Aula 01 Revisão Conceitos

Algoritmos

- Sequência de passos para a realização de uma tarefa
- Linguagem de alto nível (exemplo: receita de bolo)
- Como transformar um algoritmo em linguagem que o computador entenda?

Primeiro Programa

```
#include main()
{
printf(“Boa Tarde!\n");
}
```

O que são erros de Compilação?

```
#include <stdio.h>
int main()
{
printf(“Boa Tarde!\n");
return 0;
}
```

O que são erros de Execução

```
#include <stdio.h>

int main()
{
printf(“Boa Tarde!   *&^%&&&&* \n");
return 0;
}
```

- Resultado : Boa Tarde!   *&^%&&&&*

O que é um depurador

- Ferramenta que executa um programa passo a passo
- Ajuda a encontrar erros de execução (bugs)

Um exemplo simples com mais informações

```
#include <stdio.h>
int main()
{
  int x, y;
  printf("x: ");
  scanf("%d", &x);
  printf("y: ");
  scanf("%d", &y);
  if (x > y) printf ("O maior número e´ x = %d\n", x);
        else printf ("O maior número e´ y = %d\n", y);
}
```

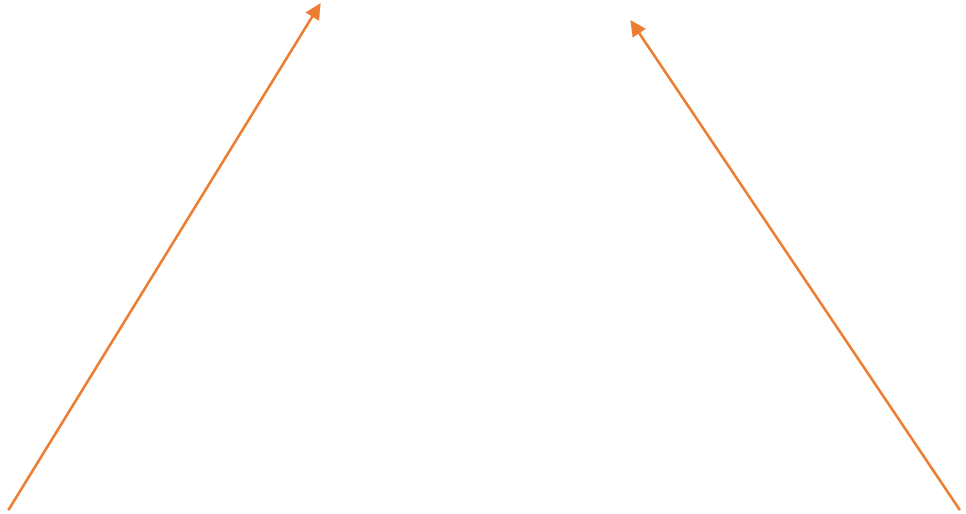
Variáveis

- Variáveis são locais onde armazenamos valores na memória.

- Toda variável é caracterizada por um nome, que a identifica em um programa, e por um tipo, que determina o que pode ser armazenado naquela variável.

Declarando uma variável

```
int   soma;
```



Tipo da variável

Nome da variável

Variáveis Inteiras

- Variáveis utilizadas para armazenar valores inteiros, em formato binário. Ex: $13_{10} = 1101_{2}$

- int: Inteiro cujo comprimento depende do computador. E o inteiro mais utilizado. Em computadores Pentium, ocupa 32 bits e pode armazenar valores de

  -2.147.483.648 a 2.147.483.647.

Variáveis Inteiras

- unsigned int: Inteiro cujo comprimento depende do computador e que armazena somente valores positivos. Em computadores Pentium, ocupa 32 bits e pode armazenar valores de 0 a 4.294.967.295.
- long int: Inteiro que ocupa 32 bits e pode armazenar valores de -2.147.483.648 a 2.147.483.647, independente do computador.
- unsigned long int: Inteiro que ocupa 32 bits e pode armazenar valores de 0 a 4.294.967.295, independente do computador.

Variáveis Inteiras

- short int: Inteiro que ocupa 16 bits e pode armazenar valores de -32.768 a 32.767.

- unsigned short int: Inteiro que ocupa 16 bits e pode armazenar valores de 0 a 65.535.

Variáveis de tipo caracter

- Variáveis utilizadas para armazenar letras e outros símbolos existentes em textos.
- São, na verdade, variáveis inteiras que armazenam um número associado ao símbolo. A principal tabela de símbolos utilizada pelos computadores é a tabela ASCII (American Standard Code for Information Interchang), mas existem outras (EBCDIC, Unicode, etc ..).

