar um valor junto com variável, para isso basta atribuir o valor assim que declarar ela

```
#include <stdio.h>

int main(void)
{
    int endoffile = 315;
    printf("%d",endoffile);
    return 0;
}
```

quando tem mais de uma variável do mesmo tipo podemos criar elas juntas separando por vírgula

```
#include <stdio.h>

int main(void)
{
    int endoffile = 315, fts;
    return 0;
}
```

uma variável do tipo char deve armazenar carácter entre aspas simples, a aspas dupla indica string e uma string deve ser armazenada dentro de uma array do tipo char

```
#include <stdio.h>

int main(void)
{
    char letra1 = 'e';
    char letra2 = 'o';
    char letra3 = 'f';
    printf("caracter = %c%c%c",letra1,letra2,letra3);
    return 0;
}
```

já na linguagem C++ existe o tipo string que armazena strings

```
#include <iostream>

int main(void)
{
    std::string eof;
    eof = "isso ai";
    std::cout << eof;
    return 0;
}
```

uma constante é uma alocação de memória que não muda o valor ao decorrer do programa, para criar uma constante criamos uma variável e adicionamos a palavra const antes do tipo, porém também é necessário atribuir o valor junto

```
#include <stdio.h>

int main(void)
{
    const float PI = 3.141592;
    printf("o valor de PI e igual a %f",PI);
    return 0;
}
```

outra forma de criar constates é usar a diretiva #define, essa diretiva substitui toda parte pelo código correspondente, para usar essa diretiva basta declarar #define colocar um nome e o valor, quando o

copilador achar a definição do define ele automaticamente vai mudar o codigo pelo valor do define

```
#include <stdio.h>
#define PI 3.141592

int main(void)
{
    printf("o valor de PI e igual a %f",PI);
    return 0;
}
```

o código anterior seria equivalente a esse

```
#include <stdio.h>

int main(void)
{
    printf("%f",3.141592);
    return 0;
}
```

mais para frente vamos ver outros exemplos de como usar a diretiva define para construir parte do código

4.4 – Vetores e matrizes

na criação de muitas variáveis para o mesmo propósito é recomendado criar uma array, uma array é uma variável como muitos segmentos, existe dois tipos de array os vetores e as matrizes, os vetores permite apenas um seguimento, já as matrizes cada segmento é um novo segmento que pode ter novos segmentos, para explicar melhor imagine que você tenha que criar 2 variáveis para definir altura e largura, no lugar dessas duas variáveis podemos cria 1 array com dois espaços, para criar uma array é a mesma coisa que criar uma variável basta colocar o tipo um nome, a diferença que temos que abrir e fechar colchetes e dentro dele colocar o tamanho no caso do exemplo vai ser 2

```
#include <stdio.h>

int main(void)
{
    int tamanho[2];
    return 0;
}
```

no exemplo acima usamos um vetor com 2 tamanhos, para armazenar o valor nele basta atribuir o valor na variável mudando o número do colchete (index) que começa no com o número 0

```
#include <stdio.h>

int main(void)
{
    int tamanho[2];
    tamanho[0] = 300;
```

```
    tamanho[1] = 15;
    printf("%d \n",tamanho[0]);
    printf("%d",tamanho[1]);
    return 0;
}
```

para iniciar os valores com a array usamos chaves e separamos o valor com vírgula

```
#include <stdio.h>

int main(void)
{
    int tamanho[2] = {300,15};
    printf("%d \n",tamanho[0]);
    printf("%d",tamanho[1]);
    return 0;
}
```

também é possível não definir o tamanho, porém só funciona para as array com valores iniciados, dessa maneira a array vai ter exatamente o tamanho igual a quantidade de valores

```
#include <stdio.h>

int main(void)
{
    int tamanho[] = {300,15};
    printf("%d\n%d",tamanho[0],tamanho[1]);
    return 0;
}
```

podemos criar strings dessa maneira, uma string deve ter o tamanho maior para adicionar o codigo de escape '\0' no final

```
#include <stdio.h>

int main(void)
{
     char forum[4] = {'e','o','f'};
     printf("%s",forum);
     return 0;
}
```

se atribuírmos os carácteres manualmente temos que adicionar o código de escape nulo manualmente também

```
#include <stdio.h>

int main(void)
{
     char forum[4];
```

```
    forum[0] = 'e';
    forum[1] = 'o';
    forum[2] = 'f';
    forum[3] = '\0';
    printf("%s",forum);
    return 0;
}
```

o codigo de escape nulo é equivalente ao codigo hexadecimal 0x00

```
#include <stdio.h>

int main(void)
{
    char forum[4];
    forum[0] = 'e';
    forum[1] = 'o';
    forum[2] = 'f';
    forum[3] = 0x00;
    printf("%s",forum);
    return 0;
}
```

uma forma mais simples de atribuir uma string a uma array é iniciar ela junto, não tem como atribuir uma string para uma array sem ser carácter por carácter

```
#include <stdio.h>

int main(void)
{
    char forum[4] = "eof";
    printf("%s",forum);
    return 0;
}
```

também podemos ler os carácteres de uma array

```
#include <stdio.h>

int main(void)
{
    char forum[4] = "eof";
    printf("%c %c",forum[0],forum[2]);
    return 0;
}
```

as matrizes também chamadas array multidimensional armazena outro segmento ao invés dos valores, a vantagem que permite armazenar uma espécie de array dentro da outra, para criar uma matriz basta usar outros colchetes

```
#include <stdio.h>

int main(void)
{
    int matriz[2][3];
    return 0;
}
```

para atribuirmos basta atribuir ao último colchete

```
#include <stdio.h>

int main(void)
{
    int matriz[2][3];
    matriz[0][0] = 315;
    matriz[0][1] = 50;
    matriz[0][2] = 30;
    matriz[1][0] = 900;
    matriz[1][1] = 630;
    matriz[1][2] = 555;
    return 0;
}
```

para iniciar os valores na array colocamos outras chaves dentro correspondente a quantidade de segmentos

```
#include <stdio.h>

int main(void)
{
    int matriz[2][3] = {{315,50,30},{900,630,555}};
    return 0;
}
```

podemos criar quantas index a gente quiser

```
#include <stdio.h>

int main(void)
{
    int matriz[10][10][10][10] = {{{{315,100}}}};
    printf("%d \n",matriz[0][0][0][0]);
    printf("%d",matriz[0][0][0][1]);
    return 0;
}
```

4.5 – Ponteiro e alocação dinâmica e estáticas

todas as variáveis existe um endereço de memória onde elas estão armazenadas, os ponteiros permite acessar e manipular esses endereços, a vantagem de um ponteiro que você pode manipular

variáveis locais fora do seu escopo, para criar um ponteiro basta criar uma variável e colocar asterisco antes do nome

```
#include <stdio.h>

int main(void)
{
    int *ponteiro;
    return 0;
}
```

com ponteiro criado basta atribuir o endereço de memória de um variável a ele, para isso atribuímos o endereço da variável colocando o símbolo & seguida do nome da variável

```
#include <stdio.h>

int main(void)
{
    int variavel = 315;
    int *ponteiro;
    ponteiro = &variavel;
    return 0;
}
```

para exibir o valor onde esta apontando basta colocar um asterisco antes do nome do ponteiro, para exibir apenas o endereço basta colocar sem asterisco

```
#include <stdio.h>

int main(void)
{
    int variavel = 315;
    int *ponteiro;
    ponteiro = &variavel;
    printf("valor: %d \n",*ponteiro);
    printf("endereco: %d",ponteiro);
    return 0;
}
```

para mudar o valor onde ele tá apontando basta colocar asterisco (no caso o asterisco vai acessar a variável e sem ele vai acessar o ponteiro)

```
#include <stdio.h>

int main(void)
{
    int variavel = 315;
    int *ponteiro;
    ponteiro = &variavel;
    *ponteiro = 100;
    printf("%d",variavel);
```

```
    return 0;
}
```

também podemos criar ponteiro para ponteiro, para isso basta colocar um novo asterisco a cada novo ponteiro e atribuir o endereço de memória dos outros ponteiros

```
#include <stdio.h>

int main(void)
{
    int variavel = 315;
    int *ponteiro1, **ponteiro2, ***ponteiro3;
    ponteiro1 = &variavel;
    ponteiro2 = &ponteiro1;
    ponteiro3 = &ponteiro2;
    printf("%d",***ponteiro3);
    return 0;
}
```

as vezes é necessário alocar um espaço na memória e depois liberá-la manualmente esse tipo de alocação é chamado alocação dinâmica de memória, diferente das variáveis locais que o espaço é liberado apenas quando a função for destruída e das variáveis globais que só é liberado quando o programa for finalizado as dinâmicas pode ser liberada a qualquer momento, para criar uma alocação dinâmica usamos a função malloc da biblioteca stdlib.h, a função malloc aloca uma quantidade especifica e retorna o endereço de memória para isso se faz necessário criar um ponteiro do tipo especifico para receber o endereço, na função malloc usamos outra função que é o sizeof que retorna o tamanho exato do tipo, também temos que passamos como argumento o tipo para o sizeof

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    int *variavel_dinamica = malloc(sizeof(int));
    *variavel_dinamica = 315;
    printf("%d",*variavel_dinamica);
    return 0;
}
```

em alguns compiladores é necessário a conversão de typecast para o tipo do ponteiro, também vamos aprender typecast mais para frente, para liberar o endereço o espaço usamos a função free da biblioteca stdlib.h e passamos como argumento o ponteiro que tem o endereço de memória

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    int *variavel_dinamica = malloc(sizeof(int));
    *variavel_dinamica = 315;
```

```
    printf("%d",*variavel_dinamica);
    free(variavel_dinamica);
    return 0;
}
```

também é possível criar vetores pelo malloc basta multiplicar pela quantidade desejada

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    int *variavel_dinamica = malloc(sizeof(int) * 2);
    variavel_dinamica[0] = 315;
    variavel_dinamica[1] = 100;
    printf("%d \n",variavel_dinamica[0]);
    printf("%d",variavel_dinamica[1]);
    free(variavel_dinamica);
    return 0;
}
```

também é possível criar matrizes com malloc, embora seja um pouco complicado ter que alocar espaço a cada segmentos

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    int **variavel_dinamica = malloc(sizeof(int) * 2);
    variavel_dinamica[0] = malloc(sizeof(int) * 2);
    variavel_dinamica[1] = malloc(sizeof(int) * 2);
    variavel_dinamica[0][0] = 315;
    variavel_dinamica[1][0] = 100;
    printf("%d \n",variavel_dinamica[0][0]);
    printf("%d",variavel_dinamica[1][0]);
    free(variavel_dinamica);
    return 0;
}
```

o exemplo acima seria o mesmo que criar uma array “int variavel_dinamica[2][2]”, também existe a função calloc da biblioteca stdlib.h que permite alocar uma determinada quantidade de vezes um espaço ela é equivalente ao malloc multiplicado, para usar ele passamos como argumento a quantidade de vezes que vai alocar e tamanho da alocação

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    int *variavel_dinamica = calloc(2,sizeof(int));
```

```
    variavel_dinamica[0] = 315;
    variavel_dinamica[1] = 100;
    printf("%d \n",variavel_dinamica[0]);
    printf("%d",variavel_dinamica[1]);
    free(variavel_dinamica);
    return 0;
}
```

quando precisamos realocar um espaço podemos usar a função realloc da biblioteca stdlib.h, nela passamos como argumento o ponteiro que vai ser realocado e o novo tamanho

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    int *variavel_dinamica = malloc(sizeof(int) * 5);
    realloc(variavel_dinamica,sizeof(int));
    *variavel_dinamica = 315;
    printf("%d \n",*variavel_dinamica);
    free(variavel_dinamica);
    return 0;
}
```

na linguagem C++ podemos usar o operador new para alocar um determinado tipo ou estrutura na memória e o operador delete para liberar

```
#include <iostream>

int main(void)
{
    int *variavel_dinamica = new int;
    *variavel_dinamica = 315;
    std::cout << *variavel_dinamica;
    delete variavel_dinamica;
    return 0;
}
```

também é possível alocar array pelo operador new

```
#include <iostream>

int main(void)
{
    int *variavel_dinamica = new int(3);
    variavel_dinamica[0] = 315;
    std::cout << variavel_dinamica[0];
    delete[] variavel_dinamica;
    return 0;
}
```

existe as alocações estáticas, essas alocações permite armazenar um valor em uma variável local que não será destruída junto com a função essas variáveis são mais utilizadas em funções recursivas, para tornar uma variável estática basta usar o operador static antes do tipo da variável

```
#include <stdio.h>

int main(void)
{
    static int variavel;
    return 0;
}
```

4.6 – Entrada de dados (input)

já vimos diversos modos de saídas de dados para o monitor agora vamos listar alguns de entrada de dados do teclado, a função puts da biblioteca stdio.h mostra uma string na tela já a função gets armazena algo que digitarmos do teclado em uma array do tipo char, para usar essa função basta passar como argumento a array onde ela vai armzenar, quando o programa chegar nessa parte o terminal vai ficar travado esperando o usuario digitar e depois apertar enter, essa função só permite armazenar strings e deve ser usada com muito cuidado evitando uma falha conhecida como buffer overflow que consiste do usuário adicionar dados maior que o tamanho da array assim acessando outro endereço de memória

```
#include <stdio.h>

int main(void)
{
    char nome[100];
    gets(nome);
    puts(nome);
    return 0;
}
```

outra forma de armazenar em variáveis especificas é usar o scanf da biblioteca stdio.h, para usar essa biblioteca temos que definir o tipo para isso usamos as mesmas formatações que o printf e depois o endereço de memória da variável que vamos armazenar, a desvantagem do scanf que armazenamento de string ele não consegue capturar o espaço então é recomendado usar o gets para string e o scanf para os outros tipos

```
#include <stdio.h>

int main(void)
{
    int idade;
    scanf("%d",&idade);
    printf("%d",idade);
    return 0;
}
```

uma função de entrada que já usamos é o getch ou getchar da biblioteca conio.h, para usar essa função basta atribuir ela a uma variável do tipo char e ela apenas armazena um carácter, uma

pequena observação essa função não tem no linux

```
#include <stdio.h>
#include <conio.h>

int main(void)
{
    char letra;
    letra = getch();
    printf("%c",letra);
    return 0;
}
```

O getchar é parecido com a função anterior a diferença que você tem que apertar enter

```
#include <stdio.h>
#include <conio.h>

int main(void)
{
    char letra;
    letra = getchar();
    printf("%c",letra);
    return 0;
}
```

outra forma de entrar com dados para seu programa é usar passagem por argumento na função principal o famoso command line, para isso basta criar duas variáveis uma inteira que será um contador e o outro um ponteiro do tipo char, é comum para os programadores em C/C++ usa o nome argc e argv, depois basta usar as index do argv na ordem que entro com os comandos para o programa, se você usa o comando “c:\> programa.exe –help” o programa.exe seria o argv[0] e o –help é o argv[1]

```
#include <stdio.h>

int main(int argc, char **argv)
{
    printf("%s \n",argv[0]);
    printf("%s",argv[1]);
    return 0;
}
```

existe um outro argumento no exemplo anterior que seria outro ponteiro para as variáveis de ambiente

```
#include <stdio.h>

int main(int argc, char **argv, char **argp)
{
    printf("%s\n",argp[0]);
    printf("%s",argp[1]);
```

```
    printf("%s",argp[2]);
    return 0;
}
```

na linguagem c++ podemos usar cin da biblioteca iostream para armazenar um valor ou string na em uma variável, para usar o cin temos que usar o namespace std igual ao cout, usar operador maior duas vezes seguido da variável

```
#include <iostream>

int main(void)
{
    char nome[100];
    std::cin >> nome;
    std::cout << nome;
    return 0;
}
```

4.7 – Manipulação de string

A biblioteca mais usada para manipular uma string é a biblioteca string.h, ela existem dezenas de funções uma delas é strlen que retorna a quantidade de carácter que tem uma string, para usar essa função basta passar como argumento a string

```
#include <stdio.h>
#include <string.h>

int main(void)
{
    int tam;
    char eof[100] = "forum eof";
    tam = strlen(eof);
    printf("%d",tam);
    return 0;
}
```

com a função strrev podemos inverter a string

```
#include <stdio.h>
#include <string.h>

int main(void)
{
    int tam;
    char eof[100] = "forum eof";
    strrev(eof);
    printf("%s",eof);
    return 0;
}
```

com a função strupr transformamos todos os carácteres em maiúsculos

```
#include <stdio.h>
#include <string.h>

int main(void)
{
    int tam;
    char eof[100] = "forum eof";
    strupr(eof);
    printf("%s",eof);
    return 0;
}
```

com a função strlwr transformamos todos os carácteres em minúsculos

```
#include <stdio.h>
#include <string.h>

int main(void)
{
    int tam;
    char eof[100] = "FORUM EOF";
    strlwr(eof);
    printf("%s",eof);
    return 0;
}
```

a função strcpy copia uma string para uma array, para usar ela temos que passar como argumento a array e depois a string a ser armazenada

```
#include <stdio.h>
#include <string.h>

int main(void)
{
    char eof[100];
    strcpy(eof,"forum eof");
    printf("%s",eof);
    return 0;
}
```

só lembrando que não e possível atribuir uma string em uma array diretamente se precisarmos atribuir string a uma array é recomendado usar o strcpy

```
#include <stdio.h>
#include <string.h>

int main(void)
{
    char eof[100];
    char fts[100] = "hfts315"; //é valido iniciar a string junto com a array
```

```
    eof = "forum eof"; //não é valido atribuir a string diretamente para array
    eof[0] = 'f'; //é valido atribuir caracter por caracter
    return 0;
}
```

para juntar (concatenar) duas strings em uma array podemos usar a função strcat, o uso dessa função é parecida com a função strcpy

```
#include <stdio.h>
#include <string.h>

int main(void)
{
    char eof[100] = "forum ";
    char fts[100] = "eof";
    strcat(eof,fts);
    printf("%s",eof);
    return 0;
}
```

com a função strstr dá para localizar uma ocorrência de string dentro de uma string ou seja uma sub string, para usar essa função basta passar como argumento a string onde vamos procurar e a outra string que será pesquisada, também temos que atribuir o retorno da função para um ponteiro do tipo char

```
#include <stdio.h>
#include <string.h>

int main(void)
{
    char eof[100] = "visitem o forum eof";
    char *ponteiro;
    ponteiro = strstr(eof,"forum");
    printf("%s",ponteiro);
    return 0;
}
```

Podemos usar a função strchr para localizar a primeira ocorreria de um caractere, o uso dessa função é parecida com strstr

```
#include <stdio.h>
#include <string.h>

int main(void)
{
    char eof[100] = "visitem o forum eof";
    char *ponteiro;
    ponteiro = strstr(eof,'f');
    printf("%s",ponteiro);
    return 0;
}
```

Tanto a função strstr e a função strchr podemos exibir a quantidade de caracteres ate a ocorrência, para isso basta subtrair o ponteiro pela string

```
#include <stdio.h>
#include <string.h>

int main(void)
{
    char eof[100] = "visitem o forum eof";
    char *ponteiro;
    ponteiro = strstr(eof,"forum");
    printf("%d",ponteiro - eof);
    return 0;
}
```

para sabe se uma string é idêntica a outra string usamos a função strcmp, essa função compara todos os caracteres das duas string caso seja idêntico vai retornar o número 0, caso a primeira string seja maior ira retorna um número positivo, caso a segunda seja maior ira retorna negativo, essa função será muito usada na estrutura de condição para comparar duas strings

```
#include <stdio.h>
#include <string.h>

int main(void)
{
    char eof[100] = "visitem o forum eof";
    char fts[100] = "visitem o forum eof";
    int lol;
    lol = strcmp(eof,fts);
    printf("%d",lol);
    return 0;
}
```

podemos usar a função sprintf da biblioteca stdio.h para armazenar uma string formatada em uma array, o uso dessa função e parecida com o printf porém temos que definir a array que será armazenada como argumento antes

```
#include <stdio.h>

int main(void)
{
    char eof[100];
    int valor = 315;
    sprintf(eof,"numero = %d",valor);
    printf("%s",eof);
    return 0;
}
```

4.8 – Manipulação de carácter

Como já sabemos cada string é um conjunto de caracteres ou uma array do tipo char que pode ser manipulada pelo posição dela, podemos manipular esses caracteres um a um usando algumas bibliotecas entre elas ctype.h, uma das funções dessa biblioteca é o toupper que converte o carácter para maiúsculo para usar ela basta passar como argumento o caracter e pegar o retorno dela convertida

```
#include <stdio.h>
#include <ctype.h>

int main(void)
{
    char eof = 'f';
    eof = toupper(eof);
    printf("%c",eof);
    return 0;
}
```

