```
#include <stdio.h>
Void main()
{
printf(“Cheguei!\n");
}
```

Introdução a Linguagem C

Antes do C era a linguagem B

- **B** foi essencialmente uma simplificação da linguagem BCPL. B só tinha um tipo de dado, que correspondia com uma palavra de máquina. A maioria de operadores o manipulavam como um inteiro; por exemplo + (adição), - (subtracção), * (multiplicação) ou / (divisão). O sistema de tipos empregado em B se caracteriza por tipagem fraca, dinâmica e também implícita.
-

B EXEMPLO

```
main()
{
    auto n,num,soma,cont;

    num = 0;
    cont = 0;

    printf( "Digite um valor para N: *n*n");

    n=getchar();

    printf( "\nOs primeiros numeros perfeitos sao:\n");

    while (cont != n);
    {
        num =num+1;
        soma = 0;
                auto i=1;
        while(i <= num-1)
        {
            if( (num % i) == 0 )
            {
                soma =soma+ i;
            }
            i=i+1;
        }
        if ( soma == num )
        {
            printf("%d*n",soma);
            cont =cont+1;
        }
    }
}
```

C : ORIGENS

Ken Thompson e Dennis Ritchie (da esquerda para direita), os criadores das linguagens B e C, respectivamente.

- O desenvolvimento inicial de C ocorreu no AT&T Bell Labs entre 1969 e 1973.5 de acordo com Ritchie, o período mais criativo ocorreu em 1972. A linguagem foi chamada "C", porque suas características foram obtidas a partir de uma linguagem anteriormente chamado de " B", que de acordo com a Ken Thompson era versão reduzida da linguagem de programação BCPL.

C

- C é uma linguagem de programação compilada de propósito geral, estruturada, imperativa, procedural, padronizada pela ISO, criada em 1972, por Dennis Ritchie, no AT&T Bell Labs, para desenvolver o sistema operacional Unix (que foi originalmente escrito em Assembly).

- C é uma das linguagens de programação mais populares e existem poucas arquiteturas para as quais não existem compiladores para C. C tem influenciado muitas outras linguagens de programação, mais notavelmente C++, que originalmente começou como uma extensão para C

Estrutura geral de um programa em C

Descrição Narrativa

Cálculo da área do círculo

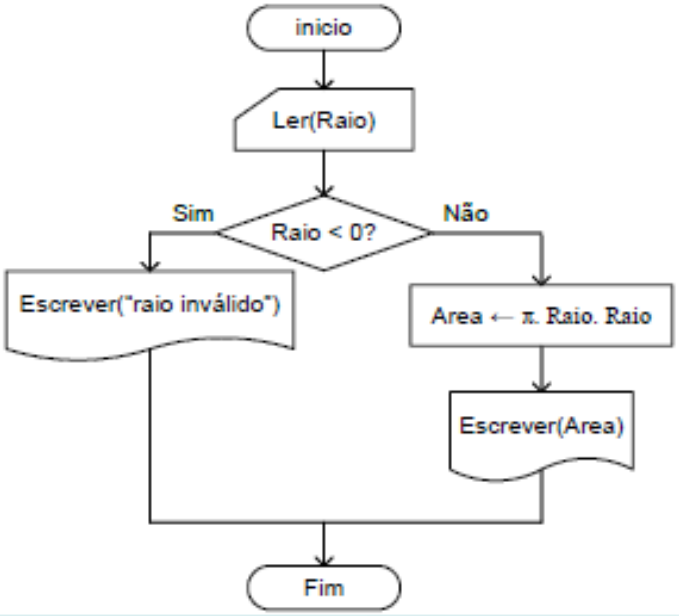
Obter o raio do círculo

Se o raio for negativo, escrever "raio invalido"

Se não, calcular a área do circulo usando a expressão ($A=\pi.R^2$)

Terminar o Algoritmo.

