1. Tipos de Dados

1.1 – Definição de Dados

São informações, que podem ser nºs ou caracteres, com os quais o programa opera.

1.2 - Definição de Variáveis

Representam localizações de memória onde são armazenados valores que podem ser modificados pelo programa.

1.3 - Variáveis em C

1.3.1. – Nome das variáveis

Pode conter letras, números e caracter de sublinhado. Porém :

- 1º caracter **NÃO** pode ser número
  ex.: br_01
  _br01
  01_br (NÃO é permitido)

- letras maiúsculas é diferente de letras minúsculas (**convenção : minúsculas**)
  ex.: **A1** é diferente de **a1**

- não podemos usar palavras reservadas
  ex.: int, float, if , else, etc...

1.3.2 - Tipos Básicos

char - apenas 1 caracter alfanumérico (geralmente ocupa 1 byte)
int - nºs inteiros ex.: 7 (geralmente ocupa 2 bytes)
float - nºs fracionários com precisão simples ex.: 7.5 (geralmente ocupa 4 bytes)
double - nºs fracionários com precisão dupla (geralmente ocupa 8 bytes)
void - para indicar que não retorna nada

1.3.3 – Declaração de Variáveis

sintaxe:
tipo nome_variável ;

ex.:

```
int     x , y ;
float   f ;
```

1.3.4 – Inicialização de Variáveis

sintaxe:
nome_variável = valor ;

ex.:

```
x = y = 10 ;
f = 3.5 ;
```

ou ainda, podemos fazer 1.3.3 e 1.3.4 juntos, como a seguir:

1.3.5 – Declaração e Inicialização de Variáveis

sintaxe:
tipo nome_variável = valor ;

ex.:

```
int     x =10 , y =10 ;
float   f = 3.5 ;
```

1.4 - Definição de Constantes

Representam localizações na memória, de dados que **não** podem ser alterados durante a execução do programa.

1.5 – Constantes em C

1.5.1. – Nome das variáveis

- por convenção sempre MAIÚSCULAS

1.5.2 - Tipos Básicos

char - caracteres alfanuméricos
int - nºs inteiros
float - nºs fracionários com precisão simples
double - nºs fracionários com precisão dupla
void - para indicar que não retorna nada

1.5.3 – Declaração e Inicialização de Constantes

sintaxe:
#define NOME_CONSTANTE valor

ex.:

```
#define   PI    3.14159
#define   MAX   500
```

2. Operadores

2.1 – Operador de Atribuição (=)

sintaxe:
nome_variável = expressão ;
ex.:

```
y = 2 ;                 /*  atribui o valor 2 a y  */x = 4 * y + 3 ;          /*  atribui  o
valor da expressão a x  */
```

CUIDADO: **Conversão de Tipos em Atribuições**
Regra: O valor do lado direito de uma atribuição é convertido no **tipo** do lado esquerdo.

ex.:

```
int     x ;
float   f ;

x = f = 3.5 ;   /*  resulta em f = 3.5  e  x = 3  */
f = x = 3.5 ;   /*  resulta em f = 3.0  e  x = 3  */
```

2.2 - Operadores Aritméticos

2.2.1 - Operadores Unários - atuam sobre apenas um operando

-	**(menos unário)**	multiplica o operando por (-1)
++	**(incremento)**	incrementa o operando em uma unidade
--	**(decremento)**	decrementa o operando em uma unidade

ex.: x = 2 ; e y = 4*x + 3 ;

++x incrementa o valor de x antes de usá-lo (portanto usaremos x = 3 e teremos y = 15)**x++** incrementa o valor de x depois de usá-1o (portanto usaremos x = 2 e teremos y = 11)**--x** decrementa o valor de x antes de usá-lo (portanto usaremos x = 1 e teremos y = 7)**x--** decrementa o valor de .x depois de usá-lo (portanto usaremos x = 2 e teremos y = 11)

2.2.2 - Operadores Binários - atuam sobre dois operandos

+ (adição)- (subtração)
***** (multiplicação)/ (divisão)**%** (mod) - fornece o resto da divisão de 2 nºs inteiros

ex.: 10 % 2 = **0** 11 % 2 = **1**

```
10    2                 11    2
 0    5                  1    5
```

2.2.3 – Precedência

++ , --
* , / , %
+ , -

Para alterar a precedência basta colocar a expressão entre parênteses. Quando dois operandos têm o mesmo nível de precedência, eles são avaliados da esquerda para a direita.

ex.:

```
(x + y) / 2      /*  será feito 1º a soma e depois a divisão  */
 x / y * 2       /*  será feito 1º a divisão e depois a multiplicação  */
```

2.3 - Operadores de Atribuição Compostos

sintaxe:
expressão_1 operador = expressão_2 é equivalente a

expressão_1 = expressão_1 operador expressão_2

ex.:

x = x * 5	**x * = 5**		
a = a + 1	**a + = 1**	ou	a ++
x = x / b	**x / = b**		
y = y – 1	**y - = 1**	ou	--y

2.4 - Operadores Relacionais

São usados para **comparar** expressões. Resultam em falso ou verdadeiro.