Tambem existe a função tolower da biblioteca ctype.h que faz o oposto da anterior

```
#include <stdio.h>
#include <ctype.h>

int main(void)
{
    char eof = 'F';
    eof = tolower(eof);
    printf("%c",eof);
    return 0;
}
```

podemos usar a função isdigit da biblioteca ctype.h para retornar um valor maior que 0 caso seja um carácter ('0' ate '9') e retorna o valor 0 caso não seja nenhum daqueles caracter

```
#include <stdio.h>
#include <ctype.h>

int main(void)
{
    char eof = '5';
    int fts;
    fts = isdigit(eof);
    printf("%d",fts);
    return 0;
}
```

podemos usar a função isxdigit da biblioteca ctype.h para retornar um valor maior que 0 caso seja um carácter hexadecimal ('0' ate '9' e 'a' ate 'f') e retorna o valor 0 caso não seja nenhum daqueles carácter

```
#include <stdio.h>
#include <ctype.h>
```

```
int main(void)
{
    char eof = 'f';
    int fts;
    fts = isxdigit(eof);
    printf("%d",fts);
    return 0;
}
```

podemos usar a função isalnum da biblioteca ctype.h para retornar um valor maior que 0 caso seja um carácter alfa numérico ('0' ate '9' e 'a' ate 'z') e retorna o valor 0 caso não seja nenhum daqueles carácter

```
#include <stdio.h>
#include <ctype.h>

int main(void)
{
    char eof = 'z';
    int fts;
    fts = isalnum(eof);
    printf("%d",fts);
    return 0;
}
```

podemos usar a função isascii da biblioteca ctype.h para retornar um valor maior que 0 caso seja um codigo ascii (00 ate 128) e retorna o valor 0 caso não seja nenhum daquele codigo

```
#include <stdio.h>
#include <ctype.h>

int main(void)
{
    char eof = 'd';
    int fts;
    fts = isascii(eof);
    printf("%d",fts);
    return 0;
}
```

podemos usar a função istoupper da biblioteca ctype.h para retornar um valor maior que 0 caso seja um carácter seja maiúsculo e retorna o valor 0 caso não seja

```
#include <stdio.h>
#include <ctype.h>

int main(void)
{
    char eof = 'A';
    int fts;
```

```
    fts = istoupper(eof);
    printf("%d",fts);
    return 0;
}
```

podemos usar a função istolower da biblioteca ctype.h para retornar um valor maior que 0 caso seja um carácter seja minusculo e retorna o valor 0 caso não seja

```
#include <stdio.h>
#include <ctype.h>

int main(void)
{
    char eof = 'a';
    int fts;
    fts = istolower(eof);
    printf("%d",fts);
    return 0;
}
```

5.0 – Lógica aritmética

O ponto forte da programação é a lógica condicional e aritmética, porque se a gente perceber o computador só processa dados e esses dados no final são números binários, ou seja um simples movimento do mouse ou ate mesmo pixel na tela são apenas sequencias de 1 e 0, a linguagem C/C++ permite manipular números da mesma forma que faríamos com uma caneta e papel porém milhares de vezes rápido e sem erros (só lembrando uma máquina nunca erra e nunca cansa, e pode processar números milhares de vezes em segundos), as operações mais usadas na matemática são soma, subtração, multiplicação e divisão, boa parte das demais operações pode ser feitas de alguma forma por essas 4 operações, uma das bibliotecas mais usadas é a math.h embora as operações citadas anteriormente não precise dessa biblioteca, alguns símbolos que são usado na matemática deve ser substituído por alguns usados na informática como por exemplo na multiplicação usamos o asterisco (*) e na divisão a barra (/), na adição usamos o mais (+) e na subtração usamos o hífen (-), podemos fazer qualquer uma dessas operações e atribuir o resultado a uma variável

```
#include <stdio.h>

int main(void)
{
   int soma, subt, mult, divi;
   soma = 3 + 5;
   subt = 10 - 5;
   mult = 2 * 8;
   divi = 10 / 3;
   printf("%d \n",soma);
   printf("%d \n",subt);
   printf("%d \n",mult);
   printf("%d",divi);
   return 0;
}
```

só lembrando que algumas operações como divisão acaba sobrando número por exemplo 10/3 sobra 1 esse número se chama resto ou modulo, para a gente conseguir o modulo de uma divisão usamos o operador porcentagem (%) no lugar da barra

```
#include <stdio.h>

int main(void)
{
    int modulo = 10 % 3;
    printf("%d",modulo);
    return 0;
}
```

também podemos fazer mais de uma operação assim

```
#include <stdio.h>

int main(void)
{
    int modulo = 10 + 3 - 5;
    printf("%d",modulo);
    return 0;
}
```

porem se houver multiplicação/divisão entre as operações de soma/subtração ele vai fazer primeiro as de multiplicação/divisão, então o resultado de 2+5*2 = 12 ao invés do resultado 14

```
#include <stdio.h>

int main(void)
{
    int modulo = 2 + 5 * 2;
    printf("%d",modulo);
    return 0;
}
```

para evitar isso colocamos entre parênteses as operações menos significativa para ser feito primeiro

```
#include <stdio.h>

int main(void)
{
    int modulo = (2 + 5) * 2;
    printf("%d",modulo);
    return 0;
}
```

o exemplo anterior também poderia ser feito assim

```
#include <stdio.h>
```

```
int main(void)
{
    int modulo = ((2 + 5) * 2);
    printf("%d",modulo);
    return 0;
}
```

é possível fazer operações com uma variável e atribuir a ela mesmo

```
#include <stdio.h>

int main(void)
{
    int eof = 10;
    eof = eof + 5;
    printf("%d",eof);
    return 0;
}
```

quando fazemos uma operação matemática para atribuir a mesma variável podemos usar o símbolo da operação seguida do símbolo de igual e o número isso seria uma sobrecarga de operadores, o exemplo abaixo seria o mesmo que o anterior

```
#include <stdio.h>

int main(void)
{
    int eof = 10;
    eof += 5;
    printf("%d",eof);
    return 0;
}
```

também é possível incrementar ou decrementar um número de um variável adicionado dois símbolos de mais ou dois símbolos de menos na frente da variável

```
#include <stdio.h>

int main(void)
{
    int x = 300, y = 100;
    x++;
    y--;
    printf("x = %d \ny = %d",x,y);
    return 0;
}
```

é possível fazer o mesmo adicionando os símbolos de ++/-- antes da variável, a diferença e como o compilador vai interpretar a incrementação naquele momento vamos ver melhor a diferença na parte de estrutura de repetição

```
#include <stdio.h>

int main(void)
{
    int x = 300, y = 100;
    ++x;
    --y;
    printf("x = %d \ny = %d",x,y);
    return 0;
}
```

com a função pow da biblioteca math.h podemos retornar a pontencia de um número, para usar ela basta passar como argumento o número e o outro número que vai elevar a potência

```
#include <stdio.h>
#include <math.h>

int main(void)
{
    int eof = pow(3,2);
    printf("%d",eof);
    return 0;
}
```

para saber a raiz quadrada basta usar a função sqrt e passar como argumento o número depois pega o retorno

```
#include <stdio.h>
#include <math.h>

int main(void)
{
    float eof = sqrt(3);
    printf("%f",eof);
    return 0;
}
```

para retornar o valor absoluto de um número usamos a função abs

```
#include <stdio.h>
#include <math.h>

int main(void)
{
    int eof = abs(315);
    printf("%d",eof);
    return 0;
}
```

com a função floor ele retorna o número com a parte quebrada zerada (3.15 = 3.0)

```
#include <stdio.h>
#include <math.h>

int main(void)
{
    float eof = floor(3.15);
    printf("%f",eof);
    return 0;
}
```

com a função ceil a agente trunca o número incrementado mais 1 (3.15 = 4.0)

```
#include <stdio.h>
#include <math.h>

int main(void)
{
    float eof = ceil(3.15);
    printf("%f",eof);
    return 0;
}
```

podemos ver que quando usamos variaveis do tipo float ele não formata os zeros deixando um número grande (3.15 = 3.150000), para formatar a saída do número float podemos colocar a quantidade máxima que ele vai exibir junto ao código de escape %f, para fazer isso antes do “f” colocamos um ponto seguido do número que vai ser exibido no meu caso o número quebrado é o 15 então são apenas dois números ficando %.2f

```
#include <stdio.h>

int main(void)
{
    float eof = 3.15;
    printf("%.2f",eof);
    return 0;
}
```

existem outras funções da biblioteca math.h porém não vamos nos aprofundar muito

5.1 – Logica booleana

Esse tipo de lógica diz se determinada ação é verdadeira ou falsa, essa ação poderia ser qualquer coisa deis de uma variável criada ou não, se determinado carácter é existente em uma string ou não, se o programa usa interface GUI ou não, qualquer coisa pode ser definida como uma logica booleana por exemplo “você está lendo esse ebook, sim ou nao?”, a lógica boolena não consiste em saber como aconteceu e sim se aconteceu e depois retorna sim ou não (podemos dizer que a logica booleana é o yin yang da programação), para converter o true/false para a programação usamos o número 1 e 0, o número 1 é true e o número 0 é false embora qualquer número acima de 1 também é true e qualquer número abaixo de 0 é false

```
#include <stdio.h>

int main(void)
{
    int v = 5, f = -5;
    printf("verdadeiro = %d \n",v);
    printf("falso = %d",f);
    return 0;
}
```

existe um tipo de variável que não é padrão na linguagem C porém ele se torna padrão na linguagem C++, esse tipo é o bool e para usar ele na linguagem C temos que declarar a biblioteca stdbool.h, ja na linguagem C++ não é necessário declarar

```
#include <stdio.h>
#include <stdbool.h>

int main(void)
{
    bool verdade = 1;
    bool falso = 0;
    return 0;
}
```

na biblioteca stdbool.h também temos a constante true e false que é equivalente ao valor 1 e 0

```
#include <stdio.h>
#include <stdbool.h>

int main(void)
{
    bool verdade = true;
    bool falso = false;
    return 0;
}
```

embora o uso do true/false seja o mesmo que usar 1/0 usando o true/false deixa o programa mais legível

5.2 – Operação bit a bit

As operações bit a bit manipula cada bit de um byte, 1 bit é a menor quantidade de dados que computador consegue ler e um bit só tem duas possibilidades 1 e 0, então não existem muitas coisas que poderia se fazer com apenas um bit (tem a lógica booleana kkkkkkk), um único bit não dá para representar todos os caracteres do alfabeto para isso se fez necessário criar uma sequencia de bits que chamamos de bytes, um byte é uma sequencia de 8 bits que é equivalente a 256 possibilidades, por exemplo a letra “a” é um byte que tem a representação decimal do número 97 e em binário do código 01100001 cada número dali é um bit, as operações bit a bit manipula o bit alterando ele de acordo com regras especificas, a primeira operação é OR (ou), a operação bit a bit OR manipula dois bytes comparando um com o outro cada bit e gerando um 3º byte, se um dos dois bits for verdadeiro ou seja 1 o resultado do bit será verdadeiro

```
01100001 = a (97)
01100010 = b (98)
--------
01100011 = c (99)
```

para se fazer a operação bit a bit OR na linguagem C/C++ usamos o operador pipe (|)

```
#include <stdio.h>

int main(void)
{
    int eof = 97 | 98;
    printf("%d",eof);
    return 0;
}
```

outra operação bit a bit é o AND (e), essa operação também compara dois bytes e gera um 3º, porém o resultado só vai ser verdadeiro se os dois bits for verdadeiro

```
01100001 = a (97)
01100010 = b (98)
--------
01100000 = não lembro o que é isso (96)
```

para fazer a operação bit a bit AND na linguagem C/C++ usamos o operador “e” comercial (&)

```
#include <stdio.h>

int main(void)
{
    int eof = 97 & 98;
    printf("%d",eof);
    return 0;
}
```

também existe uma operação bit a bit chamador XOR (ou exclusivo), essa operação também compara dois bytes e gerando um 3º, porém o resultado só vai ser verdadeiro se os dois bits for diferente um do outro

```
01100001 = a (97)
01100010 = b (98)
--------
00000011 = só sei que é um código '-' (3)
```

para usar a operação bit a bit XOR usamos esse símbolo ^

```
#include <stdio.h>

int main(void)
{
    int eof = 97 ^ 98;
```

```
    printf("%d",eof);
    return 0;
}
```

diferente dos outros anteriores a operação bit a bit NOT (negação) inverte todos os bit de um único byte

```
01100001 = a (97)
10011110 = deu negativo devido o primeiro número ser 1 (-98)
```

para usar a operação bit a bit NOT na linguagem C/C++ usamos o operador til antes do valor (~)

```
#include <stdio.h>

int main(void)
{
    int eof = ~ 97;
    printf("%d",eof);
    return 0;
}
```

5.3 – Operadores de comparação e logico

Também existe os operadores de comparação e operadores lógicos, os operadores de comparação compara dois valores retornando verdadeiro ou falso (1/0), o primeiro operador que vamos ver é o operador de igualdade, ele compara dois valores se os dois valores for igual ele retorna 1 se for diferente ele retorna 0, não é possível comparar uma string inteira apenas um único carácter ou número, para esse operador usamos duas vezes o símbolo de igual (==)

```
#include <stdio.h>

int main(void)
{
    int eof = 315 == 100;
    printf("%d",eof);
    return 0;
}
```

existe o operador diferente que é oposto do operador igual, caso seja diferente ele retorna verdadeiro e caso seja igual ele retorna falso, para usar esse operador usamos o simbolo de exclamação e o símbolo de igual (!=)

```
#include <stdio.h>

int main(void)
{
    int eof = 315 != 100;
    printf("%d",eof);
    return 0;
}
```

outro operador é o operador “maior que”, esse operador retorna verdadeiro caso o valor seja maior que o outro para usar basta usar o símbolo de maior (>)

```
#include <stdio.h>

int main(void)
{
    int eof = 315 > 100;
    printf("%d",eof);
    return 0;
}
```

tem o operador “maior ou igual a”, esse operador retorna verdadeiro se o valor for maior ou igual ao outro, para usar esse operador usamos o símbolo de maior seguido do símbolo de igual (>=)

```
#include <stdio.h>

int main(void)
{
    int eof = 315 >= 100;
    printf("%d",eof);
    return 0;
}
```

também existe o operador “menor que”, esse operador retorna verdadeiro se o valor for menor que o outro, para usar esse operador usamos o símbolo de menor (<)

```
#include <stdio.h>

int main(void)
{
    int eof = 315 < 100;
    printf("%d",eof);
    return 0;
}
```

com o operador “menor ou igual a” ele retorna verdadeiro se o valor for menor ou igual ao outro, para usar esse operador usamos o símbolo de menor seguido do símbolo de igual (<=)

```
#include <stdio.h>

int main(void)
{
    int eof = 315 <=100;
    printf("%d",eof);
    return 0;
}
```

também e possível usar os operadores lógicos esses operadores compara os dois valores retornando verdadeiro ou falso, esses operadores tem os mesmos nomes dos operadores bit a bit mais não

confunda um com o outro (os operadores bit a bit compara cada bit de um byte retornando um novo byte e os lógicos compara dois valores retornando verdadeiro ou falso), o primeiro operador logico vai ser o OR esse operador compara dois valores e retorna verdadeiro se um dos dois for verdadeiro (só lembrando qualquer número maior que 0 é verdadeiro e qualquer numero menor ou igual a 0 é falso), para usar o operador logico OR usamos duas vezes o simbolo do pipe (||)

```
#include <stdio.h>
#include <stdbool.h>

int main(void)
{
    int eof = true || false;
    printf("%d",eof);
    return 0;
}
```

o operador lógico AND retorna verdadeiro apenas se os dois valores for verdadeiro, para usar ele usamos duas vezes o “e” comercial (&&)

```
#include <stdio.h>
#include <stdbool.h>

int main(void)
{
    int eof = true && false;
    printf("%d",eof);
    return 0;
}
```

o operador NOT inverte a lógica torando um valor verdadeiro ou falso, para usar ele colocamos o símbolo de exclamação antes

```
#include <stdio.h>
#include <stdbool.h>

int main(void)
{
    int eof = ! true;
    printf("%d",eof);
    return 0;
}
```

se vocês lembra da função strcmp da biblioteca string.h ele retorna o número 0 caso as duas strings seja igual, se ele retorna o valor 0 isso quer dizer que em termo lógico seria falso e se a gente comparar com uma estrutura de condição o compilador vai interpretar isso como falso e não fazer a condição, para isso existe o operador NOT para inverter a lógica

```
#include <stdio.h>
#include <string.h>

int main(void)
```

```
{
    int eof = ! strcmp("eof","eof");
    printf("%d",eof);
    return 0;
}
```

também podemos usar o parêntese para sobrecarregar os operadores

```
#include <stdio.h>
#include <stdbool.h>

int main(void)
{
    int eof = (! ((true && true ) || false)) ;
    printf("%d",eof);
    return 0;
}
```

5.4 – Estrutura condicional

As estruturas condicionais permite abstrair ou não determinadas partes do código sobre uma condição lógica, com essa condição você pode escolher o que será executado caso a condição seja verdadeira ou o que será executado caso a condição seja falso, existem pelo menos três estruturas condicionais na linguagem C/C++ são elas o if, switch e ternaria, as três condições pode ser usadas para o mesmo propósito embora o uso de uma em determinada situação pode deixa o código mais legível, a primeira condição que vamos ver é o if essa condição permite executar determinada parte do código se a condição dentro do argumento for verdadeiro, também colocamos um escopo para os códigos que será executado

```
#include <stdio.h>

int main(void)
{
    if(315 > 100)
    {
        printf("315 e maior que 100");
    }
    return 0;
}
```

outro exemplo com mais de um if

```
#include <stdio.h>

int main(void)
{
    if(315 > 100)
    {
        printf("315 e maior que 100");
    }
    if(315 < 100)
```

```
    {
        printf("315 e menor que 100");
    }
    return 0;
}
```

para evitar criar vários if para o mesmo propósito podemos usar o “else if”, a condição “else if” só será executada se nenhuma das condições anteriores for executada e quando uma for executada nenhuma outra condição será executada, para usar o “else if” precisa de uma condição if antes

```
#include <stdio.h>

int main(void)
{
    int eof = 315;
    if(eof > 100)
    {
        printf("maior que 100");
    }
    else if(eof < 100)
    {
        printf("315 e menor que 100");
    }
    return 0;
}
```

se precisamos executar determinada condição caso nenhuma das anteriores for executada usamos um else, o else sempre tem que ficar depois do if ou do último “else if” e ele não tem argumentos

```
#include <stdio.h>

int main(void)
{
    int eof = 100;
    if(eof > 100)
    {
        printf("maior que 100");
    }
    else if(eof < 100)
    {
        printf("menor que 100");
    }
    else
    {
        printf("seu numero e igual a 100");
    }
    return 0;
}
```

podemos colocar uma estrutura condicional dentro da outra

```
#include <stdio.h>

int main(void)
{
    int idade = 59;
    if(idade > 18)
    {
        printf("voce ja e de maior");
        if(idade < 50)
        {
            printf("voce ainda e novo ")
        }
        else
        {
            printf("voce e velho '-' ");
        }
    }
    else
    {
         printf("voce ainda nao pode dirigir")
    }
    return 0;
}
```

também dá para executar o if sem escopo, porém fica muito inlegível

```
#include <stdio.h>

int main(void)
{
    int eof = 315;
    if (eof > 100)
        printf("maior");
    else if(eof < 100)
        printf("menor");
    return 0;
}
```

a outra condição é o switch que tem a mesma utilidade do “if-else if”, para usar o switch temos que definir o valor nele e usar o case para checar se o número é o mesmo, no case colocamos o valor depois dois pontos seguido do que vai ser executado caso seja verdadeiro, também sempre temos que colocar um break a cada final de case para evitar ler o próximo case

```
#include <stdio.h>

int main(void)
{
    int eof = 100;
    switch(eof)
    {
        case 1:
```

```
            printf("eof = 1");
            break;
        case 100:
            printf("eof = 100");
            break;
        case 315:
            printf("eof = 315");
            break;
    }
    return 0;
}
```

podemos usar o default caso nenhuma das condições anteriores forem executadas ele será executado

```
#include <stdio.h>

int main(void)
{
    int eof = 500;
    switch(eof)
    {
        case 1:
            printf("eof = 1");
            break;
        case 100:
            printf("eof = 100");
            break;
        case 315:
            printf("eof = 315");
            break;
        default:
            printf("nenhuma opcao");
            break;
    }
    return 0;
}
```

outra estrutura de condição é a ternaria que permite um retorno sobre uma determinada condição, para usar a condição ternaria usamos um argumento que é a condição colocamos interrogação seguido do primeiro retorno caso seja verdadeiro, colocamos dois pontos e o segundo retorno caso seja falso, no exemplo abaixo ele vai retorna o maior valor

```
#include <stdio.h>

int main(void)
{
    int x = (315 > 100) ? 315 : 100;
    printf("%d",x);
    return 0;
}
```

podemos usar nas condições ternarias qualquer tipo de retorno

```
#include <stdio.h>
#include <stdbool.h>

int main(void)
{
    (true) ? printf("end of file") : printf("eof");
    return 0;
}
```

5.5 - Estrutura de repetição

As estruturas de repetição permite repetir uma quantidade de vezes uma determinada parte do código sobre uma determinada condição, existem pelo menos 4 estruturas de repetição são elas while, do-while, for e goto, a estrutura de repetição while repete enquanto que a condição dela for verdadeira assim que a condição for falsa o loop sessa se essa condição for sempre verdadeira o loop será infinito e as vezes travando o programa naquela parte, loop infinito pode ser perigoso as vezes como na hora de executar um programa ou alocar memória se estiver preso ao um loop infinito pode travar o computador devido ao gasto de memória, para usar o while temos que colocar a condição como argumento, a cada loop o programa vai checar de novo a condição para ver se vai ou não fazer mais um loop, depois da condição criamos um escopo onde colocamos o código que vai ser repetido

```
#include <stdio.h>
#include <stdbool.h>

int main(void)
{
    while(true)
    {
        printf("forum eof \n");
    }
    return 0;
}
```

para evitar que o programa fique preso a um loop infinito temos que criar um contador e incrementar o contador a cada loop ate determinada condição, no exemplo a baixo o contador começa com o número 1 e a cada loop checa para ver se a condição do contador e maior do que 10 caso não seja ele faz o loop de novo, a cada loop o contador vai incrementar mais um quando o contador chegar no número 11 a condição vai dar falso e sessar o loop

```
#include <stdio.h>

int main(void)
{
    int contador = 1;
    while(contador <= 10)
    {
        printf("%d \n",contador);
        contador++;
```

```
    }
    return 0;
}
```

para sessar um loop podemos usar o break, no exemplo abaixo ele ficaria preso ao um loop infinito se não tiver o break

```
#include <stdio.h>
#include <stdbool.h>

int main(void)
{
    while(true)
    {
        break;
    }
    return 0;
}
```

além do break existe o continue que faz mais um loop mesmo se a condição for falsa, no exemplo abaixo ele fica preso em um loop infinito

```
#include <stdio.h>

int main(void)
{
    int contador = 1;
    while(contador <= 10)
    {
        continue;
        contador++;
    }
    return 0;
}
```

o laço while só repete se a condição for verdadeira mesmo se for o primeiro loop, já o laço do-while ele repete pelo menos uma vez o laço e se a condição for verdadeira o loop continua ate que a condição seja falsa, para usar o do-while usamos o “do” criamos um escopo para o código que será repetido, depois criamos um while sem escopo com a condição

```
#include <stdio.h>

int main(void)
{
    int contador = 1;
    do
    {
        printf("%d \n",contador);
        contador++;
    }
    while(contador <= 10);
```

```
    return 0;
}
```

o laço “for” permite especificar o contador, a condição e o incremento, para usar o laço for criamos uma variável que será o contador depois passamos como argumento essa variável, a condição e o incremento da variável separado por ponto e vírgula e por fim o escopo onde fica o código que será repetido

```
#include <stdio.h>

int main(void)
{
    int contador = 1;
    for(contador ; contador <= 10 ; contador++)
    {
        printf("%d \n",contador);
    }
    return 0;
}
```

por fim o goto que não é exatamente um laço e sim um pulo para determinada parte do codigo embora seja limitado ao mesmo escopo da função, para usar o goto temos que criar um label que pode ser qualquer nome seguindo de dois pontos, depois do label criado basta usar goto seguido do nome do label para ele pular

```
#include <stdio.h>

int main(void)
{
    goto eof;
    eof:
    return 0;
}
```

o exemplo abaixo seria um loop infinito

```
#include <stdio.h>

int main(void)
{
    eof:
    goto eof;
    return 0;
}
```

podemos manipular o goto usando o if para evitar um loop infinito

```
#include <stdio.h>

int main(void)
```

```
{
    int contador = 1;
    eof:
    if(contador > 10)
    {
        goto fts;
    }
    printf("%d\n",contador);
    contador++;
    goto eof;
    fts:
    return 0;
}
```