Fluxograma Convencional



Pseudo-código

```
Programa Calculo_da_area_do_círculo
Area, Raio: Real
PI = 3.14159
Inicio
Ler(Raio);
Se (Raio < 0) então
    Escreve("Raio invalido");
Senão
    Area ← PI * Raio * Raio;
    Escrever(Area);
Fim Se
Fim
```

Solução usando Linguagem C

Calcula_Area.c

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

int main( ){
 int raio;
 int Area;

scanf("%d",&raio);

 if(raio < 0){
  printf("raio inválido");
 }
 else{
  Area = 3.14 * raio * raio;
  printf("%f",Area);
 }
 exit(0);
}
```

Algoritmos e Linguagem C

Pseudo-código

```
Programa Calculo_da_area_do_círculo
Area, Raio: Real
PI = 3.14159
Inicio
   Ler(Raio);
   Se (Raio < 0) então
        Escreve("Raio invalido");
   Senão
        Area ← PI * Raio * Raio;
        Escrever(Area);
   Fim Se
Fim
```

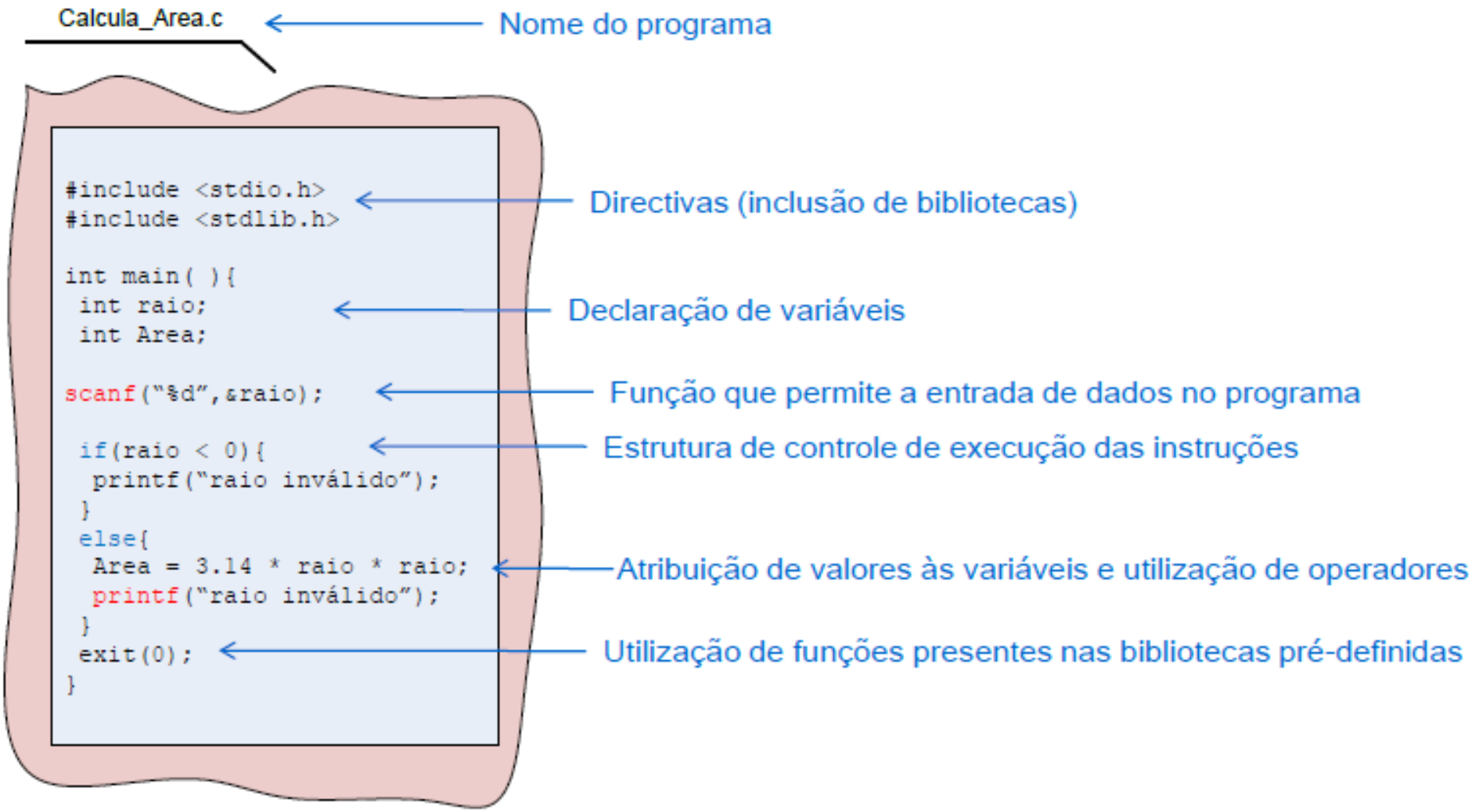
```c
#include <stdio.h>
#include <stdlib.h>