Variáveis de tipo caracter

- char: Armazena um símbolo (no caso, o inteiro correspondente). Seu valor pode ir de -128 a 127.

- unsigned char: Armazena um símbolo (no caso, o inteiro correspondente). Seu valor pode ir de 0 a 255.

Variáveis de tipo ponto flutuante

- Armazenam valores reais, da seguinte forma

$(-1)^{sinal} \cdot mantissa \cdot 2^{expoente}$ Ex: $0.5 = (-1)^{0} \cdot 1 \cdot 2^{-1}$

- Para o programador, funciona como se ele armazenasse números na forma decimal.

- Possui problemas de precisão (arredondamento).

Variáveis de tipo ponto flutuante

float: Utiliza 32 bits, sendo 1 para o sinal, 8 para o expoente e 23 para a mantissa.

Pode armazenar valores de $(+ / -)10^{-38}$ a $(+ / -)10^{38}$

double: Utiliza 64 bits, sendo 1 para o sinal, 11 para o expoente e 52 para a mantissa.

Pode armazenar valores de $(+ / -)10^{-308}$ a $(+ / -)10^{308}$

Obtendo o tamanho de um tipo

O comando sizeof(tipo) retorna o tamanho, em bytes, de um determinado tipo.

(Um byte corresponde a 8 bits).

Ex: printf ("%d", sizeof(int));

escreve 4 na tela (Pentium).

Regras para nomes de variáveis em C

Deve começar com uma letra (maíuscula ou minúscula) ou subcrito( ). Nunca pode começar com um número.

Pode conter letras maiúscula, minúscula, números e subscrito.

Não pode-se utilizar { (+ - * / \ ; . , ? como parte do nome de uma variável.

Regras para nomes de variáveis em C

As seguintes palavras já tem um significado na linguagem C e por esse motivo não podem ser utilizadas como nome de variáveis:

auto	double	int	struct
break	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

Constantes

Constantes são valores previamente determinados e que, por algum motivo, devem aparecer dentro de um programa (veremos adiante onde elas podem ser usadas).

Assim como as variáveis, as constantes também possuem um tipo. Os tipos permitidos são exatamente os mesmos das variáveis, mais o tipo string, que corresponde a uma sequëncia de caracteres.

Exemplos de constantes: 85, 0.10, ’c’, ”Meu primeiro programa”.

Constantes inteiras

Um número na forma decimal, como escrito normalmente Ex: 10, 145, 1000000

Um número na forma hexadecimal (base 16), precedido de 0x Ex: 0xA ($0xA_{16} = 10_{2}$), 0x100 ($0\ x100_{16} = 256_{2}$)

Um número na forma octal (base 8), precedido de 0 Ex: 010 ($0\ x10_{8} = 8_{2}$)

Constantes do tipo de ponto flutuante

Um número decimal. Para a linguagem C, um número só pode ser considerado um número decimal se tiver uma parte ”não inteira”, mesmo que essa parte não inteira tenha valor zero.

Utilizamos o ponto para separarmos a parte inteira da parte ”não inteira”

Ex: 10.0, 5.2, 3569.22565845

Constantes do tipo ponto flutuante

Um número inteiro ou decimal seguido da letra e um expoente. Um número escrito dessa forma deve ser interpretado como:

$$numero \cdot 10^{expoente}$$

Ex: 2e2 ($2e2 = 2 \cdot 10^{2} = 200.0$)

Constantes do tipo caracter

Uma constante do tipo caracter é sempre representado por uma letra entre aspas simples.

Ex: ’A’

Toda constante do tipo caracter pode ser usada como uma constante do tipo inteiro. Nesse caso, o valor atribuído será o valor daquela letra na tabela ASCII.

Constantes do tipo string

Uma constante do tipo é um texto entre aspas duplas

Ex: ”Meu primeiro programa”

Escrevendo o conteúdo de uma variável na tela

Podemos imprimir, além de texto puro, o conteúdo de uma variável utilizando o comando printf.

Para isso, utilizamos um símbolo no texto para representar que aquele trecho deve ser substituído por uma variável e, no final, passamos uma lista de variáveis ou constantes, separadas por vírgula.

Escrevendo o conteúdo de uma variável na tela

Ex: a=10;

printf (”A variável %s contém o valor %d”,”a”, a);

imprime - A variável a contém o valor 10

Nesse caso, %s deve ser substituído por uma variável ou constante do tipo string enquanto %d deve ser substituído por uma variável ou constante do tipo inteiro.

Formatos inteiros

%d — Escreve um inteiro na tela sem formatação.