6.0 – Função e recursividade

já usamos diversas funções e também criamos a função principal main, porém a criação de funções pode ser útil para agilizar o código uma função e um procedimento que o compilador pode fazer inúmeras vezes quando for chamado, uma função tem os argumentos e o retorno as funções deve ser criada antes da função principal main ou deve ter o protótipo dela antes, uma função também pode ter qualquer nome embora se houver outra função com o mesmo nome vai dá erro, para se cria uma função é a mesma maneira que criamos a função main colocamos primeiro o tipo de retorno o nome da função o argumento criamos um escopo e dentro dele no final do código usamos o return, na função principal ou em outras funções podemos chamar ela

```
#include <stdio.h>

int eof(void)
{
    printf("visite o forum eof\n");
    return 0;
}

int main(void)
{
    eof();
    return 0;
}
```

podemos criar a nossa função depois da função main para isso temos que colocar o protótipo dela, o protótipo ou assinatura é a parte sem o escopo da função

```
#include <stdio.h>

int eof(void);

int main(void)
{
    eof();
    return 0;
}
```

```
int eof(void)
{
    printf("visite o forum eof\n");
    return 0;
}
```

podemos criar variáveis como argumento para enviar para função e usar o retorn para retornar da função, no exemplo abaixo entramos com um valor ele retorna o quadrado do número

```
#include <stdio.h>

int eof_quad(int numero)
{
    return numero * numero;
}

int main(void)
{
    int fts = eof_quad(5);
    printf("%d",fts);
    return 0;
}
```

todas as variáveis dentro da função é liberada quando a função é destruída, então para que um valor fique em uma variável podemos usar as variáveis globais

```
#include <stdio.h>

int numero2 = 0;

void eof(int numero)
{
    numero2 += numero;
    printf("%d \n",numero2);
}

int main(void)
{
    eof(5);
    eof(6);
    return 0;
}
```

outra maneira é usar as variáveis estáticas

```
#include <stdio.h>

void eof(int numero)
{
    static int numero2 = 0;
```

```
    numero2 += numero;
    printf("%d \n",numero2);
}

int main(void)
{
    eof(5);
    eof(6);
    return 0;
}
```

para alterar variáveis locais de um outro escopo podemos criar um ponteiro na função e passar como argumento o endereço de memória isso é chamado passagem por referencia

```
#include <stdio.h>

void eof(int *numero)
{
    *numero = 315;
}

int main(void)
{
    int fts = 0;
    eof(&fts);
    printf("%d",fts);
    return 0;
}
```

Para passar uma array como argumento temos que passar por referencia e elas não precisa colocar o símbolo de endereço de memória

```
#include <stdio.h>

void eof(int *numero)
{
    numero[0] = 315;
}

int main(void)
{
    int fts[10];
    eof(fts);
    printf("%d",fts[0]);
    return 0;
}
```

quando temos que passar uma string como argumento podemos criar um ponteiro do tipo char e colocar const antes, dessa forma podemos tanto passar uma array como uma string diretamente para ser alocada

```
#include <stdio.h>

void eof(const char *texto)
{
    printf(texto);
}

int main(void)
{
    eof("visite o forum eof");
    return 0;
}
```

quando o sistema acha uma chamada de função ele para de executar aquele trecho do codigo e pula para a parte da função e quando a função é destruída ele volta para o trecho que paro e continua a execução e isso deixa o programa mais lento para evitar isso podemos usar funções inline, essas funções não faz o sistema pular para o endereço elas puxa a função para onde está o código de chamada dela, para usar função inline basta colocar inline antes do tipo de retorno

```
#include <stdio.h>

inline void eof(const char *texto)
{
    printf(texto);
}

int main(void)
{
    eof("visite o forum eof");
    return 0;
}
```

as funções recursivas permite chamar elas mesmas, porém deve se tomar cuidado com um loop infinito

```
#include <stdio.h>

void eof(void)
{
    printf("funcao recursiva \n");
    eof();
}

int main(void)
{
    eof();
}
```

para fazer a nossa função se executa depois que o programa for finalizado usamos a função atexit da biblioteca stdlib.h e passamos como argumento a nossa função

```
#include <stdio.h>
#include <stdlib.h>

void eof(void)
{
    printf("seu programa foi finalizado");
}

int main(void)
{
    atexit(eof);
    return 0;
}
```

6.1 – Diretivas de pré-processamento

As diretivas de pre processamento diz ao compilador como ele deve compilar o código, essas diretivas são as primeiras coisas que o compilador checa antes da compilação, você pode usar essas diretivas para definir se determinada parte do código vai ser compilada apenas para um sistema operacional ou impedir que determinada biblioteca seja declarada duas vezes, existem varias diretivas uma delas e para incluir biblioteca header que usamos inúmeras vezes a #include

```
#include <stdio.h>

int main(void)
{
    return 0;
}
```

outra diretiva que já usamos anteriormente é a #define para criar constante, porém essa diretiva pode ser usada para mais coisas do que simples constantes, podemos colocar códigos inteiros na diretiva #define embora muitas vezes pode deixar o código muito inlegível

```
#include <stdio.h>
#define FTS int main(void) { printf("visitem o forum eof"); return 0;}

FTS
```

para usar códigos de mais de uma linha para o #define ou ate mesmo para strings podemos usar contra barra no final

```
#include <stdio.h>
#define FTS \
int main(void) \
{ \
printf("visitem o forum eof"); \
return 0; \
}

FTS
```

também podemos passar argumentos para diretiva #define para isso basta colocar entre parênteses e colocar os nomes do argumento que será mudado

```
#include <stdio.h>
#define FTS(TEXTO) printf(TEXTO)

int main(void)
{
    FTS("forum eof");
    return 0;
}
```

outra diretiva é o #if que é bem parecida com a estrutura condicional if, a diferença que a diretiva #if é testado o que deve ser compilado e a estrutura condicional if o que deve ser executado, a diretiva #if não tem escopo só um #endif para finalizar e pode ser usada em qualquer parte do codigo

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    printf("visite o forum \n");
#if 1 == 1
    system("pause");
#endif
    return 0;
}
```

podemos colocar uma diretiva #if dentro da outra

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    printf("visite o forum \n");
#if 1 == 1
#if 1 > 0
    system("pause");
#endif
#endif
    return 0;
}
```

também podemos usar um #else depois do #if

```
#include <stdio.h>
#define OP 0
```

```
int main(void)
{
#if OP == 1
     printf("visite o forum \n");
#else
    printf("by: hfts315");
#endif
    return 0;
}
```

também existe a diretiva #ifdef que compila determinada parte do código se determinada diretiva foi declarada

```
#include <stdio.h>
#define FTS 315

int main(void)
{
#ifdef FTS
    printf("sim");
#else
    printf("nao");
#endif
    return 0;
}
```

existe o oposto da diretiva #ifdef que é o #ifndef

```
#include <stdio.h>

int main(void)
{
#ifdnef FTS
    printf("sim");
#else
    printf("nao");
#endif
    return 0;
}
```

para remover uma diretiva #define podemos usar a diretiva #undef

```
#include <stdio.h>
#define FTS 315
#undef FTS

int main(void)
{
#ifdef FTS
    printf("sim");
#else
```

```
    printf("nao");
#endif
    return 0;
}
```

além das diretivas existem umas constantes que pode ser usadas junto com as diretivas como por exemplo essas que da para saber qual sistema operacional

```
#include <stdio.h>

int main(void)
{
#ifdef WIN32
    printf("windows x32");
#endif
#ifdef WIN64
    printf("windows x64");
#endif
#ifdef _linux
    printf("linux");
#endif
    return 0;
}
```

existe a diretiva #line permite o compilador definir o número da linha, também podemos usar a constante __LINE__ para saber qual é a linha atual

```
#include <stdio.h>

int main(void)
{
    printf("o numero de linha: %d \n",__LINE__);
#line 0
    printf("o numero de linha: %d \n",__LINE__);
#line 315
    printf("o numero de linha: %d \n",__LINE__);
    return 0;
}
```

quando queremos que a source não compile podemos usar a diretiva #error seguida de uma mensagem

```
#include <stdio.h>
#error nao vai compilar, bem feito XD

int main(void)
{
    return 0;
}
```

6.2 – Criando Biblioteca Header

já usamos diversas bibliotecas em nossas sources agora vamos aprender como criar elas para poder reaproveitar o código, para criar uma biblioteca header é muito facil basta criar um arquivo com extensão .h (no meu caso criei um chamado endoffile.h), esse arquivo deve ficar ou na pasta include junto com as outras bibliotecas do compilador (no meu caso ela fica “C:\Dev-Cpp\include”) ou na pasta junto com a source, como foi dito em “bibliotecas da linguagem C/C++” se o arquivo biblioteca estiver junto a source deve incluir com aspas duplas e se tiver na pasta include deve incluir com abre e fecha menor e maior

```
#include "endoffile.h" //biblioteca na pasta da source
#include <endoffile.h> //biblioteca na pasta do compilador

int main(void)
{
    return 0;
}
```

as bibliotecas header só necessita criar as funções para funcionar, porém é bom checar se o arquivo já não foi incluso para isso usamos duas diretivas que são a #ifndef e #define, é comum usar como argumento para #ifndef e #define o nome da biblioteca entre duas vezes underline

```
#ifndef __ENDOFFILE_H__
#define __ENDOFFILE_H__
#endif
```

sempre que criar uma função coloque o protótipo dela também

```
#ifndef __ENDOFFILE_H__
#define __ENDOFFILE_H__

int eof_quad(int numero);
int eof_cubo(int numero);

int eof_quad(int numero)
{
    return numero * numero;
}

int eof_cubo(int numero)
{
    return numero * numero * numero;
}

#endif
```

podemos criar biblioteca para carregar outras bibliotecas é uma forma de facilitar e não precisar ficar declarando várias bibliotecas

```
#ifndef __ENDOFFILE_H__
#define __ENDOFFILE_H__

```

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>

#endif
```

quando estamos compilando pelo gcc e temos que incluir um diretorio que tem as bibliotecas usamos a sintaxe -I seguido do endereço do diretorio

```
fts315@skynet:~/Desktop$ gcc fts.c -o hack.o -I "/home/fts315/Desktop/bibliotecas"
```

6.3 – Criando Biblioteca Estática e LIB

as biblioteca estática são compiladas junto com arquivo objeto gerando o executavel final, para criar o arquivo objeto e não o executavel basta adicionar a sintaxe de comando -c no compilador gcc ou nas opção de compilação da IDE

```
fts315@skynet:~/Desktop$ gcc fts.c -o hack.o -c
```

poderíamos usar arquivos objetos como biblioteca, um exemplo basico seria essa com apenas uma função

```
#ifndef __HACK_OBJ__
#define __HACK_OBJ__

int fts(int x)
{
    return x * x;
}

#endif
```

depois de gerar o objeto poderíamos usar esse arquivo objeto junto a qualquer outra source para poder usar as função nela, porem na hora de compilar basta compilar junto

```
fts315@skynet:~/Desktop$ gcc lol.c hack.o -o eof
```

tambem poderia colocar o prototipo dela dentro das sources, para fazer isso teríamos que colocar o identificador extern antes

```
#include <stdio.h>

extern int fts(int x);

int main(void)
{
    printf("%d",fts(5));
    return 0;
}
```

os libs são parecido com arquivo objeto só é incluído na hora da compilação eles são uma espécime de compactação de arquivos objetos, para se criar um lib temos que criar um arquivo objeto antes no caso vou usar o hack.o que eu já tinha criado, depois usamos o programa ar que provavelmente vem junto com seu compilador, nesse programa usamos a sintaxe “ar cru” seguido do nome do arquivo lib a ser criado (deve ter extensao .a) e depois os arquivos objetos

```
fts315@skynet:~/Desktop$ ar cru fts.a hack.o
```

na compilação do gcc deve incluir a sintaxe -L seguido do diretorio que esta o arquivo lib e a sintaxe -l seguido do arquivo lib

```
fts315@skynet:~/Desktop$ gcc lol.c -o hack -L "/home/fts315/Desktop" -l fts.a
```

para visualizar os arquivos objetos dentro de um lib podemos usar a sintaxe “ar t” seguido do arquivo objeto

```
fts315@skynet:~/Desktop$ ar t fts.a
hack.o
fts315@skynet:~/Desktop$
```

6.4 – Criando Biblioteca Dinâmica (DLL e SO)

As bibliotecas dinamicas são carregadas em tempo de execução (run-time) diferente das estaticas que são compiladas junto com executavel gerando o executavel final, a vantagem das bibliotecas dinamicas que o executavel fica menor e pode gastar menos memoria a desvantagem que se faltar a biblioteca o programa pode não funcionar, as bibliotecas dinamicas usada no sistema windows são as dll boa parte das API do sistema operacional esta dentro de alguma dll ou pelo menos os protótipo dela, e a maioria das dll esta no diretorio windows\system32 esse diretorio tambem é considerado o inferno de dll devido a quantidade de dll quem tem nele, para criar uma dll no sistema windows usamos as APIs do sistema operacional e também usamos a sintaxe --shared do gcc, em uma dll não usamos a função principal main e sim DllMain que tem um retorno do tipo BOOL APIENTRY, os argumentos dela são 3 variaveis do tipo HINSTANCE, DWORD e LPVOID, e a função tem um retorno TRUE, também é necessário incluir a biblioteca windows.h para pode usar as APIs citadas

```
#include <windows.h>

BOOL APIENTRY DllMain(HINSTANCE hin, DWORD dw, LPVOID lpv)
{
    return TRUE;
}
```

como dito antes para compilar é necessário usar a sintaxe --shared no gcc

```
C:\Users\fts315\Desktop> gcc fts.c -o fts.dll --shared
```

aquela dll não tem nenhuma função então tambem não tem nehuma utilidade para nois programadores, para criar uma função que sera exportada, ou seja será acessada pelos programas que carregar aquela dll, temos que colocar um prototipo da função e no prototipo temos que colocar

__declspec(dllexport) no começo do prototipo isso permite a função ser exportada

```
#include <windows.h>

__declspec(dllexport) int fts_quad(int valor);

BOOL APIENTRY DllMain(HINSTANCE hin, DWORD dw, LPVOID lpv)
{
    return TRUE;
}

int fts_quad(int valor)
{
    return valor * valor;
}
```

no escopo da DllMain podemos usar funçoes que sera executada assim que a dll for carregada sem precisar ser chamada as funçoes (seria um autorun, cuidado com as funçoes chamadas ali porque se não fizer um checagem adequada pode ficar em loop infinito)

```
#include <windows.h>
#include <stdio.h>

__declspec(dllexport) int fts_quad(int valor);

BOOL APIENTRY DllMain(HINSTANCE hin, DWORD dw, LPVOID lpv)
{
    FILE *arq = fopen("fts.txt","w");
    fprintf(arq,"lol sou otaku, curto rock, faco programa e sou viciado em cafeina\n");
    return TRUE;
}

int fts_quad(int valor)
{
    return valor * valor;
}
```

como dito antes podemos fazer uma checagem para isso usamos as constantes DLL_PROCESS_ATTACH essa constante retorna verdadeiro quando a dll for carregada, e a constante DLL_PROCESS_DETACH quando ela for liberada

```
#include <windows.h>
#include <stdio.h>

__declspec(dllexport) int fts_quad(int valor);

BOOL APIENTRY DllMain(HINSTANCE hin, DWORD dw, LPVOID lpv)
{
    if(DLL_PROCESS_ATTACH)
    {
```

```
    }
    else if(DLL_PROCESS_DETACH)
    {
    }
    return TRUE;
}

int fts_quad(int valor)
{
    return valor * valor;
}
```

podemos executar funções dentro de uma dll com um programa do windows chamado rundll32 a sintaxe dele é a dll uma vírgula e o número da dll (ordem) ou nome dela

```
C:\Users\fts315\Desktop> rundll32 fts.dll,fts_quad
```

para carregar uma dll na linguagem C/C++ temos que usar a API LoadLibrary e passar como argumento a dll, e para libera-la ela podemos usar o FreeLibrary seguido da variavel do tipo HINSTANCE que tinha atribuído a dll

```
#include <stdio.h>
#include <windows.h>

int main(void)
{
    HINSTANCE dll;
    dll = LoadLibrary("fts.dll");
    FreeLibrary(dll);
    return 0;
}
```

só carregar a dll não é suficiente para usar as funções dentro dela, para fazer isso temos que criar um tipo com o prototipo igual ao da função e depois carregar ela pelo endereço, para criar o prototipo usamos typedef colocamos o tipo de retorno da função abrimos e fechamos parênteses e la dentro colocamos um ponteiro com um nome qualquer, depois abrimos e fechamos parênteses de novo e colocamos os tipos dos argumentos separados por virgula caso tenha mais de um, e depois instanciamos esse tipo

```
#include <stdio.h>
#include <windows.h>

typedef int (*fun)(int);

int main(void)
{
    fun eof;
    HINSTANCE dll;
    dll = LoadLibrary("fts.dll");
    FreeLibrary(dll);
    return 0;
```

```
}
```

agora só falta pegar o endereço e atribuir ele para o tipo, para pegar o tipo usamos a API GetProcAddress passamos como argumento para ela a variavel que esta atribuida a dll, e uma string que seria o nome da função, depois atribuimos a função GetProcAddress para a instância do tipo que tínhamos criado (usamos typescast do tipo para conveter sem erros)

```
#include <stdio.h>
#include <windows.h>

typedef int (*fun)(int);

int main(void)
{
    fun eof;
    HINSTANCE dll;
    dll = LoadLibrary("fts.dll");
    eof = (fun) GetProcAddress(dll,"fts_quad");
    FreeLibrary(dll);
    return 0;
}
```

para terminar só a gente manipular a instancia daquele tipo que criamos como se fosse a própria função da dll

```
#include <stdio.h>
#include <windows.h>

typedef int (*fun)(int);

int main(void)
{
    int valor;
    fun eof;
    HINSTANCE dll;
    dll = LoadLibrary("fts.dll");
    eof = (fun) GetProcAddress(dll,"fts_quad");
    valor = eof(5);
    printf("%d",valor);
    FreeLibrary(dll);
    return 0;
}
```

tambem é possível criar biblioteca dinâmica no linux que são os arquivos SO, a sintaxe para criar esse arquivo é mais facil do que as dll windows, para criar esses arquivos SO podemos criar as funçoes normais não precisar colocar nenhuma função DllMain igual as dll do windows

```
int fts_quad(int valor)
{
    return valor * valor;
}
```

na hora de compilar tambem temos que colocar a sintaxe --shared no gcc

```
fts315@skynet:~/Desktop$ gcc biblioteca.c -o fts.so --shared
```

para carregar o arquivo usamos a API dlopen da biblioteca dlfcn.h, passamos como argumento para essa API o arquivo SO e como ele sera aberto (podemos colocar a constante RTLD_LAZY), e atribuimos tudo a um ponteiro do tipo void

```
#include <stdio.h>
#include <dlfcn.h>

int main(void)
{
    void *dl;
    dl = dlopen("/home/fts315/Desktop/fts.so",RTLD_LAZY);
    return 0;
}
```

Podemos liberar o arquivo com a API dlclose os argumentos dela é a variavel do tipo void

```
#include <stdio.h>
#include <dlfcn.h>

int main(void)
{
    void *dl;
    dl = dlopen("/home/fts315/Desktop/fts.so",RTLD_LAZY);
    dlclose(dl);
    return 0;
}
```

agora criamos um protótipo para isso colocamos o tipo de retorno abre e fecha parênteses um ponteiro com qualquer nome la dentro, abre e fecha parênteses os argumentos

```
#include <stdio.h>
#include <dlfcn.h>

int main(void)
{
    void *dl;
    int (*fts)(int numero);
    dl = dlopen("/home/fts315/Desktop/fts.so",RTLD_LAZY);
    dlclose(dl);
    return 0;
}
```

agora usamos a API dlsym passamos como argumento para ela a variavel que esta armazenada o arquiv SO, e o nome da função

```
#include <stdio.h>
#include <dlfcn.h>

int main(void)
{
    void *dl;
    int (*fts)(int numero);
    dl = dlopen("/home/fts315/Desktop/fts.so",RTLD_LAZY);
    fts = dlsym(dl,"fts_quad");
    dlclose(dl);
    return 0;
}
```

depois só usar aquele tipo como se fosse a função