int main( ){
 int raio;
 int Area;

scanf("%d",&raio);

 if(raio < 0){
  printf("raio inválido");
 }
 else{
  Area = 3.14 * raio * raio;
  printf("%f",Area);
 }
 exit(0);
}
```

Estrutura geral de um programa em C



Estrutura geral de um programa em C

Regras gerais usadas em linguagem C

- Toda função tem um corpo delimitado por { }.
- Toda função é precedida de parênteses ().
- Todo programa em linguagem C deverá conter uma função ***main***.
- As linhas de códigos/instruções são sempre encerradas por *ponto-e-virgula* (;).
- Os comentários são delimitados por /* */ ou ainda //
  - Exemplo: /* engenharia informática */
  - ou //engenharia informática

C INICIANDO

- A linguagem C possui 32 palavras chaves.
- • As palavras chaves não podem ser usadas para nenhum outro propósito.
- • C diferencia palavras maiúsculas de minúsculas. Todas as palavras chaves devem ser escritas em minúsculo.
- • Exempo:
- – if (A > B) Correto!
- – IF (a > b) Errado!

Palavras reservadas

Palavras reservadas (palavras chave)

- – auto,break, case, char, const, continue,
- – default, do, double, else, enum, extern,
- – float, for, goto, if, int, long, register, return,
- – short, signed, sizeof, static, struct, switch,
- – typedef, union, unsigned, void, volatile, while.

Identificadores

- São os nomes que podem ser dados para variáveis e funções.
- Para a escolha destes nomes é necessário seguir alguams regras:
- Um identificador deve iniciar por uma letra ou por um "_" (*underscore*);
- A partir do segundo caracter pode conter letras, números e *underscore*;
- Deve-se usar nomes significativos dentro do contexto do programa;
- C é uma linguagem *case-sensitive*, ou seja, faz diferença entre nomes com letras maiúsculas e nomes com letras minúsculas. Peso e peso são diferentes;
- Costuma-se usar maiúsculas e minúsculas para separar palavras: "PesoDoCarro";
- Deve ser diferente dos comandos da linguagem;
- Pode conter números a partir do segundo caracter;
- Exemplos:
- Idade, Contador, PesoDoCarro, Usuario_1, RaioDoCirculo

Variáveis

- Uma variável é uma posição de memória que pode ser identificada através de um nome.
- Podem ter seu conteúdo alterado por um ***comando de atribuição***.
- Após a atribuição mudam de valor.

Tipos de Variáveis

- Todas as variáveis em C tem um ***tipo***;
- Cada tipo define os valores que a variável pode armazenar;
- Cada tipo ocupa uma certa quantidade de memória.

VARIÁVEIS - TIPOS

Tipo	Tamanho	Valores Válidos
char	1 byte	letras e símbolos: 'a', 'b', 'H', '^', '*', '1', '0'
int	2 bytes	de -32767 até 32767 (apenas números inteiros)
float	4 bytes	de -3.4×10^{38} até $+3.4 \times 10^{+38}$ com até 6 dígitos de precisão
double	8 bytess	de -1.7×10^{308} até $+1.7 \times 10^{+308}$ com até 10 dígitos de precisão

Declaração de Variáveis

Todas as variáveis **tem que ser declaradas *antes*** de serem usadas;

Não há uma inicialização implícita na declaração

TIPOS DE DADOS BÁSICOS

Tipo	Extensão (em *bits*)	Faixa Mínima		
void	0	Sem valor		
char	8	-127	a	127
unsigned char	8	0	a	255
signed char	8	-127	a	127
int	16	-32.767	a	32.767
unsigned int	16	0	a	65.535
signed int	16	-32.767	a	32.767
short int	16	-32.767	a	32.767
unsigned short int	16	0	a	65.535
signed short int	16	-32.767	a	32.767
long	32	-2.147.483.647	a	2.147.483.647
long int	32	-2.147.483.647	a	2.147.483.647
unsigned long int	32	0	a	4.294.967.295
signed long int	32	-2.147.483.647	a	2.147.483.647
float	32	-3.4×10^{38}	a	$3.4 \times 10^{+38}$
double	64	-1.7×10^{308}	a	$1.7 \times 10^{+308}$
long double	80	-3.4×10^{4932}	a	$1.1 \times 10^{+4932}$

Exemplo