= =	(igual – comparação) - compara se 2 valores são iguais
>	(maior que)
<	(menor que)
>=	(maior ou igual)
<=	(menor ou igual)
!=	(diferente)

ex.:

```
4 = = 3          /*  resulta em falso  */
3 > 2            /* resulta em verdadeiro  */
```

2.4.1 - Precedência

< , <= , > , >=
! = , = =

2.5 - Operadores Lógicos

Permitem **relacionar** duas ou mais expressões.

&& (e) - resulta em verdadeiro se ambas expressões forem verdadeiras
| | (ou) - resulta em verdadeiro se pelo menos uma expressão for verdadeira
! (não) - resulta em verdadeiro se a expressão for falsa

ex.:

```
(5 > 2)  &&  (3 != 2)        /*  resulta em verdadeiro – ambos verdadeiros  */
(5 < 2)  &&  (3 != 2)        /*  resulta em falso – apenas 1 verdadeiro  */
(5 < 2)  &&  (3 = = 2)       /*  resulta em falso – ambos falsos  */

(3 >= 2)  | |  (4 != 2)      /*  resulta em verdadeiro – ambos verdadeiros */
(3 >= 2)  | |  (4 = = 2)     /*  resulta em verdadeiro – pelo menos 1 verdadeiro */
(3 <= 2)  | |  (4 = = 2)     /*  resulta em falso – ambos falsos  */

!(4 = = 2)          /*  resulta em verdadeiro – pois a expressão é falsa  */
!(4 != 2)           /*  resulta em falso – pois a expressão é verdadeira  */
```

3. Algoritmos, Fluxogramas e Programas

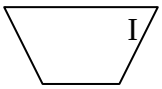
3.1 – Algoritmo

É uma lista de instruções para a execução **passo a passo** de algum processo. Todo algoritmo é composto por um grupo de ações primitivas (ações passíveis de execução por um humano ou uma máquina).
Ex.: receita de bolo, manual de instalação.

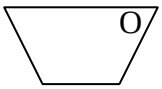
3.2 – Fluxograma ou Diagrama de Blocos

É um diagrama para a representação de um algoritmo.
Os símbolos de fluxograma adotados pela norma ANSI (American National Standards Institute) são apresentados a seguir:

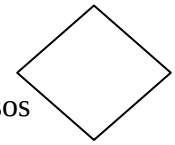
Processo



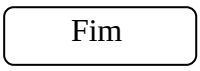
Entrada (Input)

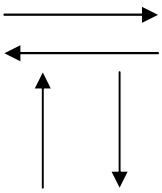


Saída (Output)



Decisão – Comparação (indica possibilidade de desvios para diversos pontos do programa).





Fluxo de direção

Conectores

3.3 – Programa

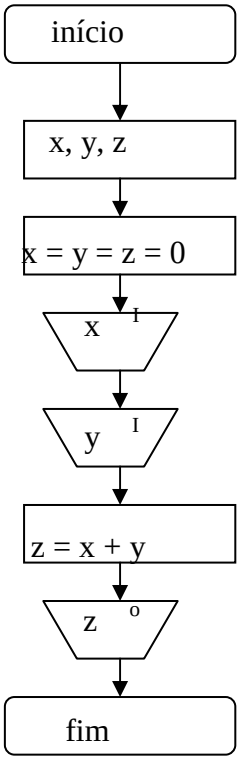
Após a elaboração do algoritmo e a construção do fluxograma correspondente, deverá ser criado o programa, que nada mais é do que a codificação do problema numa linguagem inteligível ao computador.

3.4 – Exemplos

1. Resolver a expressão z = x + y onde x e y são definidos pelo usuário.

Algoritmo:
1- início
2- declarar as variáveis: x, y, z
3- inicializar as variáveis: x = y = z = 0
4- pedir para o usuário digitar o valor de x (ler a variável x)
5- pedir para o usuário digitar o valor de y (ler a variável y)
6- calcular a expressão: z = x + y
7- mostrar o resultado: z
8- fim

Fluxograma:



4. Funções

4.1 - Funções da Biblioteca

4.1.1 – printf

Esta função imprime dados na tela e o arquivo de cabeçalho a ser incluído: **stdio.h**
sintaxe:

printf( );

Pode receber qualquer número de argumentos:

- **1º argumento:** chama-se string de formato e deve estar entre aspas duplas (") e pode conter:
  - Texto ex.: printf(“Bom dia”);
  - Códigos de barra invertida : o código de barra invertida mais utilizado é **\n** que significa nova linha (pular 1 linha na tela) ex.: printf(“Bom \n dia”);
  - Especificadores de formato: indica qual o tipo do conteúdo da variável a ser escrita