```
#include <stdio.h>
#include <dlfcn.h>

int main(void)
{
    void *dl;
    int (*fts)(int numero);
    int eof;
    dl = dlopen("/home/fts315/Desktop/fts.so",RTLD_LAZY);
    fts = dlsym(dl,"fts_quad");
    eof = (*fts)(10);
    printf("%d",eof);
    dlclose(dl);
    return 0;
}
```

para compilar e necessário linkar a lib dl junto com o gcc

```
fts315@skynet:~/Desktop$ gcc fts.c -o hack -l dl
```

6.5 – Criando Makefile

Os make files ou arquivos de criação são arquivos para facilitar a compilação de determinada source evitando do programador ou usuário final ter que digitar todos os paramentos para poder compilar, esses arquivos são usado constantemente no linux para instalar pacotes, os make files permite usar variaveis e ao mesmo tempo faz checagem dos paramentos, os arquivos make file deve estar escrito por padrão “Makefile” caso não esteja temos que especificar uma sintaxe para incluir o arquivo, o programa que é usado para compilar pelos arquivos make file é o make que costuma vim junto com os compiladores, os arquivos make usa a seguinte estrutura que é o nome do paramento seguido de dois pontos e os arquivos de checagem depois uma quebra de linha e uma tabulação e o comando usado pelo terminal

```
arquivo: fts.c
	gcc fts.c -o hack
```

para compilar basta achar o arquivo pelo prompt e digitar make

```
fts315@skynet:~/Desktop$ make
gcc fts.c -o hack
fts315@skynet:~/Desktop$
```

caso o arquivo não existi-se ele não ia compilar

```
fts315@skynet:~/Desktop$ make
make: *** Sem regra para processar o alvo `fts.c', necessário por `arquivo'.  Pare.
fts315@skynet:~/Desktop$
```

caso o arquivo make estiver com outro nome temos que usar a sintaxe -f seguido do nome do arquivo

```
fts315@skynet:~/Desktop$ make -f lol
gcc fts.c -o hack
fts315@skynet:~/Desktop$
```

podemos comentar os make files com sharps (#)

```
#by hfts315

arquivo: fts.c
	gcc fts.c -o hack
```

é possível criar novos paramentos para isso na checagem colocamos o nome do novo paramento, o novo paramento sera executado antes do anterior e o anterior sera executado apenas se der certo o outro

```
arquivo: fts.c objeto
	gcc fts.c lol.o -o hack

objeto: lol.o
	gcc objeto.c -o lol.o -c
```

também é possível criar variáveis para isso basta escrever o nome da variável depois usar igual seguido do valor, para usar as variáveis usamos cifrão seguido de abre e fecha parênteses dentro dele o nome da variável

```
compilador=gcc
source=fts.c
saida=hack

arquivo: $(source)
	$(compilador) $(source) -o $(saida)
```

podemos entrar com o valor para as variavel com a sintaxe -e seguido das variaveis e valores

```
fts315@skynet:~/Desktop$ make -e compilador=g++ saida=hfts
```

```
g++ fts.c -o hfts
fts315@skynet:~/Desktop$
```

7.0 – Estruturas

As estruturas são meios de armazenamento de dados do mesmo tipo ou de tipos diferentes, uma das estruturas mais usadas na linguagem C é o struct, para criar uma estrutura struct temos que colocar a palavra struct seguida do nome dela que pode ser qualquer um, depois criamos um escopo para colocar os tipo de dados e variáveis, depois do escopo precisamos finalizar com um ponto e vírgula

```
#include <stdio.h>

struct eof
{
};

int main(void)
{
    return 0;
}
```

agora só colocarmos as varáveis dentro da estrutura

```
#include <stdio.h>

struct eof
{
    int dia;
    int mes;
    int ano;
};

int main(void)
{
    return 0;
}
```

depois de criar a estrutura temos que instanciar (declarar) ela na função para poder usar, cada instância criada não afeta a outra, para instanciar uma estrutura na função colocamos a palavra struct seguido do nome da estrutura que vamos instanciar e o nome da instância que pode ser qualquer um

```
#include <stdio.h>

struct eof
{
    int dia;
    int mes;
    int ano;
};
```

```
int main(void)
{
    struct eof fts;
    return 0;
}
```

com a estrutura criada e instanciada basta acessar as variáveis dentro dela, para isso usamos o nome da instância seguido de um ponto e nome da variável depois podemos atribuir como se fosse uma variável mesmo

```
#include <stdio.h>

struct eof
{
    int dia;
    int mes;
    int ano;
};

int main(void)
{
    struct eof fts;
    fts.dia = 12;
    fts.mes = 10;
    fts.ano = 2013;
    printf("%d/%d/%d",fts.dia,fts.mes,fts.ano);
    return 0;
}
```

como dito antes cada instância não afeta a outra

```
#include <stdio.h>

struct eof
{
    int numero;
};

int main(void)
{
    struct eof fts;
    struct eof lol;
    fts.numero = 315;
    lol.numero = 100;
    printf("%d \n%d",fts.numero,lol.numero);
    return 0;
}
```

podemos declarar usando a mesma linha separando por vírgula quando é a mesma estruturas

```
#include <stdio.h>
```

```
struct eof
{
    int numero;
};

int main(void)
{
    struct eof fts,lol;
    fts.numero = 315;
    lol.numero = 100;
    printf("%d \n%d",fts.numero,lol.numero);
    return 0;
}
```

podemos declarar estruturas como ponteiros adicionando asterisco antes do nome da instância, na hora de acessar os dados não usamos o ponto e sim o hífen seguido do maior (->)

```
#include <stdio.h>

struct eof
{
    int numero;
};

int main(void)
{
    struct eof *fts;
    fts->numero = 315;
    printf("%d",fts->numero);
    return 0;
}
```

também e possível passar como argumento o endereço de memória de uma estrutura para uma função para poder ser manipulada

```
#include <stdio.h>

struct eof
{
    int numero;
};

void lol(struct eof *ponteiro)
{
    ponteiro->numero = 315;
}

int main(void)
{
    struct eof fts;
```

```
    lol(&fts);
    printf("%d",fts.numero);
    return 0;
}
```

poderíamos criar uma declaração global usando typedef antes do struct, e antes do ponto e vírgula os nomes predefinidos da struct separado por vírgula, dessa forma na hora de declarar podemos declarar sem a palavra struct

```
#include <stdio.h>

typedef struct eof
{
    int numero;
}lol;

int main(void)
{
    lol fts;
    fts.numero = 315;
    printf("%d",fts.numero);
    return 0;
}
```

outra forma seria criar o typedef separadamente

```
#include <stdio.h>

typedef struct eof lol;

struct eof
{
    int numero;
};

int main(void)
{
    lol fts;
    fts.numero = 315;
    printf("%d",fts.numero);
    return 0;
}
```

também podemos alocar dinamicamente pelo malloc

```
#include <stdio.h>
#include <stdlib.h>

struct eof
{
    int numero;
```

```
};

int main(void)
{
    struct eof *fts = malloc(sizeof(struct eof));
    fts->numero = 315;
    printf("%d",fts->numero);
    free(fts);
    return 0;
}
```

existem outras estruturas além da struct uma delas é o enum que serve para enumerar constantes com números consecutivos (0,1,2,3,4 etc), para criar a estrutura enum basta usar o nome dela criar um escopo e colocar o nome das constantes separado por vírgula, e para terminar um ponto e virgula depois do escopo, o contador começa no numero 0 e a próxima constante sera o numero anterior incrementado mais 1

```
#include <stdio.h>

enum
{
    FTS, MMXM, SIR_RAFIC, SUSPEITO_VIRTUAL
};

int main(void)
{
    printf("%d %d %d %d",FTS,MMXM,SIR_RAFIC,SUSPEITO_VIRTUAL);
    return 0;
}
```

para especificar um número de uma constante basta atribuir o número a ela, só lembrando que a próxima será um incremento da anterior

```
#include <stdio.h>

enum
{
    FTS = 315, MMXM, SIR_RAFIC, SUSPEITO_VIRTUAL
};

int main(void)
{
    printf("%d %d %d %d",FTS,MMXM,SIR_RAFIC,SUSPEITO_VIRTUAL);
    return 0;
}
```

outra estrutura também da linguagem C é o union que permite uma alocação tenha mais de um tipo de dado embora essa alocação só possa guardar um valor por vez

```
#include <stdio.h>
```

```
union eof
{
    int fts;
    float lol;
};

int main(void)
{
    union eof hack;
    hack.fts = 315;
    hack.lol = 3.15;
    printf("%d \n%d",&hack.fts,&hack.lol);
}
```

uma estrutura da linguagem C++ é o namespace que permite separar o código evitando variáveis funções e estruturas com o mesmo nome o namespace mais usada na linguagem C++ é o std onde existe as funções cout, cin e endl, para criar um namespace basta colocar a palavra namespace seguida do nome dele que pode ser qualquer um, depois um escopo e la dentro as variáveis e funções

```
#include <iostream>

namespace eof
{
    int fts;
};

int main(void)
{
    return 0;
}
```

para acessar os dados dentro do namespace usamos o nome do namespace seguida de duas vezes dois pontos e a variável ou função

```
#include <iostream>

namespace eof
{
    int fts;
};

int main(void)
{
    eof::fts = 315;
    std::cout << eof::fts;
    return 0;
}
```

como dito antes o namespace permite colocar variáveis com o mesmo nome evitando incompatibilidade

```
#include <iostream>

namespace eof
{
    int fts;
};

namespace hack
{
    int fts;
};

int main(void)
{
    eof::fts = 315;
    hack::fts = 100;
    std::cout << eof::fts << std::endl;
    std::cout << hack::fts;
    return 0;
}
```

para evitar não ficar colocando o namespace podemos indicar que determinada função ou variável pertence aquele namespace, para isso usamos a palavra using seguido dele e a variavel ou função (lembrando que para fazer isso tem que criar primeiro o namespace)

```
#include <iostream>

using std::cout;
using std::endl;

namespace eof
{
    int fts;
};

using eof::fts;

int main(void)
{
    fts = 315;
    cout << fts;
    return 0;
}
```

para fazer isso com todas as funções e variáveis do namespace podemos usar “using namespace” e o nome do namespace

```
#include <iostream>

using namespace std;
```

```
namespace eof
{
    int fts;
};

using namespace eof;

int main(void)
{
    fts = 315;
    cout << fts;
    return 0;
}
```

também podemos colocar um namespace dentro de outro

```
#include <iostream>

using namespace std;

namespace eof
{
    namespace fts
    {
        int hack;
    };
};

int main(void)
{
    eof::fts::hack = 315;
    cout << eof::fts::hack;
    return 0;
}
```

no using ficaria assim

```
#include <iostream>

using namespace std;

namespace eof
{
    namespace fts
    {
        int hack;
    };
};

using namespace eof::fts;
```

```
int main(void)
{
    hack = 315;
    cout << hack;
    return 0;
}
```

outra estrutura da linguagem C++ são as classes que vamos ver mais para frente em orientação a objeto

7.1 – Conversão de tipo e dados

Existem vários tipos de conversão uma delas é o typecast, esse tipo de conversao permite converter um tipo de dado para outro tipo de dado um exemplo é int para char, em teoria o typecast pode ser uma simples atribuição

```
#include <stdio.h>

int main(void)
{
    int eof = 97;
    char fts = eof;
    printf("%c",fts);
    return 0;
}
```

embora o código anterior vai funciona seria mais correto especificar a conversão usando abre e fecha parênteses e o tipo de dado a ser convertido antes da variável

```
#include <stdio.h>

int main(void)
{
    int eof = 97;
    char fts = (char) eof;
    printf("%c",fts);
    return 0;
}
```

outra conversão interessante é transformar uma string em numero int com a função atoi da biblioteca stdlib.h, para usar essa função basta passar como argumento a string e pega o retorno do numero convertido

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    char eof[10] = "315";
    int fts = atoi(eof);
```

```
    printf("%d",fts);
    return 0;
}
```

para converte para um número float usamos a função atof da biblioteca stdlib.h

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    char eof[10] = "3.15";
    float fts = atof(eof);
    printf("%f",fts);
    return 0;
}
```

existe a função itoa da biblioteca stdlib.h que converte uma variável int para uma string sobre uma base especifica, para usar essa função passamos como argumento o número a ser convertido, a array do tipo char e por fim a base (10 = decimal, 16 = hexadecimal, 8 = octal, 2 = binario)

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    char eof[10];
    int fts = 315;
    itoa(fts,eof,2);
    printf("%s",eof);
    return 0;
}
```

a função atoi não permite alguns formatos de números como por exemplo hexadecimal para isso usamos a função strtol da biblioteca stdlib.h, para usar ele passamos como argumento a string, um endereço de memória de um ponteiro do tipo char que podemos colocar NULL (NULL = 0) e a base do número da string, o retorno da função é o número convertido

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    char eof[10] = "0x61";
    int fts = strtol(eof,NULL,16);
    printf("%d",fts);
    return 0;
}
```

onde colocamos NULL poderíamos especificar um ponteiro, esse ponteiro seria útil caso tivesse outros números na string

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    char eof[15] = "0x61 1100001";
    char *ponteiro;
    int fts = strtol(eof,&ponteiro,16);
    printf("0x61 = %d \n",fts);
    fts = strtol(ponteiro,&ponteiro,2);
    printf("1100001 = %d",fts);
    return 0;
}
```

7.2 – Manipulando o tempo

Bom galera é muito útil aprender a manipular o tempo isso permite fazer programas que só execute determinados procedimentos em um certo tempo ou em um intervalo de tempo ou simplesmente mostre a hora atual, a hora no computador é registrado a cada segundo deis de 1970 isso permite formatar o horário conforme regras do GMT e não se limitar apenas a um fuso horário ou tempo, a primeira função que vamos ver vai ser o sleep da biblioteca stdlib.h, essa função permite pausar o programa por um determinado tempo, para usar essa função basta passar como argumento o tempo em segundos ou milésimos que pode variar entre o sistema operacional (o sistema windows o valor é em milésimos já no linux o valor é em segundos, então 1 segundo no windows pelo sleep é equivalente a 1000 já no linux 1 segundo é 1)

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    printf("visitem o ");
    sleep(1000);
    printf("forum eof");
    return 0;
}
```

no linux o programa anterior ficaria assim

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    printf("visitem o ");
    sleep(1);
    printf("forum eof");
    return 0;
}
```

a função time da biblioteca time.h retorna o segundos deis de 1970, para usar essa função basta passar como argumento o valor NULL e atribuir o retorno que é o tempo, embora possamos usar variáveis do tipo int é recomendado usar a variável do tipo time_t que se encontra na biblioteca time.h também

```
#include <stdio.h>
#include <time.h>

int main(void)
{
    time_t eof = time(NULL);
    printf("%d",eof);
    return 0;
}
```

na função time também é possível passar como argumento o endereço de memória da variável em vez de atribuir a mesma

```
#include <stdio.h>
#include <time.h>

int main(void)
{
    time_t eof;
    time(&eof);
    printf("%d",eof);
    return 0;
}
```

uma forma de formatar o tempo para string no formato (o nome do dia, o nome do mês, a data, a hora e o ano) podemos usar a função ctime da biblioteca time.h, passamos como argumento para ela o endereço de memória da variável time_t com o tempo

```
#include <stdio.h>
#include <time.h>

int main(void)
{
    time_t eof = time(NULL);
    printf("%s",ctime(&eof));
    return 0;
}
```

outra forma seria usar a estrutura tm da biblioteca time.h, para isso criamos um ponteiro da estrutura tm e atribuímos a função localtime da biblioteca time.h, na função localtime passamos como argumento o endereço de memória da variável time_t com o tempo em segundos, depois basta acessar as variáveis da estrutura tm que são tm_sec para os segundos, tm_min para os minutos, tm_hour para as horas, tm_mday para o dia conforme o mês, tm_yday para o dia conforme o ano, tm_mon para o mês, tm_wday para o dia da semana, tm_year para o ano

```
#include <stdio.h>
```

```
#include <time.h>

int main(void)
{
    struct tm *fts;
    time_t eof = time(NULL);
    fts = localtime(&eof);
    printf("segundos %d \n",fts->tm_sec);
    printf("minutos %d \n",fts->tm_min);
    printf("hora %d \n",fts->tm_hour);
    printf("dia(semana) %d \n",fts->tm_wday);
    printf("dia(mes) %d \n",fts->tm_mday);
    printf("dia(ano) %d \n",fts->tm_yday);
    printf("mes %d \n",fts->tm_mon);
    printf("ano %d \n",fts->tm_year);
    return 0;
}
```

a função localtime retorna com base no horário do seu computador para retorna o horario atual GMT+0 para manipular outros fuso horários usamos a função gmtime da biblioteca time.h no lugar de localtime e somamos ou subtraímos a hora conforme o GMT desejado, no exemplo abaixo podemos ver o fuso horário greenwich (GMT0), brasil (GTM-2,GTM-3 e GTM-4) e japao (GMT+9, adoro as japonesas *-*)

```
#include <stdio.h>
#include <time.h>

int main(void)
{
    struct tm *fts;
    time_t eof = time(NULL);
    fts = gmtime(&eof);
    printf("GMT0 (greenwich): %d:%d:%d \n",fts->tm_hour,fts->tm_min,fts->tm_sec);
    printf("GMT-2 (acre): %d:%d:%d \n",fts->tm_hour-2,fts->tm_min,fts->tm_sec);
    printf("GMT-3 (rj): %d:%d:%d \n",fts->tm_hour-3,fts->tm_min,fts->tm_sec);
    printf("GMT-4 (f. noronha): %d:%d:%d \n",fts->tm_hour-2,fts->tm_min,fts->tm_sec);
    printf("GMT+9 (japao): %d:%d:%d \n",fts->tm_hour+9,fts->tm_min,fts->tm_sec);
    return 0;
}
```

podemos usar a função difftime para retorna a diferença entre dois tempos, pegamos o tempo inicial e o final com a função time, depois usamos a função difftime passamos como argumento os dois e atribuindo o mesmo a uma variável do tipo time_t

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main(void)
{
    time_t eof, fts, hack;
```

```
    eof = time(NULL);
    sleep(3000);
    fts = time(NULL);
    hack = difftime(fts,eof);
    printf("%d",hack);
    return 0;
}
```

uma outra função interessante é o rand e srand da biblioteca stdlib.h que retorna números aleatórios, embora não seja uma maneira de manipular o tempo para usar essas funções precisa usar a função time, para retorna numero aleatórios usamos a função srand e passamos como argumento a função time, depois usamos a função rand com modulo e um número que seria o valor máximo a ser retornado

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main(void)
{
    srand(time(NULL));
    printf("numero randomico 1: %d \n",rand() % 100);
    printf("numero randomico 2: %d \n",rand() % 1000);
    printf("numero randomico 3: %d \n",rand() % 10);
    return 0;
}
```

7.3 – Manipulando arquivos

Acho que a manipulação de arquivos é uma das coisas mais importante na programação devido o armazenamento e leitura de informação, quando usamos variaveis aquilo fica temporariamente na memória enquanto o programa estiver rodando depois do programa ser finalizado a memoria e liberada e o seu conteúdo destruído e isso pode ser um problema se precisarmos recuperar esses dados da variável, para evitar esse problema podemos criar arquivos no HD ou outro dispositivo de armazenamento e acessar eles quando precisar, para criar um arquivo na linguagem C/C++ podemos usar a função fopen da biblioteca stdio.h, para usar essa função temos que passar como argumento o nome do arquivo ou endereço completo dele e tipo de acesso, o tipo de acesso é uma string com os seguintes caracteres, “w” para escrita, “r” para leitura, “a” para escrita e concatenação, “b” para binario e deve ser usada depois da escrita ou leitura “rb”, também podemos colocar um + no final indicando escrita e leitura ou leitura e escrita “w+”, o tipo de acesso de leitura apenas abre o arquivo já o tipo escrita ele abre o arquivo caso o arquivo não exista ele cria o arquivo, então no caso para criar usamos tipo de acesso de escrita

```
#include <stdio.h>

int main(void)
{
    fopen("eof.txt","w");
    return 0;
}
```

como dito antes podemos usar o endereço completo para criar ou acessar o arquivo (só lembrando que a linguagem C/C++ usar contra barra como código de escape então temos que usar contra barra dupla)

```
#include <stdio.h>

int main(void)
{
    fopen("C:\\Users\\fts315\\Desktop\\eof.txt","w");
    return 0;
}
```

para acessarmos um arquivo para escrita ou leitura não basta usar apenas a função fopen, temos que atribuir ela a ponteiro do tipo FILE da biblioteca stdio.h, e depois usar a função para escrever ou ler o arquivo, depois do uso da escrita ou leitura temos que fechar o arquivo para atualizar para isso usamos a função fclose da biblioteca stdio.h seguido do ponteiro FILE

```
#include <stdio.h>

int main(void)
{
    FILE *fts;
    fts = fopen("eof.txt","w");
    fclose(fts);
    return 0;
}
```

para escrever dentro do arquivo existem muitas funções uma delas é o fprintf da biblioteca stdio.h que permite escrever nele formatando a saída, para usar o fprintf é parecida com a função sprintf só que colocamos o ponteiro FILE antes

```
#include <stdio.h>

int main(void)
{
    FILE *fts;
    fts = fopen("eof.txt","w");
    fprintf(fts,"visitem o forum eof\n\n");
    fprintf(fts,"by hfts%x",315);
    fclose(fts);
    return 0;
}
```

a diferença entre o tipo de acesso “w” e “a”, que o tipo de acesso “w” vai abrir o arquivo e apagar o que estiver la dentro já o tipo de acesso “a” vai escrever no final do que estiver la dentro

```
#include <stdio.h>

int main(void)
{
```

```
    FILE *fts;
    fts = fopen("eof.txt","a");
    fprintf(fts,"visitem o forum eof\n\n");
    fprintf(fts,"by hfts315\n");
    fclose(fts);
    fts = fopen("eof.txt","a");
    fprintf(fts,"se fosse o acesso w so teria essa string la dentro");
    fclose(fts);
    return 0;
}
```

podemos escrever uma string com a função fwrite da biblioteca stdio.h, porem essa função é um pouco mais complicada que o fprintf, nela temos que passar como argumento a string a ser armazenada, o tamanho do tipo de dado, o tamanho da string e o ponteiro

```
#include <stdio.h>
#include <string.h>

int main(void)
{
    FILE *fts;
    char hack[] = "esta muito calor aqui";
    fts = fopen("eof.txt","w");
    fwrite(hack,sizeof(char),strlen(hack),fts);
    fclose(fts);
    return 0;
}
```

podemos usar fputs da biblioteca stdio.h para armazenar uma string, os argumentos dela sao a string e o ponteiro

```
#include <stdio.h>

int main(void)
{
    FILE *fts;
    fts = fopen("eof.txt","w");
    fputs("sou um viciado em animes '-' ",fts);
    fclose(fts);
    return 0;
}
```

para armazenar um unico caracter podemos usar fputc da biblioteca stdio.h nele passamos como argumento o caracter depois o ponteiro

```
#include <stdio.h>

int main(void)
{
    FILE *fts;
    fts = fopen("eof.txt","w");
```

```
    fputc('e',fts);
    fputc('o',fts);
    fputc('f',fts);
    fclose(fts);
    return 0;
}
```

para fazermos o inverso ou seja ler o arquivo e armazenar em uma variavel temos diversas funções, uma delas é o fscanf da biblioteca stdio.h que ler os dados no arquivo e armazena de forma formatada em uma variavel, nos argumentos passamos o ponteiro do arquivo a strings com tipo do arquivo e por fim a variavel,

```
#include <stdio.h>

int main(void)
{
    FILE *fts;
    int hack;
    fts = fopen("eof.txt","r");
    fscanf(fts,"%d",&hack);
    fclose(fts);
    printf("%d",hack);
    return 0;
}
```

outra forma é usar o fread da biblioteca stdio.h, porem teriamos que saber o tamanho exato do arquivo a ser copiada para isso termiamos e usar duas funçoes uma é o fseek para pular para o final do arquivo e o ftell para pegar a posição (vamos aprender isso mais para frente entao vou usar um arquivo que eu ja sei o tamanho), os argumentos do fread sao a array do tipo char onde vai ser armazenado o tamanho do tipo de dado da variavel, o tamanho do arquivo decrementado - 1 (no caso o meu arquivo tem a string eof que são 3) e por fim o ponteiro do arquivo

```
#include <stdio.h>

int main(void)
{
    FILE *fts;
    char hack[100];
    fts = fopen("eof.txt","r");
    fread(hack,sizeof(hack),3 - 1,fts);
    fclose(fts);
    printf("%s",hack);
    return 0;
}
```