```
// Exemplo de programa em C

#include <stdio.h>   //  Arquivo de cabeçalho (header)
void main()
{
   int contador;              // declarações simples
   float PrecoDoQuilo;
   double TaxaDeCambio;
   char LetraDigitada;
   int IdadeManoel, IdadeJoao, IdadeMaria;  // Pode colocar mais de uma variável na
                                            // na mesma linha

   double  TaxaDoDolar,
           TaxaDoMarco,
           TaxaDoPeso,         // Também pode trocar de linha no meio
           TaxaDoFranco;


.......
}
```

INICIALIZAÇÃO

```
// Exemplo de programa em C

#include <stdio.h>   //  Arquivo de cabeçalho (header)
void main()
{
   int NroDeHoras = 0;              // declara e inicializa com Zero
   float PrecoDoQuilo = 10.53;      // declara e inicializa com 10.53
   double  TaxaDoDolar = 1.8,
           TaxaDoMarco = 1.956,
           TaxaDoPeso = 1.75,
           TaxaDoFranco = 0.2;

.......
}
```

```
#include <stdio.h>

main()
{
    float f1 = 16777216; // 2^24
    float f2 = 16777217; // 2^24 + 1
    double d = 16777217;

    printf("%f\n", f1);
    printf("%f\n", f2);
    printf("%f\n", d);
}
```

Constantes

- Constantes são identificadores que não podem ter seus valores alterados durante a execução do programa.
- Para criar uma constante existe o comando #define que, em geral é colocado no início do programa-fonte.

Constantes

Para criar uma constante existe o comando #define que, em geral é colocado no início do programa-fonte, ou seja, no seu cabeçalho [PIN 99, SCH 96]. Exemplos:

```
#define LARGURA_MAXIMA          50
#define NRO_DE_DIAS_DA_SEMANA 7
#define NRO_DE_HORAS_DO_DIA    24
#define FALSO                   0
#define VERDADEIRO              1
#define UNIVERSIDADE            "Cambridge University"
#define TERMINO                 '/'
#define VALOR_DE_PI             3.1415  // Obs: não se coloca ponto-e-vírgula após o valor

void main ()
{
        int TotalDeHoras;
        TotalDeHoras = 10 * NRO_DE_DIAS_DA_SEMANA * NRO_DE_HORAS_DO_DIA;
        printf("Local: %s", UNIVERSIDADE);
        .....
}
```

Exemplo: Constantes

```
#define LARGURA_MAXIMA 50            // Não se coloca ponto-e-vírgula após o valor
#define NRO_DE_DIAS_DA_SEMANA 7
#define NRO_DE_HORAS_DO_DIA 24
#define VALOR_DE_PI  3.1415

void main ()
{
   int TotalDeHoras;

    TotalDeHoras = 10 * NRO_DE_DIAS_DA_SEMANA * NRO_DE_HORAS_DO_DIA;
......
}
```

VARIÁVEIS : ESCOPO

Variáveis *locais* são aquelas declaradas dentro de uma função. Em algumas literaturas de C, variáveis locais são referidas como variáveis "automáticas", porque em C pode-se usar a palavra reservada *auto* para declará-las. Variáveis locais só podem ser referenciadas por comandos que estão dentro do bloco no qual as variáveis foram declaradas, ou seja, elas não são reconhecidas fora de seu próprio bloco de código, que começa com { e termina com }.

```
void func1(void)
{
        int x;          // Variável local
        x = 10;
}

void func2(void)
{
        int x;          // Variável local
        x = -199;         (não tem relação com a anterior)
}
:
```

VARIÁVEIS : ESCOPO

Variáveis *globais* são aquelas reconhecidas pelo programa inteiro, isto é, que podem ser usadas por qualquer pedaço de código e que guardam seus valores durante toda a execução do programa. Para criar variáveis globais basta declará-las fora de qualquer função.