Ex: printf ("%d", 10)

imprime: 10

Formatos inteiros

%< numero > d — Escreve um inteiro na tela, preenchendo com espaços a esquerda para que ele ocupe pelo menos < numero > casas na tela.

Ex: printf ("%4d", 10)

imprime: < espaco >< espaco >10

Formatos inteiros

%0< numero > d — Escreve um inteiro na tela, preenchendo com zeros a esquerda para que ele ocupe pelo menos comprimento < numero >.

Ex: printf ("%04d", 10)

Imprime: 0010

Formatos inteiros

%< numero 1 >.0< numero 2 > d — Escreve um inteiro na tela, preenchendo com espa¸cos a esquerda para que ele ocupe pelo menos < numero 1 > casas na tela e com zeros para que ele possua pelo menos comprimento < numero 2 >.

Ex: printf ("%6.04d", 10)

Imprime: < espaco >< espaco >0010

Formatos inteiros

A letra d pode ser substituída pelas letras u e l, ou as duas, quando desejamos escrever variáveis do tipo unsigned ou long, respectivamente.

Ex: printf ("%d", 4000000000)

Imprime: -294967296 na tela, enquanto que

printf ("%u", 4000000000)

Imprime: 4000000000.

Formatos ponto flutuante

%f — Escreve um ponto flutuante na tela, sem formatação

Ex: printf ("%f", 10.0)

Imprime: 10.000000

Formatos ponto flutuante

%e — Escreve um ponto flutuante na tela, em notação científica

Ex: printf ("%e", 10.02545)

imprime 1.002545e+01

Formatos ponto flutuante

%< tamanho >.< decimais > f — Escreve um ponto flutuante na tela, com tamanho < tamanho > e < decimais > casas decimais.

Lembre-se que o ponto, utilizado para separar a parte inteira da decimal, também conta no tamanho.

Ex: printf ("%6.2f", 10.0)

Imprime: < espaco >10.00

Formatos ponto flutuante

A letra f pode ser substituida pelas letras lf, para escrever um double ao invés de um float

Ex: printf ("%6.2lf", 10.0)

Imprime: < espaco >10.00

Formato caracter

%c — Escreve uma letra.

Ex: printf ("%c", ’A’)

Imprime: A

Note que printf ("%c", 65)

Também imprime a letra: A.

Formato string

%s — Escreve uma string

Ex: printf ("%s", "Meu primeiro programa");

Imprime: Meu primeiro programa

A função scanf

realiza a leitura de um texto a partir do teclado

parâmetros: – uma string, indicando os tipos das variáveis que serão lidas e o formato dessa leitura. – uma lista de variáveis

aguarda que o usuário digite um valor e atribui o valor digitado à variável

A função scanf

```
#include <stdio.h>
int main()
{
        int n;
        printf("Digite um número: ");
        scanf("%d",&n);
        printf("O valor digitado foi %d\n",n);
return 0;
 }
```

A função scanf

O programa acima é composto de quatro passos:

1. Cria uma variável n;
2. Escreve na tela Digite um número:
3. Lê o valor do número digitado
4. Imprime o valor do número digitado

A função scanf
Leitura de várias variáveis

```
#include <stdio.h>
int main()
{
  int m, n, o;
  printf("Digite três números: ");
  scanf("%d %d %d",&m, &n, &o);
 printf("O valores digitados são\ %d %d %d\n", m, n, o);
return 0;
}
```

O endereço de uma variável

Toda variável tem um endereço de memória associado a ela.

Esse endereço é o local onde essa variável é armazenada no sistema (como se fosse o endereço de uma casa, o local onde as pessoas “são armazenadas”).

O endereço de uma variável

Normalmente, o endereço das variáveis não são conhecidos quando o programa é escrito.

O endereço de uma variável é dependente do sistema computacional e também da implementação do compilador C que está sendo usado.

O endereço de uma mesma variável pode mudar entre diferentes execuções de um mesmo programa C usando uma mesma máquina.

O operador “address-of” & de C

o operador & retorna o endereço de uma determinada variável

Ex: printf ("%d", &valor);

o endereço da variável valor.

O operador “address-of” & de C

É necessário usar o operador & no comando scanf, pois esse operador indica que o valor digitado deve ser colocado no endereço referente a uma variável.

Esquecer de colocar o & comercial é um erro muito comum que pode ocasionar erros de execução.