Com a função fgets da biblioteca stdio.h copiamos uma quantidade especifica de string, passamos como argumento para ela o ponteiro a quantidade de string e a variavel que sera armazenada

```
#include <stdio.h>

int main(void)
```

```
{
    FILE *fts;
    char hack[100];
    fts = fopen("eof.txt","r");
    fgets(hack,3,fts);
    fclose(fts);
    printf("%s",hack);
    return 0;
}
```

para copiar um unico caracter usamos fgetc da biblioteca stdio.h, so passamos como argumento o ponteiro do arquivo e atribuimos o retorno para uma variavel do tipo char

```
#include <stdio.h>

int main(void)
{
    FILE *fts;
    char hack[100];
    fts = fopen("eof.txt","r");
    hack[0] = fgetc(fts);
    hack[1] = fgetc(fts);
    hack[2] = fgetc(fts);
    hack[3] = 0x00;
    fclose(fts);
    printf("%s",hack);
    return 0;
}
```

Para criar um arquivo temporario ou seja ele sera destruido depois que o programa for finalizado usamos a função tmpfile da biblioteca stdio.h, para usar essa função basta atribuir a um ponteiro do tipo FILE, essa função é perfeita para armazenar dados temporariamente que deve ser destruido no final do programa

```
#include <stdio.h>

int main(void)
{
    FILE eof = tmpfile();
    fprintf(eof,"315");
    return 0;
}
```

para deletar um arquivo usamos a função remove da biblioteca stdio.h, nela passamos como argumento o nome ou endereço do arquivos

```
#include <stdio.h>

int main(void)
{
    remove("eof.txt");
```

```
    return 0;
}
```

para renomear usamos a função rename no argumento dela passamos o nome ou endereço do arquivo e o novo nome

```
#include <stdio.h>

int main(void)
{
    rename("eof.txt","fts.txt");
    return 0;
}
```

podemos nos movimentar dentro de um arquivo com a função fseek da biblioteca stdio.h, passamos como argumento para essa função o ponteiro do arquivo, a posição para onde ele vai pular, e a costante que indica ponto de partida do pulo (inicio = SEEK_SET, atual = SEEK_CUR, fim = SEEK_END)

```
#include <stdio.h>

int main(void)
{
    FILE *fts = fopen("eof.txt","r");
    fseek(fts,-1,SEEK_END);
    printf("ultimo caracter do arquivo = %c",fgetc(fts));
    fclose(fts);
    return 0;
}
```

com a função ftell da biblioteca stdio.h podemos retornar o a quantidade de posições ate chega na posição atual, para usar ela basta passar como argumento o ponteiro do arquivo e atribuir a uma variavel do tipo int

```
#include <stdio.h>

int main(void)
{
    FILE *fts = fopen("eof.txt","r");
    int eof;
    fseek(fts,5,SEEK_SET);
    eof = ftell(fts);
    printf("%d",eof);
    fclose(fts);
    return 0;
}
```

outra forma de pegar a posição atual é usar a função fgetpos da biblioteca stdio.h, nessa função passamos como argumento o ponteiro do arquivo, e o endereço de memoria de uma variavel do tipo fpos_t

```
#include <stdio.h>

int main(void)
{
    FILE *fts = fopen("eof.txt","r");
    fpos_t eof;
    fseek(fts,3,SEEK_SET);
    fgetpos(fts,&eof);
    printf("%d",eof);
    fclose(fts);
    return 0;
}
```

tambem podemos avançar para determinada posição com a função fsetpos, passamor como argumento para ela o ponteiro do arquivo e o endereço de memoria de uma variavel do tipo fpos_t com a posição

```
#include <stdio.h>

int main(void)
{
    FILE *fts = fopen("eof.txt","r");
    fpos_t eof = 4;
    fsetpos(fts,&eof);
    printf("%d",ftell(fts));
    fclose(fts);
    return 0;
}
```

Com a função rewind da biblioteca stdio.h voltamos para a posição inicial, nela so passamos como argumento o ponteiro do arquivo

```
#include <stdio.h>

int main(void)
{
    FILE *fts = fopen("eof.txt","r");
    fpos_t eof = 4;
    fsetpos(fts,&eof);
    rewind(fts);
    printf("%d",ftell(fts));
    fclose(fts);
    return 0;
}
```

com a função feof da biblioteca stdio.h retornamos verdadeiro quando estiver no final do arquivo, para usar essa função so passamos o ponteiro do arquivos e pegamos o retorno

```
#include <stdio.h>

int main(void)
```

```
{
    FILE *fts = fopen("eof.txt","r");
    while(1)
    {
        if(feof(fts))
        {
            break;
        }
        printf("%c",fgetc(fts));
    }
    fclose(fts);
    return 0;
}
```

uma coisa que se deve tomar cuidado na hora de carregar um arquivo e colocar ele em uma array, essa array deve ser maior que quantidade de caracteres do arquivo para evitar falha buffer overflow e falha de segmentação, para evitar isso podemos abrir o arquivo ver o tamanho ate o ultimo caracter e alocar dinamicamente com a função malloc

```
#include <stdio.h>

int main(void)
{
    FILE *fts;
    char *eof;
    int tam = 0;
    fts = fopen("eof.txt","r");
    fseek(fts,0,SEEK_END);
    tam = ftell(fts);
    rewind(fts);
    eof = malloc(sizeof(char) * tam + 1);
    return 0;
}
```

na linguagem C++ podemos usar a biblioteca fstream, para abrir um arquivos instanciamos o fstream, depois usamos a função open passamos como argumento o arquivo depois o tipo de acesso que é (ios::in seria o w, ios::out seria o r, ios::binary seria o b, ios::app seria o a), no tipo de acesso ser houver dois separamos pelo pipe, tambem usamos a função close para fechar

```
#include <iostream>
#include <fstream>

using namespace std;

int main(void)
{
    fstream fts;
    fts.open("eof.txt",ios::out | ios::app);
    fts.close();
    return 0;
}
```

para escrevermos no arquivo basta redirecionar o fluxo que seria o mesmo do cout para o objeto instanciado

```
#include <iostream>
#include <fstream>

using namespace std;

int main(void)
{
    fstream fts;
    fts.open("eof.txt",ios::out | ios::app);
    fts << "eof ";
    fts.close();
    return 0;
}
```

para ler o arquivo fazemos o inverso

```
#include <iostream>
#include <fstream>

using namespace std;

int main(void)
{
    fstream fts;
    char hack[100];
    fts.open("eof.txt",ios::in);
    fts >> hack;
    fts.close();
    cout << hack;
    return 0;
}
```

é possivel descobrir os atributos do arquivo como tamanho, ultima vez que foi acessado ou modificado, inode que seria uma especie de endereço logico no linux, tipo de permissão etc, para isso vamos usar a estrutura stat e a função stat da biblioteca sys/stat.h embora essa biblioteca seja do linux ela tambem existe nos compiladores do windows, para começar declaramos a struct stat depois usamos a função stat passando como argumento para ela uma string que seria o arquivo, e o endereço de memoria da struct, por fim usamos os atributos da estrutura stat que são st_size para retornar o tamanho em bits, st_atime para o tempo do ultimo acesso (tempo em segundos deis de 1970), st_ino para o inode (so funciona no linux), st_nlink quantidade de links (so funciona no linux), st_uid para identificador do dono (so funciona no linux) e st_gid para identificar o grupo (so funciona no linux), existem outros atributos alem dos que eu citei

```
#include <stdio.h>
#include <sys/stat.h>
#include <time.h>
```

```
int main(void)
{
    struct stat fts;
    time_t acesso, modifi;
    stat("eof.txt",&fts);
    printf("tamanho: %d \n",fts.st_size);
    acesso = fts.st_atime;
    modifi = fts.st_mtime;
    printf("acesso: %s \n",ctime(&acesso));
    printf("modificado: %s \n",ctime(&modifi));
    printf("inode: %d \n",fts.st_ino);
    printf("qnt links: %d \n",fts.st_nlink);
    printf("id do dono: %d \n",fts.st_uid);
    printf("id do grupo: %d \n",fts.st_gid);
    return 0;
}
```

7.4 – Manipulando diretorio

Para manipular diretorio no sistema windows usamos a biblioteca dirent.h ja no sistema linux usamos sys/dir.h, para listar os arquivos de um diretorio temos que abrir esse diretorio com a função opendir da biblioteca dirent.h (sys/dir.h no caso do linux), passamos como argumento o diretorio que vamos abrir e atribuimos o retorno da função para um ponteiro do tipo DIR, podemos usar o closedir para fechar o diretorio nele passamos como argumento o ponteiro DIR

```
#include <stdio.h>
#include <dirent.h>

int main(void)
{
    DIR *eof;
    eof = opendir("c:\\");
    closedir(eof);
    return 0;
}
```

depois basta usar a função readdir da biblioteca dirent.h (sys/dir.h), passando como argumento para ela o ponteiro que atribuimos o diretorio e por fim atribuimos o retorno para um ponteiro de uma estrutura do tipo dirent, como a função readdir ler arquivo por arquivo usar ela em um laço é a melhor alternativa

```
#include <stdio.h>
#include <dirent.h>

int main(void)
{
    DIR *eof;
    struct dirent *fts;
    eof = opendir("c:\\");
    while(fts = readdir(eof))
    {
```

```
    }
    closedir(eof);
    return 0;
}
```

para ler o nome dos arquivos basta usar a array d_name da estrutura dirent

```
#include <stdio.h>
#include <dirent.h>

int main(void)
{
    DIR *eof;
    struct dirent *fts;
    eof = opendir("c:\\");
    while(fts = readdir(eof))
    {
        printf("%s \n",fts->d_name);
    }
    closedir(eof);
    return 0;
}
```

tambem é possivel conseguir o inode do arquivo ou diretorio pela estrutura dirent pela varaivel d_ino, porem não é funcional no windows

```
#include <stdio.h>
#include <sys/dir.h>

int main(void)
{
    DIR *eof;
    struct dirent *fts;
    eof = opendir("/");
    while(fts = readdir(eof))
    {
        printf("nome: %s \ninode: %d \n\n",fts->d_name,fts->d_ino);
    }
    closedir(eof);
    return 0;
}
```

7.5 – Manipulando o terminal

O uso do terminal pode faciltar na manipulação do sistema operacional podendo agilizar uso de API para programas ja criado, para usar o terminal existe a função system da biblioteca stdlib.h, essa função permite passar como argumento o comando que seria usado no termina e depois exibi no programa a saida

```
#include <stdio.h>
#include <stdlib.h>
```

```
int main(void)
{
    system("dir");
    return 0;
}
```

a cada função system seria equivalente a um novo terminal aberto entao não da para acessar informação de determinado system pelo outro system, no exemplo abaixo o primeiro system vai para o diretorio anterior e o segundo system ler o diretorio, o segundo system vai ler o diretorio que esta o executavel e nao o diretorio anterior que esta o primeiro system

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    system("cd ..");
    system("dir");
    return 0;
}
```

para evitar o exemplo acima podemos usar comandos no mesmo system separados, no sistema windows usamos duas vezes o "e" comercial (&&) no linux usamos ponto e virgula (;)

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    system("cd .. && dir");
    return 0;
}
```

o system vai enviar o fluxo diretamente para tela do computador sem a possibilidade de manipular diretamente, mais podemos manipular indiretamente armazenando o fluxo para o HD com o operador de saida do terminal que é o simbolo de maior seguido do nome do arquivo, depois abrimos o arquivo e pegamos os dados novamente

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    FILE *fts;
    char *hack;
    int tam, cont = 0;
    system("dir > eof.txt");
    fts = fopen("eof.txt","r");
    fseek(fts,0,SEEK_END);
    tam = ftell(fts);
```

```
    rewind(fts);
    hack = malloc(sizeof(char) * tam + 1);
    while(1)
    {
         if(feof(fts))
        {
            break;
        }
        hack[cont] = fgetc(fts);
        cont++;
    }
    fclose(fts);
    printf("%s",hack);
    return 0;
}
```

podemos tambem usar a função popen da biblioteca stdio.h para manipula a saida do terminal, o popen e paracido com a manipulação de arquivo, usamos um ponteiro do tipo FILE para atribuir ele, passamos como argumento o comando para função popen e o tipo de acesso, depois e so ler o ponteiro como se fosse um arquivo, e para fechar usamos pclose passando como argumento o ponteiro

```
#include <stdio.h>

int main(void)
{
    FILE *fts;
    fts = popen("cd .. && dir","r");
    while(1)
    {
        if(feof(fts))
        {
             break;
        }
        printf("%c",fgetc(fts));
    }
    pclose(fts);
    return 0;
}
```

8.0 – Orientação a objeto

orientação a objeto tambem chamado POO (Programação Orientada a Objeto) permite manipular o codigo como um todo, permitindo abstrair e organizar determinadas partes como uma unica estrutura ou unica classe, dessa estrutura podera ser gerado o objeto ou instacia, é pelo objeto que permite manipular a classe e cada objeto gerado da mesma classe não afetara o outro, as variveis e funções usando o conceito de orientação a objeto devera ser chamada de atributos (variaveis) e metodos (funções) embora chama-las de variaveis e funções tambem é aceitavel porem seja mais correto o termo atributos e metodos, uma classe pode receber os atributos e metodos de outra classe isso é chamado herança e é forma facil de reaproveitar o codigo e não precisar rescreve-lo, com o encapsulamento permitimos escolher como deve ser acessado a classe e na linguagem C++ existe 3

tipos de acesso que são public (publico), private (privado), protected (protegido), tambem existe o polimorfismo que é quando determinada atributo ou metodo asume duas ou mais formas, existem os construtores que permite iniciar um valor dentro de um atributo ou chamar determinado metodo quando declarado

8.1 – Orientação a objeto: Class

Como dito antes as classes são as estruturas usadas na orientação a objeto da linguagem C++, para se criar uma class é a mesma coisa que criar uma estrutura struct declaramos o tipo da estrutura que no caso é class seguida do nome e depois um escopo e por fim um ponto e virgula

```
#include <iostream>

using namespace std;

class eof
{
};

int main(void)
{
    return 0;
}
```

para instanciar ou gerar o objeto basta colocar o nome da class seguido do nome da instancia

```
#include <iostream>

using namespace std;

class eof
{
};

int main(void)
{
    eof objeto;
    return 0;
}
```

tambem poderiamos instanciar com ponteiros

```
#include <iostream>

using namespace std;

class eof
{
};

int main(void)
```

```
{
    eof *objeto;
    return 0;
}
```

ou dinamicamente com o operador new e liberar com o operador delete

```
#include <iostream>

using namespace std;

class eof
{
};

int main(void)
{
    eof *objeto = new eof();
    delete objeto;
    return 0;
}
```

8.2 – Orientação a objeto: Encapsulamento

O encapsulamento permite restringir determinados acesso dentro da classe isso permite manipular como deve ser acessado determinados metodos e atributos, em teoria nao precisamos saber como funciona internamente um computador para poder usalo o encapsulamento permite fazer isso nao é necessario saber como a class funciona para poder usala, existem 3 tipos acessos na linguagem C++ que são os public, private e protected, os public permite qualquer acesso onde a class foi instanciada ou herdada, os private permite acesso apenas dentro da mesma class, e os protected permite o acesso dentro da mesma class e as class herdeira, para definir o tipo de acesso colocamos o tipo de acesso seguido de dois pontos (se nao definimos nenhum tipo o compilador vai definir o tipo padrao que seria o private)

```
#include <iostream>

using namespace std;

class eof
{
    public:
};

int main(void)
{
    return 0;
}
```

poderiamos definir os outros tipos dentro tambem

```
#include <iostream>
```

```
using namespace std;

class eof
{
    public:
    private:
    protected:
};

int main(void)
{
    return 0;
}
```

não existe uma ordem e nem um limite para o mesmo tipo

```
#include <iostream>

using namespace std;

class eof
{
    protected:
    public:
    private:
    protected:
    public:
};

int main(void)
{
    return 0;
}
```

os atributos e metodos teram a permissão do ultimo tipo de acesso antes dele

```
#include <iostream>

using namespace std;

class eof
{
    public: int fts;
    public: int lol;
    private: int hack;
};

int main(void)
{
    return 0;
}
```

quando o tipo de acesso é o mesmo que o anterior nao precisamos definir o tipo de acesso denovo

```
#include <iostream>

using namespace std;

class eof
{
    public:
        int fts;
        int lol;
    private: int hack;
};

int main(void)
{
    return 0;
}
```

para acessar atributos é a mesma coisa que a estrutura struct, no caso usamos ponto, caso seja ponteiro usamos hífen e maior (->) depois o atributo desejado

```
#include <iostream>

using namespace std;

class eof
{
    public: int fts;
};

int main(void)
{
    eof a;
    eof *b;
    eof *c = new eof();
    a.fts = 315;
    b->fts = 100;
    c->fts = 150;
    cout << a.fts << endl;
    cout << b->fts << endl;
    cout << c->fts << endl;
    delete c;
    return 0;
}
```

os métodos também são iguais

```
#include <iostream>
```

```
using namespace std;

class eof
{
    public:
        void fts(void)
        {
            cout << "alo testando, testando 123";
        }
};

int main(void)
{
    eof hack;
    hack.fts();
    return 0;
}
```

podemos criar métodos fora do escopo da class, para fazer isso temos que colocar um prototipo dela dentro da class e no método temos que colocar o nome da class junto com o nome do metodo separado por duas vezes o ponto (::)

```
#include <iostream>

using namespace std;

class eof
{
    public:
        void fts(void);
};

void eof::fts(void)
{
    cout << "alo testando, testando 123";
}

int main(void)
{
    eof hack;
    hack.fts();
    return 0;
}
```

a maneira anterior vai alocar a class e o método em partes diferente na memória então não é muito recomendado fazer isso, é possível alocar os dois juntos mesmo separados daquela forma para isso podemos usar o inline

```
#include <iostream>

using namespace std;
```

```
class eof
{
    public:
        void fts(void);
};

inline void eof::fts(void)
{
    cout << "alo testando, testando 123";
}

int main(void)
{
    eof hack;
    hack.fts();
    return 0;
}
```

também podemos criar uma class dentro da class

```
#include <iostream>

using namespace std;

class eof
{
    public:
        class fts
        {
            public:
                int hack;
        };
};

int main(void)
{
    return 0;
}
```

para acessar a segunda class temos que instanciar ela, para isso colocamos a primeira class seguida de dois pontos e a segunda class

```
#include <iostream>

using namespace std;

class eof
{
    public:
        class fts
```

```
        {
            public:
                int hack;
        };
};

int main(void)
{
    eof::fts lol;
    lol.hack = 315;
    cout << lol.hack;
    return 0;
}
```

podemos criar varias class dentro da outra

```
#include <iostream>

using namespace std;

class eof
{
    public:
        class fts
        {
            public:
                class hack
                {
                        public:
                            int valor;
                };
        };
};

int main(void)
{
    eof::fts::hack lol;
    lol.valor = 315;
    cout << lol.valor;
    return 0;
}
```

se a gente colocar o identificador friend antes do metodo não precisamos instanciar a class para usar aquele método