    % [largura][.precisão] **tipo**

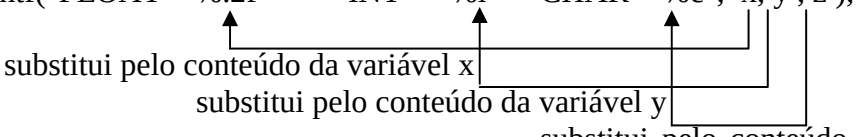
%c	1 único caracter
%s	2 ou + caracteres
%d ou %i	inteiro decimal
%f	ponto flutuante (double).
largura	especifica quantas casas antes da vírgula

	precisão	especifica quantas casas depois da vírgula

- **2º argumento em diante:** chama-se itens de dados e **não** vem entre aspas duplas, são tantos quantos forem os especificadores de formato do 1º argumento.
  ex.:

```
float   x = 1.25;
int     y = 1;
char    z = 'a';

printf("FLOAT = %.2f      INT =  %i     CHAR = %c",  x, y , z );
```



Saída na tela :

```
FLOAT = 1.25      INT = 1      CHAR = a
```

4.1.2 – scanf

Esta função lê dados do teclado e o arquivo de cabeçalho a ser incluído: **stdio.h**
sintaxe:

scanf( );

Pode receber qualquer número de argumentos:

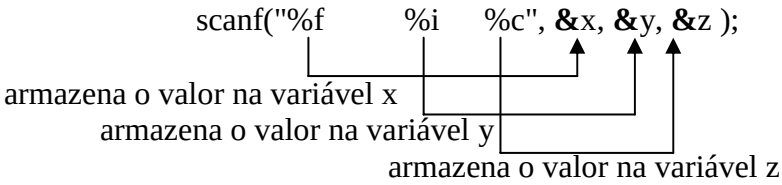
- **1º argumento:** chama-se string de formato e deve estar entre aspas duplas (") e pode conter:
  - Especificadores de formato: indica qual o tipo do conteúdo da variável a ser escrita

    % [largura][.precisão] **tipo**

%c	1 único caracter
%s	2 ou + caracteres
%d ou %i	inteiro decimal
%f	ponto flutuante (double)
largura	especifica quantas casas antes da vírgula
precisão	especifica quantas casas depois da vírgula

- **2º argumento em diante:** chama-se itens de dados e **não** vem entre aspas duplas, são tantos quantos forem os especificadores de formato do 1º argumento e devem ser precedidos pelo operador **& (endereço de)**.
  ex.:

```
float   x ;
int     y ;
char    z ;
```



Entrada na tela :

```
1.25    1    a
```

Observação:

Comentários em C - devem ser colocados para facilitar a manutenção do programa, como por exemplo em passagens que não estejam muito óbvias; explicando uma variável; resumindo o funcionamento de uma função, etc. Devem iniciar com um **/*** e terminar com ***/** podendo começar em uma linha e terminar em outra.

Ex.:

```
/*  Isto é um comentário   */

/*  Isto
é outro
comentário */
```

4.2 - Funções definidas pelo usuário

É uma seção de código independente e autônoma, escrita para desempenhar uma tarefa específica. Deve conter protótipo e definição da função.

4.2.1 – Protótipo da função

sintaxe:

tipo_retorno nome_função(tipo_arg nome$_1$, ..., tipo_arg nome$_n$);

tipo_retorno – tipo de variável que a função retornará. Pode ser char, int, float, double e void (se não retornar nada).

nome_função – descreve o que a função faz.

tipo_arg – tipo e nome das variáveis que serão passados para a função. Pode ser char, int, float, **nome$_n$** double, void.

- **Sempre termina com ; (ponto e vírgula) e vem nas componentes iniciais.**

4.2.2 – Definição da função

sintaxe:
tipo_retorno nome_função(tipo_arg nome$_1$, ..., tipo_arg nome$_n$)
{
instruções;
}

tipo_retorno – tipo de variável que a função retornará. Pode ser char, int, float, double e void (se não retornar nada).
nome_função – descreve o que a função faz.
tipo_arg – tipo e nome das variáveis que serão passados para a função. Pode ser char, int, float, **nome$_n$** double, void.

é a função propriamente dita. A 1ª linha é idêntica ao protótipo com exceção do **;** (ponto e vírgula). O corpo da função deve estar entre **{ }**(chaves).

