UFABC

Linguagem de Programação C

Prof. Fabrício Olivetti de França

UFABC

Linguagem C

Linguagem C

- Imperativo e estruturado
- Pequeno conjunto de palavras-chaves, operadores, etc.
- Tipagem estática, porém fraca
- Permite criação de tipos de dados abstratos
- Acesso à memória do sistema
- Biblioteca padrão

UFABC

Linguagem C

Criado por Dennis Ritchie na Bell Labs com o objetivo de reescrever o sistema operacional Unix sem o uso de linguagem de máquina.

UFABC

Linguagem C

Tipos fundamentais:

int - números inteiros

char - caracteres

float e double - números em ponto flutuante

UFABC

Linguagem C

Tipos derivados que permitem criações de estruturas mais complexas:

- ponteiros
- vetores
- estruturas
- uniões

UFABC

Linguagem C

Fluxo de controle:

- Blocos de comandos: **{ }**
- Tomada de decisão: **if-else**
- Seleção: **switch**
- Laços com teste no topo: **while, for**
- Laços com teste no fundo: **do**
- Pulos para o início ou de saída dos laços: **continue, break**

UFABC

UFABC

Compilador GCC

primeiroPrograma.c

```
#include <stdio.h>

main(){
    printf(“Meu primeiro programa!\n”);
}
```

UFABC

Primeiro Programa

```
#include <stdio.h>

main(){
    printf("Meu primeiro programa!\n");
}
```

Inclui informações de comandos da biblioteca **stdio**

Define a função principal que será executada ao chamar o programa

UFABC

Primeiro Programa

```
#include <stdio.h>

main(){
    printf("Meu primeiro programa!\n");
}
```

Define o bloco de comandos pertencentes a função principal

UFABC

Primeiro Programa

```
#include <stdio.h>

main(){
    printf(“Meu primeiro programa!\n”);
}
```

Comando que imprime mensagem na tela.

UFABC

Primeiro Programa

```
#include <stdio.h>

main(){
    printf("Meu primeiro programa!\n");
}
```

Cada comando deve terminar com ";"

UFABC

Compilando e executando

No terminal digite:

gcc -o primeiroPrograma primeiroPrograma.c

Em seguida:

./primeiroPrograma

UFABC

GCC

GCC - Gnu Compiler Collection é um compilador de programas escritos nas linguagens C e C++.

Ele transforma o conjunto de instruções em um arquivo texto para a linguagem de máquina.

Além disso, ele se encarrega de otimizar seu programa.

UFABC

GCC

gcc -o nomedoexecutável nomedocodigo.c

-o: indica o nome do arquivo executável

UFABC

GCC

gcc -fverbose-asm -S -masm=intel -O arquivo.c

-fverbose-asm: indica que quero um código assembly legível

-S: compila para código assembly

-masm: indica a plataforma atual

UFABC

Mingw

Para quem for utilizar o windows, o gcc está implementado no software www.mingw.org

Ou a IDE http://www.codeblocks.org/ que já instala o Mingw (GCC) junto.

UFABC

UFABC

Processos

loopEterno.c

```
main( ){
    while( 1 );
}
```

Fica eternamente nesse laço de repetição!

UFABC

loopEterno.c

Enquanto o programa executa, entre em um outro terminal e digite: **top**

```
top - 09:10:12 up 19:17,  3 users,  load average: 1.38, 0.78, 0.69
Tasks: 215 total,   2 running, 213 sleeping,   0 stopped,   0 zombie
%Cpu(s): 34.1 us,  1.7 sy,  0.0 ni, 63.3 id,  0.9 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem:   8076740 total,  6823492 used,  1253248 free,     1436 buffers
KiB Swap: 31249404 total,        0 used, 31249404 free.  3550792 cached Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
14137 olivetti  20   0    4076    680    608 R 100.0  0.0   0:43.52 loopEterno
```

UFABC

Processos

Cada programa em execução é referenciado pelo Sistema Operacional através de um número de processo.

Através do número do processo o SO envia mensagens para o programa em execução.

UFABC

Processos

Vamos enviar uma mensagem para que o processo **morra**.