```
#include <iostream>

using namespace std;

class eof
{
```

```
    public:
        friend void fts(void)
        {
            cout << "a vida e um algoritimo insano '-' ";
        }
};

int main(void)
{
    fts();
    return 0;
}
```

ate agora a gente só tá usando o acesso publico vamos usar o acesso privado um pouco, a declaração dos dois são as mesmas como dito antes, a diferença que o acesso publico pode ser usado em qualquer parte do código deis de que seja instanciado, já o privado só poderá ser usado dentro da própria class, um exemplo seria criar um método publico para entrar com o valor esse valor seria armazenado dentro de um atributo privado o mesmo método anterior chamaria outro método privado que faria uma conta e armazenaria em um atributo publico, em outras palavras a gente só teria acesso publico à entrada de dados e o resultado e não ao processo da conta

```
#include <iostream>

using namespace std;

class eof
{
     public:
          void entrada(int valor)
          {
               fts = valor;
               quadrado();
          }
          int resultado;
     private:
          int fts;
          void quadrado(void)
          {
               resultado = fts * fts;
          }
};

int main(void)
{
    eof objeto;
    objeto.entrada(5);
    cout << objeto.resultado;
    return 0;
}
```

no exemplo acima não precisamos saber como a class toda funciona a única coisa necessária é uso

do método “entrada” e do atributo “resultado”, existe também os acesso protegido que vamos ver mais para frente em herança

8.3 – Orientação a objeto: Herança

A herança permite que determinada class receba todos os atributos e metodos publicos e protegidos de uma outra class, isso permite deixar o código reutilizável, para herdar uma class depois do nome dela colocamos dois pontos (:) seguido do tipo de acesso daquela class (maioria das vezes é publico), e por fim o nome da class da onde esta herdando os metodos e atributos, no exemplo abaixo a class fts esta herdando os métodos e atributos da class eof, ou seja ele herdo apenas o método hack

```
#include <iostream>

using namespace std;

class eof
{
    public:
        int hack(int valor)
        {
            return valor * valor;
        }
};

class fts: public eof
{
};

int main(void)
{
    return 0;
}
```

para usar e mesma coisa que tivesse criado esses métodos e atributos dentro da class

```
#include <iostream>

using namespace std;

class eof
{
    public:
        int hack(int valor)
        {
            return valor * valor;
        }
};

class fts: public eof
{
```

```
};

int main(void)
{
    fts lol;
    cout << lol.hack(315);
    return 0;
}
```

no exemplo acima eu instanciei a class fts e usei o método que ele herdo, o uso de herança pode agilizar criação de novos métodos, por exemplo o codigo abaixo eu faço um novo método retornando o cubo do número para isso eu uso o método herdado

```
#include <iostream>

using namespace std;

class eof
{
    public:
        int hack(int valor)
        {
            return valor * valor;
        }
};

class fts: public eof
{
    public:
        int cubo(int numero)
        {
            return numero * hack(numero);
        }
};

int main(void)
{
    fts lol;
    cout << lol.cubo(5);
    return 0;
}
```

podemos herdar mais de uma class para isso basta separar por vírgula

```
#include <iostream>

using namespace std;

class eof
{
    public:
```

```
        int hack(int valor)
        {
            return valor * valor;
        }
};

class matematica
{
    public:
        float pi;
};

class fts: public eof, public matematica
{
};

int main(void)
{
    fts lol;
    lol.pi = 3.1415;
    cout << lol.pi;
    return 0;
}
```

quando uma class herda duas ou mais class e essas class tem um metodo ou atributo com o mesmo nome isso pode gerar ambiguidade no codigo e pode da erro, para evitar devemos especificar o nome da class seguido de duas vezes dois pontos (::) e o atributo ou metodo

```
#include <iostream>

using namespace std;

class fts
{
    public:
        int valor;
};

class eof
{
    public:
        int valor;
};

class hack: public fts, public eof
{
    public:
};


int main(void)
```

```
{
    hack lol;
    lol.fts::valor = 315;
    lol.eof::valor = 100;
    cout << lol.fts::valor << endl;
    cout << lol.eof::valor << endl;
    return 0;
}
```

também podemos endereçar um class herdada para isso basta gerar o objeto das class herdada e atribuir o outro objeto que herdo elas

```
#include <iostream>

using namespace std;

class fts
{
    public:
        int valor;
};

class eof
{
    public:
        int valor;
};

class hack: public fts, public eof
{
    public:
};

int main(void)
{
    hack lol;
    fts heranca1 = lol;
    eof heranca2 = lol;
    heranca1.valor = 315;
    heranca2.valor = 100;
    cout << heranca1.valor << endl;
    cout << heranca2.valor << endl;
    return 0;
}
```

como dito o tipo de acesso protegido ou protected pode ser acessado apenas pela mesma class ou pelas class herdada, no exemplo abaixo eu não poderia usar diretamente o metodo hack embora eu poderia herdar ele e criar um metodo publico para chamar ele

```
#include <iostream>
```

```
using namespace std;

class eof
{
    protected:
        int hack(int valor)
        {
            return valor * valor;
        }
};

class fts: public eof
{
    public:
        int flavio(int x)
        {
            return hack(x);
        }
};

int main(void)
{
    fts f;
    cout << f.flavio(315);
    return 0;
}
```

8.4 – Orientação a objeto: Construtores

Os construtores permite iniciar valores nos atributos ou chamar metodos assim que o objeto for instanciado para se criar um construtor basta criar um metodo sem tipo de retorno com o mesmo nome da class

```
#include <iostream>

using namespace std;

class eof
{
    public:
        eof(void)
        {
        }
};

int main(void)
{
    eof lol;
    return 0;
}
```

e dentro do construtor atribuímos os valores aos atributos

```
#include <iostream>

using namespace std;

class eof
{
    public:
        eof(void)
        {
            fts = 315;
        }
        int fts;
};

int main(void)
{
    eof lol;
    cout << lol.fts;
    return 0;
}
```

também é possível chamar métodos pelo construtor

```
#include <iostream>

using namespace std;

class eof
{
    public:
        eof(void)
        {
            fts();
        }
        void fts(void)
        {
            cout << "boku wa fts desu";
        }
};

int main(void)
{
    eof lol;
    return 0;
}
```

podemos passar valores pelo construtor assim que instanciar

```
#include <iostream>

using namespace std;

class eof
{
    public:
        eof(int x)
        {
            fts = x;
        }
        int fts;
};

int main(void)
{
    eof lol(315);
    cout << lol.fts;
    return 0;
}
```

também existem os destrutores que são chamados quando o objeto for destruído, para criar um destrutor criamos um método com o mesmo nome da class igual ao construtor porém colocamos um til (~) antes do nome

```
#include <iostream>

using namespace std;

class eof
{
    public:
        eof()
        {
            construido();
        }
        ~eof()
        {
            destruido();
        }
        void construido(void)
        {
            cout << "foi construido ^^ " << endl;
        }
        void destruido(void)
        {
            cout << "foi destruido '-' " << endl;
        }
};

int main(void)
```

```
{
    eof *fts = new eof();
    delete fts;
    return 0;
}
```

8.5 – Orientação a objeto: Polimorfismo

Polimorfismo é quando determinado objeto, metodo, atributo, evento ou qualquer outra coisa assume duas ou mais formas, na linguagem C++ os objeto polimórfico é usado o identificador virtual, um dos metodos virtual é fazer que determinado metodo a cada herança ele faça ação diferente, para fazer isso declaramos ele como virtual na class, e depois recriamos o prototipo dele nas class que herdar

```
#include <iostream>

using namespace std;

class eof
{
    public:
        virtual void lol(void)
        {
        }
};

class fts: public eof
{
    public:
        virtual void lol(void)
        {
            cout << "class fts" << endl;
        }
};

class hack: public eof
{
    public:
        virtual void lol(void)
        {
            cout << "class hack" << endl;
        }
};

class fbi: public hack, public fts
{
    public:
};

int main(void)
{
```

```
    fbi objeto;
    objeto.hack::lol();
    objeto.fts::lol();
    return 0;
}
```

tambem é possível colocar acesso virtual para uma class herdada, assim todas as class acessara a mesma sem gera ambiguidade no código

```
#include <iostream>

using namespace std;

class eof
{
    public:
        int valor;
};

class fts: virtual public eof
{
    public:
};

class hack: virtual public eof
{
    public:
};

class lol: public hack, public fts
{
    public:
};

int main(void)
{
    lol objeto;
    objeto.valor = 315;
    cout << objeto.valor << endl;
    cout << objeto.hack::valor << endl;
    cout << objeto.fts::valor << endl;
    return 0;
}
```

uma estrutura polimorfica é o union que permite ter mais de um tipo para o mesmo endereço de memoria (embora só possa assumir um valor por vez)

```
#include <iostream>

using namespace std;
```

```
union eof
{
   int inteiro;
   float quebrado;
};

int main(void)
{
    eof fts;
    fts.inteiro = 315;
    cout << fts.inteiro << endl;
    fts.quebrado = 3.15;
    cout << fts.quebrado << endl;
    return 0;
}
```

9.0 – API Windows

Uma API é uma função ou procedimento de determinada plataforma que só funciona para aquela plataforma e nenhuma outra, uma API é uma função do próprio sistema operacional e não da linguagem ou de uma biblioteca da linguagem, o sistema operacional windows existem milhares de APIs como por exemplo movimento do mouse, apertar de botao ou simplesmente simular aperta de botão, desligar ou ligar sistemas e drives, manipular o editor de registro, acessar tipos arquivos que só exitem no sistema windows, executar programas, manipular a memoria, criar e deletar usuario ou arquivos, interagir com hardware entre outras milhares de coisas que só é possível no sistema windows, boa parte das APIs do sistema windows fica dentro de arquivos chamados DLL e se faltar uma dessas DLL o sistema operacional não podera executar determinadas tarefas, e boa parte das API do windows pode terminar com a letra A indicando ASC ou a letra W indicando UNICODE por exemplo CopyFileA ou CopyFileW, para usar boa parte das API na linguagem C/C++ temos que declarar a biblioteca “windows.h” essa biblioteca chama outras bibliotecas que acessa as DLL de forma facil, como dito antes no sistema operacional windows existem milhares de APIs não é possível eu colocar todas aqui mesmo por que só conheço um numero bem limitado de API, a primeira API que vamos ver vai ser a CopyFile da biblioteca windows.h que permite copiar determinado arquivo para outro diretorio, nessa API passamos como argumento o arquivo a ser copiado e o destino tambem existe um outro argumento que seria uma checagem se vai copiar o arquivo caso exista ou não nesse argumento usamos a constante TRUE ou FALSE da biblioteca windows.h (no exemplo abaixo eu to colocando apenas o nome da pasta, ou seja, essa pasta esta no mesmo diretório do executável porém eu tambem poderia colocar o endereço completo)

```
#include <stdio.h>
#include <windows.h>

int main(void)
{
    CopyFileA("fts.txt","arquivo\\fts.txt",FALSE);
    return 0;
}
```

tambem existe o tipo BOOL da biblioteca windows.h, e as constante TRUE e FALSE que seria equivalente a biblioteca stdbool.h

```
#include <stdio.h>
#include <windows.h>

int main(void)
{
    BOOL verdadeiro = TRUE;
    BOOL falso = FALSE;
    printf("v = %d \nf = %d",verdadeiro,falso);
    return 0;
}
```

alem da API CopyFile existe a API MoveFile que move o arquivo, para usar essa API passamos como argumento o arquivo a ser copiado e o destino

```
#include <stdio.h>
#include <windows.h>

int main(void)
{
    MoveFileA("fts.txt","arquivo\\fts.txt");
    return 0;
}
```

existe API DeleteFile que remove o arquivo, o argumento dela é o arquivo a ser deletado

```
#include <stdio.h>
#include <windows.h>

int main(void)
{
    DeleteFileA("fts.txt");
    return 0;
}
```

com a API RemoveDirectory podemos remover um diretório, os argumentos dela é o diretório a ser removido

```
#include <stdio.h>
#include <windows.h>

int main(void)
{
    RemoveDirectoryA("arquivo");
    return 0;
}
```

existe API CreateDirectory que cria um diretório, os argumentos dela são o nome do diretório e o tipo de atributo de segurança (não vou me aprofundar nisso então podemos colocar um 0 nele)

```
#include <stdio.h>
```

```
#include <windows.h>

int main(void)
{
    CreateDirectoryA("arquivo",0);
    return 0;
}
```

com a API GetCurrentDirectory retornamos o diretório atual, nele passamos como argumento o tamanho da array onde será armazenado e a array do tipo char

```
#include <stdio.h>
#include <windows.h>

int main(void)
{
    char fts[100];
    GetCurrentDirectoryA(100,fts);
    printf("%s",fts);
    return 0;
}
```

para mudar o diretório atual usamos a API SetCurrentDirectory e passamos como argumento o diretório

```
#include <stdio.h>
#include <windows.h>

int main(void)
{
    char fts[100];
    SetCurrentDirectoryA("c:\\");
    GetCurrentDirectoryA(100,fts);
    printf("%s",fts);
    return 0;
}
```

para a gente executar um programa ou abrir um diretorio ou arquivo podemos usar API ShellExecute passamos como argumento o handle para janela (não vamos usar isso por enquanto então colocamos 0), passamos como argumento de tipo de acesso (open = executa, explore = abri caso seja diretório, existem outros além desses dois), o nome do arquivo ou diretório, o argumento do arquivo (esse argumento são para aqueles programa de comando), o diretorio corrente (podemos colocar 0 caso definimos o endereço completo do arquivo), e como ele sera chamado (SW_SHOW = normal, SW_HIDE = oculto, SW_MAXIMIZE = maximizado, SW_MINIMIZE = minimizado)

```
#include <stdio.h>
#include <windows.h>

int main(void)
{
    ShellExecuteA(0,"open","c:\\windows\\system32\\cmd.exe","/c dir && pause",0,SW_SHOW);
```

```
    return 0;
}
```

agora usando o ShellExecute para abrir uma pasta

```
#include <stdio.h>
#include <windows.h>

int main(void)
{
    ShellExecuteA(0,"explore","c:\\",0,0,SW_SHOW);
    return 0;
}
```

também é possível executar programas com a função WinExec nele passamos como argumento comando a ser executado e controle da janela (não vou me aprofundar nisso por enquanto então colocamos 0)

```
#include <stdio.h>
#include <windows.h>

int main(void)
{
    WinExec("c:\\windows\\system32\\cmd.exe /c dir && pause",0);
    return 0;
}
```

para mudar o título da janela do console usamos a API SetConsoleTitle e passamos como argumento o novo título, porém essa API só vai mudar o título enquanto o programa estiver rodando

```
#include <stdio.h>
#include <conio.h>
#include <windows.h>

int main(void)
{
    SetConsoleTitle("forum eof");
    getch();
    return 0;
}
```

para mudar a cor do terminal tanto o texto (foreground) quanto o fundo (background), temos que pegar o handle da janela para isso usamos a API GetStdHandle passamos como argumento para ela a constante STD_OUTPUT_HANDLE e atribuímos a função para uma variável do tipo HANDLE, para terminar usamos a API SetConsoleTextAttribute para mudar a cor, passamos como argumento para ela a variável do tipo HANDLE e um número em hexadecimal de 0x00 ate 0xff (o primeiro número muda a cor do fundo e o segundo muda a cor do texto, exemplo 0x64 o número 6 vai coloca o fundo amarelo e o 4 a cor do texto vermelha)

```
#include <stdio.h>
```

```
#include <windows.h>

int main(void)
{
    HANDLE fts;
    fts = GetStdHandle(STD_OUTPUT_HANDLE);
    printf("texto com a cor padrao do terminal");
    SetConsoleTextAttribute(fts,0x02);
    printf("fundo preto cor verde");
    SetConsoleTextAttribute(fts,0xf5);
    printf("fundo branco cor roxo");
    SetConsoleTextAttribute(fts,0x64);
    printf("fundo amarelo cor vermelho");
    return 0;
}
```

podemos mudar a posição do cursor do terminal tambem usando a API SetConsoleCursorPosition, nessa função passamos como argumento o handle da janela que no caso teria que usar a API GetStdHandle para conseguir, e uma estrutura _COORD com a posição X e Y

```
#include <stdio.h>
#include <windows.h>

int main(void)
{
    struct _COORD eof;
    HANDLE fts;
    eof.X = 3;
    eof.Y = 5;
    fts = GetStdHandle(STD_OUTPUT_HANDLE);
    SetConsoleCursorPosition(fts,eof);
    return 0;
}
```

para criar uma thread (subprocesso) usamos a API CreateThread, nele passamos como argumento o atributo de segurança (vamos colocar 0 aqui), o tamanho do stack (tambem vamos colocar um 0), a função que sera chamada no thread (temos que usar typecast para LPTHREAD_START_ROUTINE), o argumento da função (podemos colocar 0), como sera iniciado a thread (podemos colocar 0 para iniciar assim que usar função ou usar CREATE_SUSPENDED para começar a thread pausado), e o último argumento é o endereço de memória de uma variável do tipo DWORD que seria o pid da thread, para finalizar atribuimos a função para uma variável do tipo HANDLE para poder manipular mais tarde

```
#include <stdio.h>
#include <windows.h>

void fts(void)
{
     printf("processo filho\n");
}
```

```
int main(void)
{
    DWORD pid;
    HANDLE filho;
    printf("processo pai\n");
    filho = CreateThread(0,0,(LPTHREAD_START_ROUTINE)fts,0,0,&pid);
    return 0;
}
```

para evitar que o thread pai se finalize antes do filho podemos usar a API WaitForSingleObject no final do codigo, nela passamos como argumento a variavel do tipo HANDLE e a constante INFINITE ou tempo em milésimos

```
#include <stdio.h>
#include <windows.h>

void fts(void)
{
     printf("processo filho\n");
}

int main(void)
{
    DWORD pid;
    HANDLE filho;
    printf("processo pai\n");
    filho = CreateThread(0,0,(LPTHREAD_START_ROUTINE)fts,0,0,&pid);
    WaitForSingleObject(filho,INFINITE);
    return 0;
}
```

Podemos iniciar a thread pausada usando a constante CREATE_SUSPENDED, e podemos iniciar com a API ResumeThread nela passamos como argumento a variavel HANDLE

```
#include <stdio.h>
#include <windows.h>

void fts(void)
{
     printf("processo filho\n");
}

int main(void)
{
    DWORD pid;
    HANDLE filho;
    printf("processo pai\n");
    filho = CreateThread(0,0,
(LPTHREAD_START_ROUTINE)fts,0,CREATE_SUSPENDED,&pid);
    ResumeThread(filho);
    WaitForSingleObject(filho,INFINITE);
```

```
    return 0;
}
```

também poderíamos pausar uma thread com a API SuspendThread, para usar ela basta passar como argumento a variavel HANDLE da thread

```
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>

void fts(void)
{
     while(1)
     {
         printf("loop infinito >.< ");
     }
}

int main(void)
{
    DWORD pid;
    HANDLE filho;
    filho = CreateThread(0,0,(LPTHREAD_START_ROUTINE)fts,0,0,&pid);
    sleep(2000);
    SuspendThread(filho);
    WaitForSingleObject(filho,INFINITE);
    return 0;
}
```

para finalizar uma thread podemos usar ExitThread dentro dela e passamos como argumento o retorno

```
#include <stdio.h>
#include <windows.h>

void fts(void)
{
     int eof = 0;
     while(1)
     {
         if(eof >= 10)
         {
               ExitThread(0);
         }
         printf("loop = %d",eof);
         eof++;
     }
}

int main(void)
{
```

```
    DWORD pid;
    HANDLE filho;
    filho = CreateThread(0,0,(LPTHREAD_START_ROUTINE)fts,0,0,&pid);
    WaitForSingleObject(filho,INFINITE);
    return 0;
}
```

poderíamos finalizar a thread filho pela thread pai com a função TerminateThread nela passamos como argumento o handle da thread seguido do valor de retorno

```
#include <stdio.h>
#include <windows.h>

void fts(void)
{
     while(1)
     {
         printf("loop infinito >.< ");
     }
}

int main(void)
{
    DWORD pid;
    HANDLE filho;
    filho = CreateThread(0,0,(LPTHREAD_START_ROUTINE)fts,0,0,&pid);
    TerminateThread(filho,0);
    WaitForSingleObject(filho,INFINITE);
    return 0;
}
```

com a API GetCurrentThread ela retorna o handle da thread atual

```
#include <stdio.h>
#include <windows.h>

HANDLE filho;

void fts(void)
{
    filho = GetCurrentThread();
}

int main(void)
{
    DWORD pid;
    CreateThread(0,0,(LPTHREAD_START_ROUTINE)fts,0,0,&pid);
    TerminateThread(filho,0);
    WaitForSingleObject(filho,INFINITE);
    return 0;
}
```

quando temos mais de uma thread usar a API WaitForSingleObject não funcionaria muito bem então temos que usar a API WaitForMultipleObjects, o uso dessa a API é um pouco diferente da outra porém seu objetivo é o mesmo, nas variaveis HANDLE e DWORD criamos uma array para armazenar ambas threads, e na API WaitForMultipleObjects passamos como argumento a quantidade de threads, a array com a handle das threads, TRUE ou FALSE para esperar todas as threads finalizar para continuar, e o tempo de espera (podemos usar a constante INFINITE)

```
#include <stdio.h>
#include <windows.h>

void fts(void)
{
    printf("thread fts\n");
}

void hack(void)
{
    printf("thread hack\n");
}

int main(void)
{
    HANDLE filho[10];
    DWORD pid[10];
    filho[0] = CreateThread(0,0,(LPTHREAD_START_ROUTINE)fts,0,0,&pid[0]);
    filho[1] = CreateThread(0,0,(LPTHREAD_START_ROUTINE)hack,0,0,&pid[1]);
    WaitForMultipleObjects(2,filho,TRUE,INFINITE);
    return 0;
}
```

no código acima podemos ver que ambas as threads vai mostra algo na tela porem se ambas threads tentar mostra algo na tela ao mesmo tempo isso vai impedir que uma mostre (isso é bug muito comum conhecido como deadlock), para evita isso podemos usar mutex que vai deixar apenas um evento por vez parando os demais, para a gente criar um mutex temos que criar uma variavel do tipo HANDLE para atribuir ele e usar a API CreateMutex, passamos como argumento para essa api o atributo de segurança (podemos colocar 0), TRUE ou FALSE para começar com mutex parado, é o nome (podemos colocar 0 tambem), e no começo de cada thread usamos a API WaitForSingleObject passamos como argumento o handle do mutex e constante INFINITE isso vai pausar os demais eventos, e no final da thread colocamos a API ReleaseMutex com o argumento handle do mutex isso vai continuar o evento