- **Não termina com ; (ponto e vírgula), o corpo deve estar entre { } e vem após o final da função main , ou seja, após }/*main*/ (chave final).**

4.2.3 – Exemplo de função que NÃO retorna valor

protótipo: void moldura( );

referência a função dentro do programa: moldura( );

definição:
```
void moldura( )
{
printf("******");
printf("\n*     *");
```

```
printf("\n ******");
}
```

Saída na tela :

```
******
*     *
******
```

4.2.4 – Exemplo de função que RETORNA valor

protótipo: float divisão(int x, int y);

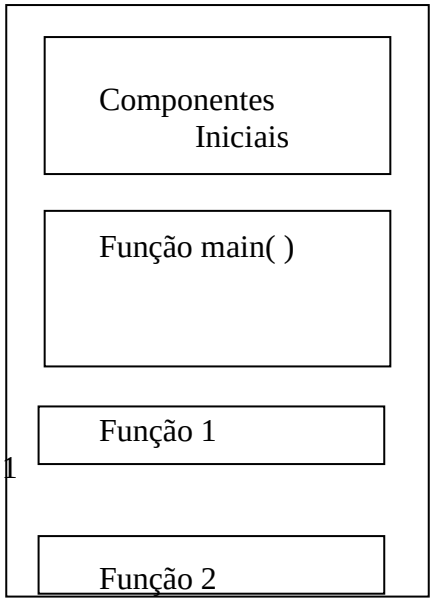
referência a função dentro do programa: d=divisão(a, b);

definição:

```
float divisão(int   x, int   y)
{
return(x / y);
}
```

- **Diferença:** **precisa do return( ) para devolver o resultado para d.**

4.3 - Estrutura de um Programa em C



contém:	arquivos de cabeçalho, declaração de constantes, protótipos e variáveis globais
contém:	variáveis locais e instruções do programa
contém:	variáveis locais e definição da função 1
contém:	variáveis locais e definição da função 2

4.4 - Exercícios

1. Resolver a expressão z = x + y onde x e y são definidos pelo usuário.

```
#include <stdio.h>
float soma (float a, float b);              /* protótipo da função soma */
main( )
{
float     x,y,z;
x=y=z=0;
printf("\n Este programa calcula a expressão Z=X+Y ");
printf("\n\n Digite o valor para x=  ");
scanf("%f", &x);
printf("\n\n Digite o valor para y=  ");
scanf("%f", &y);
z=soma(x, y);                               /* chamada a função soma */
printf("\n\n\n A expressão z=x+y para x=%.2f e y=%.2f é %.2f ", x, y, z);
} /* main */

float soma (float a, float b)               /* definição da função soma */
{
return(a+b);
} /* soma */
```

2. Ler 2 notas e calcular a média.

```
#include <stdio.h>
float media (float x, float y);             /* protótipo da função media */
main( )
{
float     a, b, m;
a=b=m=0;
printf("\n Este programa calcula a média de 2 notas ");
printf("\n\n Digite o valor da 1ª nota =  ");
scanf("%f", &a);
printf("\n\n Digite o valor da 2ª nota =  ");
scanf("%f", &b);
m=media(a, b);                              /* chamada à função media */
printf("\n\n\n A média das notas %.2f e %.2f é %.2f ", a, b, m);
} /* main */

float media (float x, float y)              /* definição da função media */
{
return((x+y)/2);
} /* media */
```

3. Pedir a idade para o usuário e calcular quantos meses e dias de vida ele tem aproximadamente.

```
#include <stdio.h>
int meses(int idade);                       /* protótipo da função meses */
int dias(int mes);                          /* protótipo da função dias */
```

```
main( )
{
int        idade, m, d;
idade= m = d = 0;
printf(“\n Este programa calcula quantos meses e dias de vida você tem ”);
printf(“\n\n Digite sua idade =  “);
scanf(“%i”, &idade);
m=meses(idade);                    /* chamada à função meses */
d=dias(m);                         /* chamada à função dias */
printf(“\n\n\n Você tem aproximadamente %i meses e %i dias de vida, m, d;
} /* main */

int meses (int idade)              /* definição da função meses */
{
return(idade * 12);
} /* meses */

int dias (int mes)                 /* definição da função dias */
{
return(mes * 30);
} /* dias */
```

4. **Calcular o consumo médio de gasolina de um tanque de automóvel. Pedir para o usuário entrar com a distância (km) e volume (litros). Cm = d (km) / v (litros)**

```
#include <stdio.h>
float divisao(float x, float y);   /* protótipo da função divisao */
main( )
{  float  d, v, cm;
d=v=cm=0;
printf(“\n Este programa calcula o consumo médio de gasolina ”);
printf(“\n\n Digite o valor da distância em km =  “);
scanf(“%f”, &d);
printf(“\n\n Digite o volume gasto em litros =  “);
scanf(“%f”, &v);
cm=divisao(d, v);                  /* chamada à função divisao */
printf(“\n\n\n O consumo médio de gasolina foi de %.2f ”, cm);
} /* main */

float divisao (float x, float y)   /* definição da função divisao */
{
return(x/y);
} /* divisao */
```