Digite:

kill numeroprocesso

UFABC

UFABC

Variáveis, Constantes e Memória

Variáveis

Similar ao Java e diferente do Python, na linguagem C devemos declarar explicitamente o tipo de cada variável:

```
int x;
char c;
float f;
double d;
```

UFABC

Variáveis

Na prática, a diferença entre os tipos é apenas a quantidade de bits que cada um ocupa:

int - tamanho do inteiro definido pelo processador

char - 1 byte (8 bits)

float - 4 bytes (32 bits)

double - 8 bytes (64 bits)

UFABC

Variáveis

Além disso temos qualificadores do int:

short int - 2 bytes (16 bits)

long int - 4 bytes (32 bits) ou 8 bytes (64 bits)

UFABC

Variáveis

E qualificadores de int e char:

unsigned - representa valores >= 0

signed - contém valores negativos também (padrão)

UFABC

Variáveis

Uma variável do tipo **unsigned char** pode representar valores entre **0** e $\mathbf{2^8}$, sendo 8 o tamanho do tipo em bits.

Uma variável do tipo **signed char** representa valores entre **-128** e **127**.

UFABC

Variáveis

E, finalmente:

long double - precisão estendida (dependente da plataforma)

UFABC

Variáveis

Vetores de tamanho pré-definidos podem ser criados da seguinte forma:

tipo nome[TAMANHO];

/* vetor de inteiros de tamanho 100 */

int v[100];

UFABC

Variáveis

Para verificar o tamanho de um tipo de variável podemos utilizar o comando **sizeof**:

printf(“%d\n, sizeof(int));

ou

int x;
printf(“%d\n, sizeof(x));

UFABC

tiposVars.c

UFABC

Constantes

Para definir uma constante, antes da função **main** utilizamos a diretiva **#define**:

```
#define PI 3.14
main(){
    int pipi;
    pipi = 2*PI;
}
```

UFABC

Constantes

Tudo definido pelo define será substituído no código durante a compilação:

```
#define PI 3.14
main(){
    float pipi;
    pipi = 2*PI;
}
```

```
main(){
    float pipi;
    pipi = 2*3.14;
}
```

UFABC

Constantes

Outra forma de definir constantes é a enumeração:

```
#enum {DOM, SEG, TER, QUA, QUI, SEX, SAB}
main(){
    int dia;
    dia = SEG;
}
```

```
main(){
    int dia;
    dia = 1;
}
```

UFABC

constantes.c

gcc -E constantes.c

UFABC

Operadores

Operadores

A Linguagem C possui cinco operadores aritméticos:
+, -, *, /, %.

Para o caso da divisão (/) entre inteiros, o quociente é truncado (corta-se qualquer valor após a vírgula).

UFABC

Operadores

Outros operadores:

++ e **--**: incrementa ou decrementa o valor numérico em 1

x op= y equivalente a x = x op y

Com op sendo qualquer operador aritmético.

UFABC

Operadores

A Linguagem C possui seis operadores relacionais:
<, >, <=, >=, ==, !=

E três operadores lógicos:
&& (e), || (ou), ! (não)

UFABC

Operadores

Nota: diferente de Java e Python, C não possui o tipo booleano, as expressões lógicas são avaliadas para **0** ou **1**.

Além disso, qualquer valor diferente de **zero** é considerado como **verdadeiro**.

UFABC

Operadores

Como em C todos os tipos são interpretados como sequências de bits, temos os operadores:

& - realiza a operação E entre dois números

| - OU

^ - OU EXCLUSIVO

<< - Desloca os bits n casas para a esquerda

>> - Desloca para a direita

~ - complemento do número

UFABC

Códigos

opsAritmeticos.c

opsLogicos.c

opsBinarios.c

UFABC

UFABC

Blocos de Instruções

Blocos de Instruções em C

Na linguagem C, os blocos de instruções são definidos por **{ }**.

```
{
    a = a + b;
    b = a - b;
    a = a - b;
}
```

UFABC

Blocos de Instruções em C

Esses blocos definem um objetivo específico do algoritmo!

```
/* Troca os valores de a e b */
{
    a = a + b;
    b = a - b;
    a = a - b;
}
```

UFABC

Blocos de Instruções em C

Os comentários são delimitados por **/*** e ***/** e devem ficar em uma linha própria, antes do código a ser comentado.