```
#include <stdio.h>
#include <windows.h>

HANDLE eof;

void fts(void)
{
    WaitForSingleObject(eof,INFINITE);
    printf("thread fts\n");
```

```
    ReleaseMutex(eof);
}

void hack(void)
{
    WaitForSingleObject(eof,INFINITE);
    printf("thread hack\n");
    ReleaseMutex(eof);
}

int main(void)
{
    HANDLE filho[10];
    DWORD pid[10];
    eof = CreateMutex(0,FALSE,0);
    filho[0] = CreateThread(0,0,(LPTHREAD_START_ROUTINE)fts,0,0,&pid[0]);
    filho[1] = CreateThread(0,0,(LPTHREAD_START_ROUTINE)hack,0,0,&pid[1]);
    WaitForMultipleObjects(2,filho,TRUE,INFINITE);
    return 0;
}
```

caso iniciamos o valor TRUE em CreateMutex podemos inicar com a própria API ReleaseMutex

```
#include <stdio.h>
#include <windows.h>

HANDLE eof;

void fts(void)
{
    WaitForSingleObject(eof,INFINITE);
    printf("thread fts\n");
    ReleaseMutex(eof);
}

void hack(void)
{
    WaitForSingleObject(eof,INFINITE);
    printf("thread hack\n");
    ReleaseMutex(eof);
}

int main(void)
{
    HANDLE filho[10];
    DWORD pid[10];
    eof = CreateMutex(0,TRUE,0);
    ReleaseMutex(eof);
    filho[0] = CreateThread(0,0,(LPTHREAD_START_ROUTINE)fts,0,0,&pid[0]);
    filho[1] = CreateThread(0,0,(LPTHREAD_START_ROUTINE)hack,0,0,&pid[1]);
    WaitForMultipleObjects(2,filho,TRUE,INFINITE);
```

```
    return 0;
}
```

além do mutex podemos usar semáforos que seria uma forma mais organizada de executa threads evitando deadlock, diferente do mutex que é a primeira thread a ser executada pausa as demais o semáforo segue uma ordem numérica que é incrementada ou decrementado a cada final assim executando a outra thread, todo semáforo tem um número inicial que começa no 0 e o numero final (quando chega no número final volta para o número inicial e começa a ordem de novo e apenas executa quantidade de threads equivalente ao número final), para criar um semáforo usamos a API CreateSemaphore passamos como argumento o atributo de segunraça (colocamos 0), o numero maximo do contado para iniciar, o número maximo de threads no semaforo, e o nome (isso opcional pode colocar 0 aqui também) e atribuimos para uma variavel do tipo HANDLE, dentro de cada thread usamos o WaitForSingleObject passamos como argumento para ele o handle do semaforo e a constante INFINITE, e no final dele usamos a API ReleaseSemaphore, os argumentos dela são, handle do semaforo, a quantidade de incremento, a última posição (isso é opcional podemos colocar um 0)

```
#include <stdio.h>
#include <windows.h>

HANDLE eof;

void fts(void)
{
    WaitForSingleObject(eof,INFINITE);
    printf("thread fts\n");
    ReleaseSemaphore(eof,1,0);
}

void hack(void)
{
    WaitForSingleObject(eof,INFINITE);
    printf("thread hack\n");
    ReleaseSemaphore(eof,1,0);
}

int main(void)
{
    HANDLE filho[10];
    DWORD pid[10];
    eof = CreateSemaphore(0,2,2,0);
    ReleaseMutex(eof);
    filho[0] = CreateThread(0,0,(LPTHREAD_START_ROUTINE)fts,0,0,&pid[0]);
    filho[1] = CreateThread(0,0,(LPTHREAD_START_ROUTINE)hack,0,0,&pid[1]);
    WaitForMultipleObjects(2,filho,TRUE,INFINITE);
    return 0;
}
```

9.1 – API Linux

O sistema linux tem mais APIs do que o sistema windows isso devido o kernel do linux ser todo

feito pela linguagem C e o linux é opensource (todos podem usar, modificar e distribuir da maneira que desejar) diferente do windows, as APIs do windows estão em DLLs no linux estao em biblioteca header ou lib do próprio sistema, no linux você pode manipular ate o kernel diferente do windows que você usa superficialmente, quando instala alguns programas no linux costuma instalar também novas bibliotecas no sistema operacional isso permite novas APIs para brincar, a primeira API que vamos ver para o sistema linux será o mkdir para criar diretorio nele passamos como argumento o nome do diretório que sera criado

```
#include <stdio.h>
#include <unistd.h>

int main(void)
{
    mkdir("hfts315");
    return 0;
}
```

para remover diretorio podemos usar a API rmdir passamos como argumento o diretorio

```
#include <stdio.h>
#include <unistd.h>

int main(void)
{
    rmdir("hfts315");
    return 0;
}
```

para criar um arquivo podemos usar a API creat passamos como argumento o nome do arquivo

```
#include <stdio.h>
#include <unistd.h>

int main(void)
{
    creat("fts.txt");
    return 0;
}
```

podemos usar API access para ver o tipo de acesso de um arquivo em uma condição if, os argumentos da API é o arquivo e tipo que poderia ser as constantes R_OK para leitura, W_OK para escrita, X_OK para execução, e F_OK para ver se o arquivo existe

```
#include <stdio.h>
#include <unistd.h>

int main(void)
{
    if(access("fts.txt",F_OK))
    {
        creat("fts.txt");
    }
```

```
    return 0;
}
```

para criar um hard link de um arquivo ou diretorio usamos a API link seguida do arquivo da onde sera criado e o nome do link

```
#include <stdio.h>
#include <unistd.h>

int main(void)
{
    link("fts.txt","link.txt");
    return 0;
}
```

para criar um link simbólico usamos a API symlink o uso dela é o mesmo que anterior, a diferença entre os dois que o hard link gera um arquivo apontando para o mesmo inode então mesmo se apagar um dos arquivo ainda fica la, já o link simbólico são parecido com os do windows ou seja se apagar o arquivo o link fica quebrado

```
#include <stdio.h>
#include <unistd.h>

int main(void)
{
    symlink("fts.txt","link.txt");
    return 0;
}
```

podemos usar API getcwd para ver o diretório atual, passamos como argumento uma array do tipo char e o tamanho dela

```
#include <stdio.h>
#include <unistd.h>

int main(void)
{
    char hack[300];
    getcwd(hack,300);
    printf("%s",hack);
    return 0;
}
```

podemos criar um subprocesso com a API fork, ela não tem argumentos apenas atribuir ela para uma variável do tipo pid_t

```
#include <stdio.h>
#include <unistd.h>

int main(void)
```

```
{
    pid_t fts = fork();
    return 0;
}
```

agora usamos um condição if e comparamos aquela variável com o valor 0, dentro da estrutura if colocamos as funções que serão executada pelo processo filho (a codinção if deve ser criada depois do fork porque se não tanto o processo pai quanto filha vão executar os códigos duas ou mais vezes)

```
#include <stdio.h>
#include <unistd.h>

int main(void)
{
    pid_t fts = fork();
    if(fts == 0)
    {
        printf("processo filho\n");
    }
   return 0;
}
```

para evitar que o processo filho execute fora do escopo usamos a função exit da biblioteca stdlib.h para finalizá-lo

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(void)
{
    pid_t fts = fork();
    if(fts == 0)
    {
        printf("processo filho\n");
        exit(0);
    }
    printf("processo pai\n");
    return 0;
}
```

podemos usar a API getpid para pegar o pid atual

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(void)
{
    pid_t fts = fork();
    if(fts == 0)
```

```
    {
        printf("processo filho: %d \n",getpid());
        exit(0);
    }
    printf("processo pai: %d \n",getpid());
    return 0;
}
```

Podemos usar API kill da biblioteca signal.h para finalizar determinado processo ou subprocesso, os argumentos dela são o pid seguido do signal code

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>

int main(void)
{
    pid_t fts = fork();
    if(fts == 0)
    {
        pid_t hack = getpid();
        printf("processo filho: %d \n",hack);
        kill(hack,9);
    }
    printf("processo pai: %d \n",getpid());
    return 0;
}
```

9.3 – API Socket (Windows e Linux)

O uso de sockets permite dois ou mais computadores se comuniquem tanto em rede local uma lan ou uma rede remoto a internet, o sistema linux foi criado para ser usado em rede deis do inicio já o sistema windows só depois do windows 95 então o linux quando se trata de rede é mil vezes melhor que o sistema windows e também mais rápido, a comunicação entre duas maquinas é usado diversos protocolos para interagir de maneira correta entre elas (e todos os demais protocolos segue um único padrao ou arquitetura que nada mais nada menos que um protocolo para os outros protocolo '-'), os sockets são um meio de comunicação entre uma conexão e existem pelo menos dois sockets em uma conexão, que seria o socket servidor e o socket cliente, ambos os sockets são configurado para ambos sockets são configurado para aceitar ou conectar a determinado IP ou porta (o ip nada mais é que um protocolo de endereçamento), o socket servidor vai ficar esperando uma conexão em determinado porta e socket cliente vai ser conectar ao IP da maquina naquela porta se a conexao for um sucesso ambas as maquina poderão se comunicar (independente se a maquina é o cliente ou servidor poderá enviar e receber dados, embora como esses dados é processado em ambos sockets isso depende da configuração e da programação do cliente e servidor), no windows é necessario iniciar um serviço chamado WSA para poder usar socket e também deve incluir a lib wsock32 para compilar e as biblioteca para usar socket na linguagem c/c++ é o “winsock.h” e “winsock2.h” (recomendo usar winsock2.h se seu windows for acima do 98), já no sistema linux não é necessario incluir nenhuma lib ou iniciar serviço porem o uso de biblioteca e bem extenso sendo elas “sys/socket.h”, “sys/types.h”, “netdb.h”, “arpa/inet.h”, “netinet/in.h” entre outras, para iniciar o serviço WSA no windows criamos uma variavel do tipo WSADATA e DWORD, na variavel

DWORD atribuimos uma API que seria o MAKEWORD da biblioteca windows.h nessa api passamos como argumento 2 e 0 (isso seria a versao), por fim usamos a API WSAStartup passamos como argumento para ela a variavel DWORD e o endereço de memoria da variavel WSADATA

```
#include <stdio.h>
#include <windows.h>
#include <winsock2.h>

int main(void)
{
    WSADATA servico;
    DWORD versao = MAKEWORD(2,0);
    WSAStartup(versao,&servico);
     return 0;
}
```

só lembrando para copilar no windows é necessário declarar a lib wsock32

```
C:\Users\fts315\Desktop> gcc fts.c -o hack -l wsock32
```

como dito antes a maioria das API de sockets do sistema windows esta na biblioteca winsock2.h no linux esta em varias bibliotecas para não ocorrer conflito e para o código ficar acessível para os dois sistemas operacionais recomendo usar a diretiva #ifdef

```
#include <stdio.h>
#ifdef WIN32
#include <windows.h>
#include <winsock2.h>
#else
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#endif

int main(void)
{
#ifdef WIN32
    WSADATA servico;
    DWORD versao;
    WSAStartup(versao,&servico);
#endif
    return 0;
}
```

agora criamos o socket com a API socket (essa API esta na biblioteca sys/socket.h no linux), passamos como argumento para ela a arquitetura ou family do socket que no caso usaremos AF_INET (existem outras porem essa é a mais usada atualmente), o tipo de socket (SOCK_STREAM usado para tcp, SOCK_DGRAM usado para udp e SOCK_RAW), e por fim o

protocolo que podemos usar algumas constantes só temos que declarar a biblioteca netdb.h no linux para isso (IPPROTO_TCP para protocolo tcp, IPPROTO_UDP para protocolo udp, IPPROTO_ICMP para protocolo icmp entre outros), também atribuímos a API para uma variavel do tipo int

```
#include <stdio.h>
#ifdef WIN32
#include <windows.h>
#include <winsock2.h>
#else
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#endif

int main(void)
{
    int sock;
#ifdef WIN32
    WSADATA servico;
    DWORD versao;
    WSAStartup(versao,&servico);
#endif
    sock = socket(AF_INET,SOCK_STREAM,IPPROTO_TCP);
#ifdef WIN32
#endif
    return 0;
}
```

podemos fechar um socket com a API closesocket do windows ou a API close no linux (no caso tambem coloquei uma diretiva #ifdef

```
#include <stdio.h>
#ifdef WIN32
#include <windows.h>
#include <winsock2.h>
#else
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#endif

int main(void)
{
    int sock;
```

```
#ifdef WIN32
    WSADATA servico;
    DWORD versao;
    WSAStartup(versao,&servico);
#endif
    sock = socket(AF_INET,SOCK_STREAM,IPPROTO_TCP);
#ifdef WIN32
    closesocket(sock);
#else
     close(sock);
#endif
    return 0;
}
```

agora vem a parte que configuramos o socket para ser servidor ou cliente, no caso do cliente e necessario usar a API connect e a struct sockaddr_in para configurar o ip e porta onde vai se conectar, no caso do servidor é necessario usar a API bind, listen e accept e duas estrutras sockaddr_in e mais duas variaveis do tipo int, primeiro vamos aprender a fazer o cliente devido ele ser o mais facil, para começar criamos a estrutura sockaddr_in onde vamos configurar o ip e a porta onde ele vai se conectar, no atributo do sockaddr_in colocamos sin_family e a mesma arquitetura do socket, em sin_addr.s_addr usamos a API inet_addr e passamos como argumento para ela o ip da maquina onde vai conectar, e por sin_port atribumos a API htons e passamos o numero da porta

```
#include <stdio.h>
#ifdef WIN32
#include <windows.h>
#include <winsock2.h>
#else
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#endif

int main(void)
{
    int sock;
    struct sockaddr_in remoto;
#ifdef WIN32
    WSADATA servico;
    DWORD versao;
    WSAStartup(versao,&servico);
#endif
    sock = socket(AF_INET,SOCK_STREAM,IPPROTO_TCP);
    remoto.sin_family = AF_INET;
    remoto.sin_addr.s_addr = inet_addr("127.0.0.1");
    remoto.sin_port = htons(80);
#ifdef WIN32
    closesocket(sock);
```

```
#else
     close(sock);
#endif
    return 0;
}
```

agora usamos a API connect para conectar, passamos como argumento para ela a variavel que tem o socket, o endereço de memoria da struct com o ip onde vai se conectar (tambem deve usar typecast para “struct sockaddr *”), e o tamanho da struct com o ip onde vai se conectar (usamos a função sizeof), pronto se der tudo certo ele se conectaria naquele ip e porta

```
#include <stdio.h>
#ifdef WIN32
#include <windows.h>
#include <winsock2.h>
#else
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#endif

int main(void)
{
    int sock;
    struct sockaddr_in remoto;
#ifdef WIN32
    WSADATA servico;
    DWORD versao;
    WSAStartup(versao,&servico);
#endif
    sock = socket(AF_INET,SOCK_STREAM,IPPROTO_TCP);
    remoto.sin_family = AF_INET;
    remoto.sin_addr.s_addr = inet_addr("127.0.0.1");
    remoto.sin_port = htons(80);
    connect(sock,(struct sockaddr *)&remoto,sizeof(remoto));
#ifdef WIN32
    closesocket(sock);
#else
     close(sock);
#endif
    return 0;
}
```

depois de conectado é só enviar ou receber os dados com a API send para enviar e a API recv para receber, os argumentos de ambas as APIs são o socket, a array do tipo char que vai ser enviado ou sera armazenada, o tamanho do dado enviado ou recebido, e a flag que podemos colocar 0, os dados enviado ou recebido depende do servidor e a quantidade de dados enviado ou recebido com uma ordem ou se o servidor e cliente fica preso em um loop infinito esperando novos mensagem para

enviar ou receber isso depende do servidor e utilidade dele, um exemplo os servidores http se receber uma requisição do tipo “get / http/\n\n” eles envia a pagina html

```
#include <stdio.h>
#include <string.h>
#ifdef WIN32
#include <windows.h>
#include <winsock2.h>
#else
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#endif

int main(void)
{
    int sock;
    struct sockaddr_in remoto;
    char fts[3];
    char dado[] = "get / http/\n\n";
#ifdef WIN32
    WSADATA servico;
    DWORD versao;
    WSAStartup(versao,&servico);
#endif
    sock = socket(AF_INET,SOCK_STREAM,IPPROTO_TCP);
    remoto.sin_family = AF_INET;
    remoto.sin_addr.s_addr = inet_addr("127.0.0.1");
    remoto.sin_port = htons(80);
    connect(sock,(struct sockaddr *)&remoto,sizeof(remoto));
    send(sock,dado,strlen(dado),0);
    while(recv(sock,fts,1,0))
    {
        printf("%c",fts[0]);
    }
#ifdef WIN32
    closesocket(sock);
#else
     close(sock);
#endif
    return 0;
}
```

para criar um socket servidor precisamos criar pelo menos duas struct sockaddr_in uma sera a estrutura da configuração do próprio servidor como a porta onde os clientes vão conectar, o outro sera as configuração do cliente que conectar, tambem e necessario criar duas novas variaveis do tipo int uma vai ser o novo socket e a outra o tamanho da estrutura, para começar vamos configurar a struct sockaddr_in do servidor que eu chamei de local, no sin_family tambem colocamos a arquitetura do socket, na porta fazemos a mesma coisa (é recomendado usar portas altas acima de

1024 devido as portas baixa ser portas padrao), no sin_addr.s_addr usamos INADDR_ANY

```
#include <stdio.h>
#ifdef WIN32
#include <windows.h>
#include <winsock2.h>
#else
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#endif

int main(void)
{
    int sock, sock2, tam;
    struct sockaddr_in local, remoto;
#ifdef WIN32
    WSADATA servico;
    DWORD versao;
    WSAStartup(versao,&servico);
#endif
    sock = socket(AF_INET,SOCK_STREAM,IPPROTO_TCP);
    local.sin_family = AF_INET;
    local.sin_addr.s_addr = INADDR_ANY;
    local.sin_port = htons(10315);
#ifdef WIN32
    closesocket(sock);
#else
     close(sock);
#endif
    return 0;
}
```

agora usamos o bind que seria equivalente ao connect os argumentos também são iguais, depois usamos listen isso permite colocar a quantidade maxima de conexao naquele socket, os argumento são o socket e quantidade

```
#include <stdio.h>
#ifdef WIN32
#include <windows.h>
#include <winsock2.h>
#else
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <arpa/inet.h>
#include <netinet/in.h>
```

```
#endif

int main(void)
{
    int sock, sock2, tam;
    struct sockaddr_in local, remoto;
#ifdef WIN32
    WSADATA servico;
    DWORD versao;
    WSAStartup(versao,&servico);
#endif
    sock = socket(AF_INET,SOCK_STREAM,IPPROTO_TCP);
    local.sin_family = AF_INET;
    local.sin_addr.s_addr = INADDR_ANY;
    local.sin_port = htons(10315);
    bind(sock,(struct sockaddr *)&local,sizeof(local));
    listen(sock,1);
#ifdef WIN32
    closesocket(sock);
#else
     close(sock);
#endif
    return 0;
}
```

para terminar usamos accept dentro de um laço while para esperar e permitir a conexão, os argumentos são o socket, o endereço de memoria da outra struct sockaddr_in, e o endereço de memoria da variavel int que vai armazenar o tamanho, e por fim atribuímos essa API para outra variável int que sera o outro socket

```
#include <stdio.h>
#ifdef WIN32
#include <windows.h>
#include <winsock2.h>
#else
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#endif

int main(void)
{
    int sock, sock2, tam;
    struct sockaddr_in local, remoto;
#ifdef WIN32
    WSADATA servico;
    DWORD versao;
    WSAStartup(versao,&servico);
```

```
#endif
    sock = socket(AF_INET,SOCK_STREAM,IPPROTO_TCP);
    local.sin_family = AF_INET;
    local.sin_addr.s_addr = INADDR_ANY;
    local.sin_port = htons(10315);
    bind(sock,(struct sockaddr *)&local,sizeof(local));
    listen(sock,1);
    while(sock2 = accept(sock,(struct sockaddr *)&remoto,&tam))
    {
    }
#ifdef WIN32
    closesocket(sock);
#else
    close(sock);
#endif
    return 0;
}
```

depois e só enviar ou receber os dados com o send e recv, no caso meu servidor vai enviar a hora e desconectar aquele o socket apenas o socket do cliente

```
#include <stdio.h>
#include <string.h>
#include <time.h>
#ifdef WIN32
#include <windows.h>
#include <winsock2.h>
#else
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#endif

int main(void)
{
    int sock, sock2, tam;
    struct sockaddr_in local, remoto;
    time_t tempo = time(NULL);
    struct tm *fts;
    char hora[100];
#ifdef WIN32
    WSADATA servico;
    DWORD versao;
    WSAStartup(versao,&servico);
#endif
    sock = socket(AF_INET,SOCK_STREAM,IPPROTO_TCP);
    local.sin_family = AF_INET;
    local.sin_addr.s_addr = INADDR_ANY;
```

```
    local.sin_port = htons(10315);
    bind(sock,(struct sockaddr *)&local,sizeof(local));
    listen(sock,1);
    while(sock2 = accept(sock,(struct sockaddr *)&remoto,&tam))
    {
        fts = localtime(&tempo);
        sprintf(hora,"%d:%d",fts->tm_hour,fts->tm_min);
        send(sock2,hora,strlen(hora),0);
#ifdef WIN32
        closesocket(sock2);
    }
    closesocket(sock);
#else
        close(sock2);
    }
    close(sock);
#endif
    return 0;
}
```