O operador “address-of” & de C

O programa abaixo imprime o valor e o endereço da variável:

```
#include <stdio.h>
int  main (void)
{ int n = 8;
 printf("valor %d, endereço 0x%x\n",n,&n);
return 0;
 }
```

Formatos de leitura de variável

Os formatos de leitura são muito semelhantes aos formatos de escrita utilizados pelo printf.

A seguir tem-se alguns formatos possíveis de leitura Código Função

%c	Lê um único caracter
%s	Lê uma série de caracteres
%d	Lê um número decimal
%u	Lê um número decimal sem sinal
%l	Lê um inteiro longo
%f	Lê um número em ponto flutuante
%lf	Lê um double

Atribuição

Atribuir um valor de uma expressão a uma variável significa calcular o valor daquela expressão e copiar aquele valor para uma determinada variável.

Atribuição

No exemplo abaixo, a variável soma recebe o valor calculado da expressão a + b

```
soma = a + b;
```

Variável

Expressão

Atribuição

O operador de atribuição é o sinal de igual (=)

À esquerda do operador de atribuição deve existir somente o nome de uma variável.

=

À direita, deve haver uma expressão cujo valor será calculado e armazenado na variável

Expressão

Uma expressão é um conjunto de operações aritméticas, lógicas ou relacionais utilizados para fazer “cálculos” sobre os valores das variáveis.

Ex: a + b

Calcula a soma de a e b

Expressões

Uma constante é uma expressão e como tal, pode ser atribuída a uma variável (ou em qualquer outro lugar onde uma expressão seja necessária)

Ex: a = 10;

Uma variável também é uma expressão

Ex: a = b;

Expressões

< expressao > + < expressao >: Calcula a soma de duas expressões. Ex: a = a + b;

< expressao > - < expressao >: Calcula a subtração de duas expressões. Ex: a = a - b;

< expressao > * < expressao >: Calcula o produto de duas expressões. Ex: a = a * b;

Expressões

< expressao > / < expressao >: Calcula o quociente de duas expressões.

Ex: a = a / b;

< expressao > % < expressao >: Calcula o resto da divisão (inteira) de duas expressões.

Ex: a = a % b;

- < expressao >: Calcula o oposto da expressão.

Ex: a = -b;

Precedência

Exercício:

Qual o valor da expressão 5 + 10 % 3?

E da expressão 5 * 10 % 3?

Precedência

Precedência é a ordem na qual os operadores serão calculados quando o programa for executado.

Em C, os operadores são calculados na seguinte ordem:

- ++, -- , na ordem em que aparecerem na expressão
- * e /, na ordem em que aparecerem na expressão.
- %
- + e -, na ordem em que aparecerem na expressão.

Alterando a precedência

(< expressao >) também é uma expressão, que calcula o resultado da expressão dentro dela para só então permitir que as outras expressões executem. Deve ser utilizada quando a ordem da precedência não atende aos requisitos do programa.

Ex: 5 + 10 % 3 retorna 6, enquanto (5 + 10) % 3 retorna 0

Você pode usar quantos parênteses desejar dentro de uma expressão, contanto que utilize o mesmo número de parênteses para abrir e fechar expressões.

Incremento(++) e Decremento(––)

Operadores de incremento e decremento tem duas funções: servem como uma expressão e incrementam ou decrementam o valor da variável ao qual estão associados em uma unidade.

Ex: c++ — incrementa o valor da variável c em uma unidade

Dependendo da posição do operado r de incremento e decremento, uma função é executada antes da outra.

Incremento(++) e Decremento(––)

Operador à esquerda da variável: Primeiro a variável é incrementada, depois a expressão retorna o valor da expressão.

Ex: #include <stdio.h>

main ()

{ int a = 10;

printf ("%d", ++a);

}

Imprime 11

Incremento(++) e Decremento(––)

operador à direita da variável: Primeiro a expressão retorna o valor da variável, e depois a variável é incrementada.

Ex: #include <stdio.h>

int main (void)

{ int a = 10;

printf ("%d", a++);

}

Imprime 10

Incremento(++) e Decremento(––)

Em uma expressão, os operadores de incremento e decremento são sempre calculados primeiro (tem maior precedência)

```
#include <stdio.h>
int main (void)
{ int a = 10;
  printf ("%d", a * ++a);
}
```

Imprime 121

Atribuições simplificadas

Uma expressão da forma

a = a + b

onde ocorre uma atribuição a uma das variáveis da expressão pode ser simplificada como

a += b

Atribuições simplificadas

Comando	Exemplo	Corresponde
+=	a += b	a = a + b;
-=	a -= b	a = a - b;
*=	a *= b;	a = a * b;
/=	a /= b;	a = a / b;
%=	a %= b;	a = a % b;