Dica: seu código deve ser possível de ser entendido com um comentário por bloco. Caso sinta a necessidade de acrescentar comentários, repense o nome das variáveis e a sequência de instruções.

UFABC

Blocos Condicionais

Em C temos dois blocos condicionais: **if-else** e **switch**.

If-else, já conhecido de PI faz um desvio de acordo com o resultado de uma expressão lógica.

switch define uma escolha a ser feita de acordo com o valor de uma variável.

UFABC

Blocos Condicionais

```
if( condicao ){
    instruções;
}
```

UFABC

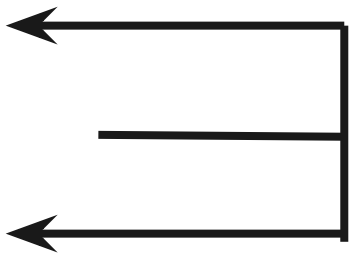
Blocos Condicionais

```
if( condicao ){
    instruções;
}else{
    outrasinstruções;
}
```

UFABC

Blocos Condicionais

```
if( condicao1 ){
    instruções;
}else if( condicao2 ){
    outrasinstruções;
}else{
    maisinstruções;
}
```



As instruções pertencentes ao bloco **if** estão delimitadas pelo bloco **{ }**.

UFABC

Controle Condicional

```
if( x>10 ){
   printf(“x é muito grande!\n”);
}else if( x<10 ){
   printf(“x é muito pequeno!\n”);
}else{
   printf(“x tem o tamanho certo!\n”);
}
```

UFABC

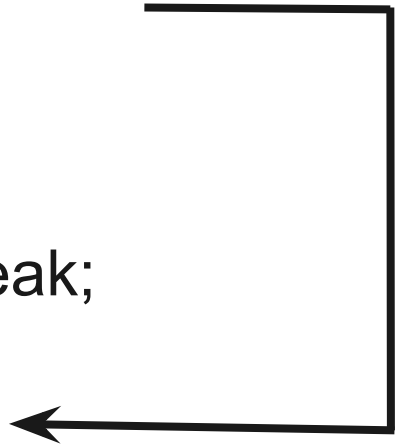
Blocos Condicionais

```
switch( variável ){
    case valor1:
            Instruções; break;
    case valor2:
            instruções;
    default:
            instruções;
}
```

UFABC

Blocos Condicionais

```
switch( variável ){
    case valor1:
            Instruções; break;
    case valor2:
            instruções;
    default:
            instruções;
}
```

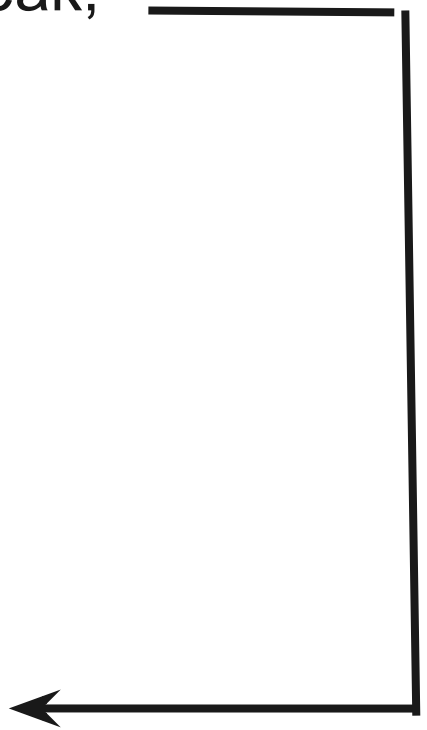


Faz um **goto** para o label contendo o valor da variável.

UFABC

Blocos Condicionais

```
switch( variável ){
    case valor1:
            Instruções; break;
    case valor2:
            instruções;
    default:
            instruções;
}
```



O **break** indica para ir para o final do bloco.

UFABC

Blocos Condicionais

```
switch( variável ){
    case valor1:
        Instruções; break;
    case valor2:
        instruções;
    default:
        instruções;
}
```

A ausência de **break** faz com que o programa continue executando até o final do bloco.

UFABC

Controle Condicional

```
switch( x ){
    10:
            printf(“x tem o tamanho certo!\n”);
            break;
    default:
            printf(“x tem o tamanho errado!\n”);
            break;
}
```

UFABC

Blocos de Repetição

A linguagem C possui três blocos de repetição:

while: repete o bloco enquanto certa condição for atendida.

do..while: igual ao while, porém garante a execução do bloco pelo menos uma vez.

for: delimita o valor de uma ou mais variáveis entre um limite inferior e superior.