para pegar ver o ip do socket cliente basta usar a API inet_ntoa e passar o atributo sin_addr da estrutura sockaddr_in do socket cliente

```
#include <stdio.h>
#include <string.h>
#include <time.h>
#ifdef WIN32
#include <windows.h>
#include <winsock2.h>
#else
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#endif

int main(void)
{
    int sock, sock2, tam;
    struct sockaddr_in local, remoto;
    time_t tempo = time(NULL);
    struct tm *fts;
    char hora[100];
#ifdef WIN32
    WSADATA servico;
    DWORD versao;
    WSAStartup(versao,&servico);
#endif
    sock = socket(AF_INET,SOCK_STREAM,IPPROTO_TCP);
```

```
    local.sin_family = AF_INET;
    local.sin_addr.s_addr = INADDR_ANY;
    local.sin_port = htons(10315);
    bind(sock,(struct sockaddr *)&local,sizeof(local));
    listen(sock,1);
    while(sock2 = accept(sock,(struct sockaddr *)&remoto,&tam))
    {
        fts = localtime(&tempo);
        sprintf(hora,"%d:%d",fts->tm_hour,fts->tm_min);
        send(sock2,hora,strlen(hora),0);
        printf("ip do socket cliente: %s\n",inet_ntoa(remoto.sin_addr));
#ifdef WIN32
        closesocket(sock2);
    }
    closesocket(sock);
#else
        close(sock2);
    }
    close(sock);
#endif
    return 0;
}
```

nos codigos anteriores estamos usando apenas o IP para se conectar a um host ou domínio temos que converter esse dns para o ip, para fazer isso podemos usar gethostbyname e atribuir ela para um ponteiro de uma estrutura hostent

```
#include <stdio.h>
#ifdef WIN32
#include <windows.h>
#include <winsock2.h>
#else
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#endif

int main(void)
{
    int sock;
    struct hostent *host;
#ifdef WIN32
    WSADATA servico;
    DWORD versao;
    WSAStartup(versao,&servico);
#endif
    host = gethostbyname("localhost");
    printf("%s",inet_ntoa((struct in_addr)host->h_addr));
```

```
    return 0;
}
```

depois só converter o ip para string usando inet_ntoa, passamos como argumento asterisco e abre e fecha parênteses e la dentro o atributo h_addr da estrutra hostent (tambem usamos typecast “struct in_addr *”)

```
#include <stdio.h>
#ifdef WIN32
#include <windows.h>
#include <winsock2.h>
#else
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#endif

int main(void)
{
    int sock;
    struct hostent *host;
#ifdef WIN32
    WSADATA servico;
    DWORD versao;
    WSAStartup(versao,&servico);
#endif
    host = gethostbyname("localhost");
    printf("%s",inet_ntoa(*((struct in_addr *)host->h_addr)));
    return 0;
}
```

outro exemplo seria esse com socket cliente que conectava ao servidor http que tinha feito anteriormente

```
#include <stdio.h>
#include <string.h>
#ifdef WIN32
#include <windows.h>
#include <winsock2.h>
#else
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#endif
```

```
int main(void)
{
    int sock;
    struct sockaddr_in remoto;
    struct hostent *host;
    char fts[3];
    char dado[] = "get / http/\n\n";
#ifdef WIN32
    WSADATA servico;
    DWORD versao;
    WSAStartup(versao,&servico);
#endif
    sock = socket(AF_INET,SOCK_STREAM,IPPROTO_TCP);
    host = gethostbyname("localhost");
    remoto.sin_family = AF_INET;
    remoto.sin_addr.s_addr = inet_addr(inet_ntoa(*((struct in_addr *)host->h_addr)));
    remoto.sin_port = htons(80);
    connect(sock,(struct sockaddr *)&remoto,sizeof(remoto));
    send(sock,dado,strlen(dado),0);
    while(recv(sock,fts,1,0))
    {
        printf("%c",fts[0]);
    }
#ifdef WIN32
    closesocket(sock);
#else
     close(sock);
#endif
    return 0;
}
```

10.0 – CGI

Também é possível usar a linguagem C/C++ para uso web como cgi, para usar esses executáveis para uso web e necessário um servidor http configurado para uso de cgi, a saída de dados do executavel é redirecionado para o servidor web e a entrada de dados para o executavel são pelas variaveis de ambiente do executavel que no caso poderia ser por um formulario igual a linguagem PHP (metodo GET e POST), também é necessário na primeira saida de dados usar header para o servidor e de depois duas quebras de linhas seguida do codigo html (tambem é necessario trocar a extensao do executavel para .cgi)

```
#include <stdio.h>

int main(void)
{
    printf("Content-Type: text/html\n\n");
    printf("<html><head></head><body>pagina web</body></html>");
    return 0;
}
```

você também poderia usar css e javascript isso porque essas duas linguagens são executadas no

navegador e não no servidor (client-side)

```
#include <stdio.h>

int main(void)
{
    printf("Content-Type: text/html\n\n");
    printf("<html><head>");
    printf("<style type=\"text/css\">body{background-color: black; color: white}</style>");
    printf("<script type=\"text/javascript\">alert(\"seja bem vindo\");</script></head>");
    printf("<body>by hfts315</body></html>");
    return 0;
}
```

também podemos usar variáveis e funções da linguagem C/C++ para deixar mais dinâmico

```
#include <stdio.h>
#include <time.h>

int main(void)
{
    struct tm *fts;
    time_t tempo = time(NULL);
    fts = localtime(&tempo);
    printf("Content-Type: text/html\n\n");
    printf("<html><body>hora atual: %d:%d</body></html>",fts->tm_hour,fts->tm_min);
    return 0;
}
```

para pegar informação do servidor como IP ou ate a passagem de argumento como GET e POST podemos usar a função getenv da biblioteca stdlib.h, passamos como argumento o nome da variavel de ambiente (REMOTE_ADDR para o ip de quem conecto, HTTP_USER_AGENT para o agente, QUERY_STRING para pegar a passagem de formulario do metodo get não formatada, HTTP_COOKIE para os cookies entre outras)

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    char *ip, *agente, *argumento, *cookie;
    ip = getenv("REMOTE_ADDR");
    agente = getenv("HTTP_USER_AGENT");
    argumento = getenv("QUERY_STRING");
    cookie = getenv("HTTP_COOKIE");
    printf("Content-Type: text/html\n\n");
    printf("<html><head></head><body>");
    printf("seu IP: %s <br>",ip);
    printf("seu agente: %s <br>",agente);
    printf("passagem: %s<br>",argumento);
    printf("cookie: %s<br>",cookie);
```

```
    printf("</body></html>");
    return 0;
}
```

para adicionar um cookie podemos usar a header “Set-Cookie: “ seguido do cookie

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    char *cookie;
    cookie = getenv("HTTP_COOKIE");
    printf("Content-Type: text/html\n");
    printf("Set-Cookie: fts=315\n\n");
    printf("<html><head></head><body>");
    printf("cookies: %s <br>",cookie);
    printf("</body></html>");
    return 0;
}
```

para redirecionar usamos a header “Location: ” seguido da url

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    printf("Content-Type: text/html\n");
    printf("Location: http://endoffile.umforum.net\n\n");
    printf("<html><head></head><body></body></html>");
    return 0;
}
```

10.1 - ASM

A linguagem assembly é uma linguagem de baixo nível tambem é considerada uma das mais difícil de aprender comparada as outras linguagens de programação por causa da sintaxe dela, a sintaxe dela pode variar de compilador, sistema operacional ou ate processador, as sintaxes mais usadas dessa linguagem pelo menos eu acho é a INTEL e AT&T, a sintaxe INTEL são divididas por 4 partes que é o label (o label é opcional na maioria das vezes), a operação que será feita (ou opcode), onde será atribuído depois um virgula para separar e o valor que será atribuído, já na sintaxe AT&T é invertido a parte onde será atribuído e o valor

```
intel: operação onde , valor
at&t: opereção valor , onde
```

a linguagem assembly diferente das demais linguagens é apenas uma pequena conversao de codigos para linguagem da maquina (um exemplo é a operação nop que é nada mais que o codigo hexadecimal 0x90), por ela ser uma pequena conversão o executavel fica mil vezes menor do que um executavel de outra linguagem porém a source fica muito maior e tem muito mais chance de te

algum erro nela, o processador do computador tem uma memoria interna chamada registrador diferente da memoria RAM os registradores é muito mais rapido porém é limitado a quantidade e espaço, existem 4 registradores padrões que podemos usar para armazenar dados são eles “a”, “b”, “c” e “d” esse 4 registradores podem se dividi por 2 registradores sendo cada um 8 bytes, que seria o byte alto “ah”, “bh”, “ch”, “dh”, e o byte baixo “al”, “bl”, “cl”, “dl”, e cada registrador de 8bits pode armazenar exatamente um valor de 0 a 255, os dois registradores de 8bits juntos gera um registrador total de 16bits que tem um “x” no final subsistido o “h” e o “l” que seria o “ax”, “bx”, “cx”, “dx” e pode guardar um valor de 0 a 65535, um registrador “ah” com valor 03 e o registrador al com valor 15 seria equivalente ao registrado ax com o valor 0315 (pode ser que o valor formado não seja esse 0315 e sim 1503 dependendo do processador, tambem nao será o numero em decimal 0315 e sim o numero 783), e a maneira de como o processador vai ler o valor isso depende de como estamos usado o registrador se for “ax” ele ia ler 0315 ou “ah” ia ler apenas o 03, além desses registradores de 16bytes existem o de 32 e 64 bytes que são os mais usados atualmente nos sistemas operacionais, os registradores de 32 bytes tem a letra “e” na frente “eax”,”ebx”,”ecx”,”edx” e os 64bytes a letra “r” sendo eles “rax”,”rbx”,”rcx”,”rdx”

16 (8)	32	64
ax (ah + al)	eax	rax

eu não vou ensinar a programa em assembly puro aqui só vou ensinar apenas o basico de asm inline, ou seja usar asm dentro da própria linguagem C/C++, para começar usamos a função asm seguido do argumento que nada mais é que uma string com o codigo asm

```
#include <stdio.h>

int main(void)
{
    asm("");
    return 0;
}
```

a primeira operação ou opcode que vamos aprender será o “nop” citado anteriormente, esse opcode não faz nada quando o programa chega nessa parte ele simplesmente pula ela

```
#include <stdio.h>

int main(void)
{
    asm("nop");
    return 0;
}
```

podemos fazer outros opcodes separados por ponto e vírgula (ponto e virgula na linguagem assembly pura é o comentário porém no asm inline da linguagem c/c++ seria equivalente a uma quebra de linha)

```
#include <stdio.h>

int main(void)
{
    asm("nop ; nop");
```

```
    return 0;
}
```

também podemos usar \r\n para quebra de linha

```
#include <stdio.h>

int main(void)
{
    asm("nop \r\n nop");
    return 0;
}
```

por preferência para manter o codigo organizado faça dessa maneira

```
#include <stdio.h>

int main(void)
{
    asm("nop ;"
        "nop");
    return 0;
}
```

os label serve para localizar, chamar ou pular para determinada parte do codigo e eles sempre tem dois pontos no final, os label podem ser criado antes de um opcode dessa maneira

```
#include <stdio.h>

int main(void)
{
    asm("fts: nop");
    return 0;
}
```

ou sozinho desse jeito

```
#include <stdio.h>

int main(void)
{
    asm("fts:");
    return 0;
}
```

os registradores deve ser colocado um símbolo de porcentagem antes por exemplo o registrador %ax, e os numeros usado o símbolo de cifrao antes $315 (os numeros hexadecimal colocamos 0x ficando $0x13b), para atribuirmos um valor a um registrador usamos o opcode mov seguido do valor que será atribuído e o registrador separado por virgula (só lembrando que isso é a sintaxe at&t)

```
#include <stdio.h>

int main(void)
{
    asm("mov $315,%ax");
    return 0;
}
```

também podemos atribuir um valor de um registrador para outro

```
#include <stdio.h>

int main(void)
{
    asm("mov $315,%bx ;"
        "mov %bx,%ax");
    return 0;
}
```

podemos especificar a quantidade de bytes que será copiado com movb (8bytes), movw(16bytes), movl(32bytes) e movq(64bytes)

```
#include <stdio.h>

int main(void)
{
    asm("movb $03,%ah ;"
        "movb $15,%al ;"
        "movw %ax,%bx");
    return 0;
}
```

podemos somar um valor a um registrador com o opcode add nele colocamos o valor a ser atribuido e o registrador, o exemplo abaixo eu coloco o valor 215 no registrador ax depois eu adiciono mais 100 ao valor dando o valor 315

```
#include <stdio.h>

int main(void)
{
    asm("movw $215,%ax ;"
        "add $100,%ax");
    return 0;
}
```

também é possível subtrair com o opcode sub

```
#include <stdio.h>
```

```
int main(void)
{
    asm("movw $215,%ax ;"
        "sub $100,%ax");
    return 0;
}
```

para multiplicação usamos o opcode imul

```
#include <stdio.h>

int main(void)
{
    asm("movw $315,%ax ;"
        "imul $2,%ax");
    return 0;
}
```

podemos incrementar mais um ao número com o opcode inc seguido do registrador

```
#include <stdio.h>

int main(void)
{
    asm("inc %ax ;");
    return 0;
}
```

também podemos decrementar com o opcode dec

```
#include <stdio.h>

int main(void)
{
    asm("dec %ax ;");
    return 0;
}
```

podemos dá um pulo para determinado label com o opcode jmp depois digitamos o nome do label para onde vamos pular

```
#include <stdio.h>

int main(void)
{
    asm("jmp fts ;"
        "fts: ");
    return 0;
}
```

sempre tome cuidado com loop infinito

```
#include <stdio.h>

int main(void)
{
    asm("fts: ;"
        "jmp fts");
    return 0;
}
```

podemos pular para outro label que esteja dentro de outra função asm, no exemplo abaixo ele pula para outro label sem executar o printf

```
#include <stdio.h>

int main(void)
{
    asm("jmp fts ;");
    printf("o arduino e show *-*");
    asm("fts:");
    return 0;
}
```

para comparar dois registradores podemos usar o opcode cmp

```
#include <stdio.h>

int main(void)
{
    asm("movw $315, %ax ;"
        "movw $100, %bx ;"
        "cmp %ax,%bx ;");
    return 0;
}
```

para pular para determinado ponto se a comparação for igual podemos usar o opcode cmp e depois o opcode je que pula caso a comparação seja verdadeira (isso seria equivalente estrutura condicional if)

```
#include <stdio.h>

int main(void)
{
    asm("movw $315, %ax ;"
        "movw $315, %bx ;"
        "cmp %ax,%bx ;"
        "je fts");
    printf("se a comparacao for igual nao vai mostra esse texto '-' ");
    asm("fts:");
    return 0;
```

```
}
```

existe o jne que pula caso não se comparação não for igual (isso seria equivalente o if com um operador not)

```
#include <stdio.h>

int main(void)
{
    asm("movw $315, %ax ;"
        "movw $100, %bx ;"
        "cmp %ax,%bx ;"
        "jne fts");
    printf("se a comparacao for diferente nao mostra esse texto '-' ");
    asm("fts:");
    return 0;
}
```

existe o opcode jg pula caso o numero seja maior (só lembrando que isso at&t então o exemplo abaixo seria equivalente a isso bx > ax ou 100 > 315)

```
#include <stdio.h>

int main(void)
{
    asm("movw $315, %ax ;"
        "movw $100, %bx ;"
        "cmp %ax,%bx ;"
        "jg fts");
    printf("se o numero for maior nao mostra esse texto '-' ");
    asm("fts:");
    return 0;
}
```

com o opcode jl seria o inverso do anterior ele pula caso o numero seja menor

```
#include <stdio.h>

int main(void)
{
    asm("movw $315, %ax ;"
        "movw $100, %bx ;"
        "cmp %ax,%bx ;"
        "jl fts");
    printf("se o numero for menor nao mostra esse texto '-' ");
    asm("fts:");
    return 0;
}
```

para maior e igual podemos usar opcode jeg

```
#include <stdio.h>

int main(void)
{
    asm("movw $315, %ax ;"
        "movw $100, %bx ;"
        "cmp %ax,%bx ;"
        "jge fts");
    printf("se o numero for maior ou igual nao mostra esse texto '-' ");
    asm("fts:");
    return 0;
}
```

para o menor e igual seria o opcode jle

```
#include <stdio.h>

int main(void)
{
    asm("movw $315, %ax ;"
        "movw $100, %bx ;"
        "cmp %ax,%bx ;"
        "jle fts");
    printf("se o numero for menor ou igual nao mostra esse texto '-' ");
    asm("fts:");
    return 0;
}
```

já aprendemos a fazer pulo comparação e incremento então podemos fazer laços igual o while, um exemplo seria o registrador ax com o valor 0 e ficar em um loop incrementado ate chega a determinada condição para sessar (ou no caso pular)

```
#include <stdio.h>

int main(void)
{
    asm("movw $0, %ax ;"
        "eof: ;"
        "cmp $10,%ax ;"
        "jle fts ;"
        "inc %ax ;"
        "jmp eof ;"
        "fts: ");
    return 0;
}
```

podemos usar o opcode loop para pular para determinado ponto, para usar esse opcode temos que definir um valor para o registrador cx, criar um label e por fim usar o opcode loop seguido do label (esse opcode e bem parecido com o laço for)

```
#include <stdio.h>
```

```
int main(void)
{
    asm("movw $10, %cx ;"
        "fts: ;"
        "loop fts");
    return 0;
}
```

ate agora vimos como usar os opcodes e registradores agora vamos aprender como jogar os valores que estao em registrador na memória ou em variáveis, a primeira manipulação de memória que vamos ver sera a stack, a stack é uma memória que leva uma estrutura basica que seria a primeira coisa a entrar será a ultima a sair, ela funciona empilhando os dados um em cima dos outros por isso o nome dela é stack ou pilha, a stack tem várias utilidades uma delas é empilhar dados na ordem certa para ser executada por determinada função, outra é para você armazenar os dados dos registradores ou ate manipular eles mais tarde sem perde eles, os dois opcode mais usados para manipular o stack é o push para empilhar algo no stack e o pop para retirar também existe um registrado especifico que aponta para o topo do stack que é o “sp”, como dito antes para empilhar um valor usamos opcode push depois o registrador ou valor

```
#include <stdio.h>

int main(void)
{
    asm("movw $10, %ax ;"
        "push %ax");
    return 0;
}
```

e para retirar usamos o opcode pop

```
#include <stdio.h>

int main(void)
{
    asm("movw $10, %ax ;"
        "push %ax ;"
        "pop %ax");
    return 0;
}
```

no exemplo abaixo eu uso a pilha para trocar de ordem os registradores ax e bx

```
#include <stdio.h>

int main(void)
{
    asm("movw $10, %ax ;"
        "movw $315,%bx ;"
        "push %ax ;"
        "push %bx ;"
```

```
        "pop %ax ;"
        "pop %bx");
    return 0;
}
```

para a gente atribuir um valor de um registrador para uma variável usamos dois pontos depois da ultima string, usamos uma outra string que deve ter o símbolo de igual e a letra do registrador de onde será atribuído, logo em seguida um abre e fecha parênteses e a variável dentro, tambem todos os registradores deve ter dois símbolos de porcentagem

```
#include <stdio.h>

int main(void)
{
    int fts;
    asm("movw $315, %%ax" : "=a"(fts));
    printf("registrador ax: %d",fts);
    return 0;
}
```

para armazenar valor de dois ou mais registradores separamos por virgula

```
#include <stdio.h>

int main(void)
{
    int fts, eof;
    asm("movw $315, %%ax ;"
        "movw $100, %%bx" : "=a"(fts), "=b"(eof));
    printf("registrador ax: %d \n",fts);
    printf("registrador ax: %d",eof);
    return 0;
}
```

também podemos dá uma quebra de linha para ficar organizado

```
#include <stdio.h>

int main(void)
{
    int fts, eof;
    asm("movw $315, %%ax ;"
        "movw $100, %%bx"
        : "=a"(fts), "=b"(eof));
    printf("registrador ax: %d \n",fts);
    printf("registrador ax: %d",eof);
    return 0;
}
```

para entrar com os valores colocamos dois pontos depois daquele outro que seria a saida, uma string com a letra do registrador, abre e fecha parênteses com a variavel que tem o valor que sera

armazenado no registrador

```
#include <stdio.h>

int main(void)
{
    int fts = 315;
    asm("inc %%ax"  :  : "a"(fts));
    return 0;
}
```

podemos entrar e sair com valores, no exemplo abaixo eu entro com a variável fts o asm processa e incrementa mais um depois retorna para a mesma variável o novo valor

```
#include <stdio.h>

int main(void)
{
    int fts = 315;
    asm("inc %%ax"  : "=a"(fts) : "a"(fts));
    printf("%d",fts);
    return 0;
}
```

11.0 – Fim do arquivo (EOF)

Bom galera eu agradeço a todos que chegaram no final desse arquivo sem pular nenhum caracter XD, agradeço também os manos(as) que sempre estão ajudando no crescimento do fórum não só no fórum eof como qualquer outro fórum e blogs, tambem não posso deixar de agradecer o mmxm, f.ramon e susp3it0@virtual os atuais adms do fórum eof e outros membros ativos por sempre postar bons conteúdos e sources la, sobre esse ebook ter chegado ao final eu futuramente vou adicionar novos conteúdos que fico faltando e novas APIs como GTK e CURL entre outras, qualquer modificação que eu fizer postarei la no fórum ou no facebook informando, qualquer sugestão ou critica será bem-vinda, duvidas estarei sempre disposto a ajudar, para aqueles que não é membro do fórum recomendo muito se cadastrar tem muito conteudo de primeira sobre diversos assuntos sendo eles programação, pentest, sistema operacionais, games, animes, filmes entre outros, então é isso galera ate a próxima ^^

return 0;