UFABC

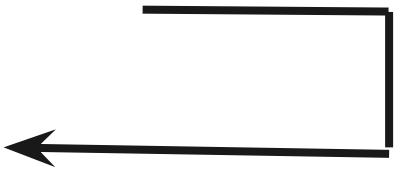
Blocos de Repetição

```
while( condicao ){
    instruções;
}
```

UFABC

Blocos de Repetição

```
while( condicao ){
    instruções;
}
```



Enquanto a condição for verdadeira, executa a sequência de instruções.

UFABC

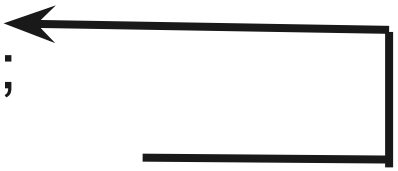
Blocos de Repetição

```
do{
    instruções;
}while( condição );
```

UFABC

Blocos de Repetição

```
do{
    instruções;
}while( condição );
```



Executa a seq. de instruções e repete tal seq. enquanto a condição for verdadeira.

UFABC

Controle Repetição

```
while( x != 10 ){
    printf(“entre com o valor de x: “);
    scanf(“%d”, &x);
}
do{
    printf(“entre com o valor de x: “);
    scanf(“%d”, &x);
}while( x != 10 );
```

UFABC

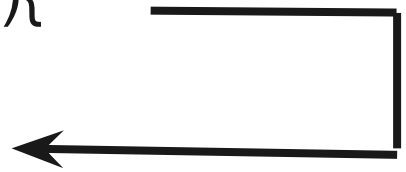
Blocos de Repetição

```
for( var = valor; condição; var = var + constante){
        instruções;
}
```

UFABC

Blocos de Repetição

```
for( var = valor; condição; var = var + constante){
        instruções;
}
```

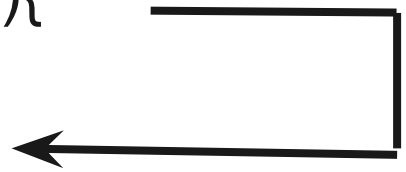


Enquanto a condição for verdadeira, executa a sequência de instruções.

UFABC

Blocos de Repetição

```
for( var = valor; condição; var = var + constante){
        instruções;
}
```

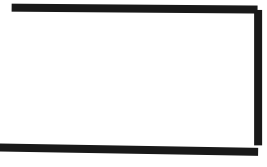


Inicialmente a variável **var** assumirá o conteúdo **valor**.

UFABC

Blocos de Repetição

```
for( var = valor; condição; var = var + constante){
        instruções;
}
```



A cada repetição a variável **var** será alterada pela expressão final.

UFABC

Controle Repetição

```
for( x = 0; x!=10; x=x+1 ){
    if( x != 10 ){
        printf(“x tem o valor errado!\n”);
    }
}
if( x == 10 ){
    printf(“x tem o valor correto!\n”);
}
```

UFABC

Códigos

condicional.c

selecao.c

repeticaoFor.c

repeticaoWhile.c

UFABC

Entrada e Saída (básico)

printf(string, opções);

string = string com o conteúdo e a formatação
opções = valores de diversos tipos a serem impressos

UFABC

Entrada e Saída (básico)

```
printf("Ola mundo!\n");
```

\n - pule uma linha
\t - tabulação

UFABC

Entrada e Saída (básico)

```
int x = 10;
printf("O número é: %d\n", x);
```

%d - imprima um inteiro

UFABC

Entrada e Saída (básico)

%d - int
%ld - long int
%u - unsigned int
%c - char
%f - float
%lf - double
%s - string

UFABC

Entrada e Saída (básico)

%.xf - float com x casas decimais

UFABC

Entrada e Saída (básico)

scanf(string, variáveis);

```
int x;
scanf("Entre com um inteiro: %d", &x);
```

isso é importante!!!

UFABC

Entrada e Saída (básico)

Vocês só tem permissão para utilizar printf e scanf dentro da função main()!!!!

UFABC