```
#include <stdio.h>

int cont;           // Variável global

void func1(void);
void func2(void);

void main (void) {
        cont = 100;
        func1();
}

void func1(void) {
        int temp;
        temp = cont;
        func2();
        printf("cont eh %d", cont);          // Exibirá 100
}

void func2(void) {
        int cont;
        for (cont=1; cont<10; cont++)
                putchar('.');
}
```

Operadores Aritméticos

- Binários: + (soma) - (subtração) * (multiplicação) / (divisão) % (resto da divisão)
- Unários: - (troca de sinal)
- Quando os operandos são inteiros, a divisão é inteira (a parte fracionária é truncada)
- O operador % não permite operandos `float` ou `double`

Operadores Aritméticos

```c
#include <stdio.h>

main()
{
    int a = 7, b = 2;

    printf("%d\n", a + b);
    printf("%d\n", a - b);
    printf("%d\n", a * b);
    printf("%d\n", a / b);
    printf("%f\n", ((double) a) / b);
    printf("%d\n", a % b);
    printf("%d\n", -a);
}
```

Operadores relacionais e lógicos

- > >= < <= == (igual) != (diferente)
- && (e) || (ou) ! (não)
- Expressões são avaliadas das esquerda para a direita, até o resultado ser conhecido
  - Ou seja, parte da expressão pode não ser avaliada
  - Muitos algoritmos levam isso em conta (para garantir corretude)
- O número zero representa o valor FALSO, e qualquer número diferente de zero representa o VERDADEIRO
  - Expressões lógicas retornam 1 (verdadeiro) ou 0 (falso)
- Cuidado: é permitido fazer atribuições dentro de expressões. Erro comum: trocar igual (==) com atribuição (=)
  - Ex.:
```
x = 2;     (x == 3); // vale 0 (FALSO)
x = 2;     (x = 3);  // vale 3 (VERDADEIRO)
```
- O operador ! converte 0 em 1, e qualquer valor diferente de 0 em 0

Operadores relacionais e lógicos

```c
#include <stdio.h>

main()
{
    int a = 7, b = 2;

    printf("%d\n", a == b);
    printf("%d\n", a != b);
    printf("%d\n", (a > 0) && (b <= 2));
    printf("%d\n", !a);

    printf("%d\n", a == 3);
    printf("%d\n", a = 3);
    printf("%d\n", a);
}
```

Operadores de incremento e decremento

- ++ (soma 1 à variável) -- (subtrai 1 à variável)

```
int x = 3;
x++; // agora x vale 4
```

- Prefixado (++/-- antes da variável): utiliza na expressão o valor da variável **depois** do incremento
- Pós-fixado (++/-- depois da variável): utiliza na expressão o valor da variável **antes** do incremento

```
int x = 3, y;
y = x++; // agora x vale 4 e y vale 3
y = ++x; // agora x e y valem 5
```

Operadores de atribuição

- = (atribuição simples)
- += -= *= /= %= <<= >>= &= |= ^=

```
int x = 2;
x = 2*3 + 1 // x vale 7
x += 2;  // x vale 9 (o mesmo que x = x + 2)
x -= 5;  // x vale 4 (o mesmo que x = x - 5)
x *= 2;  // x vale 8 (o mesmo que x = x * 2)
x /= 3;  // x vale 2 (o mesmo que x = x / 3)
x %= 3;  // x vale 2 (o mesmo que x = x % 3)
x <<= 1; // x vale 4 (o mesmo que x = x << 1)
x >>= 1; // x vale 1 (o mesmo que x = x << 2)
x &= 3;  // x vale 1 (o mesmo que x = x & 3)
x |= 4;  // x vale 5 (o mesmo que x = x | 4)
x ^= 7;  // x vale 2 (o mesmo que x = x ^ 7)
```

A FUNÇÃO PRINTF

- A função printf é parte de um conjunto de funções pré-definidas armazenadas em uma biblioteca padrão de rotinas da linguagem C.
- Ela permite apresentar na tela os valores de qualquer tipo de dado.
- Para tanto, printf utiliza o mecanismo de *formatação*, que permite traduzir a representação interna de variáveis para a representação ASCII que pode ser apresentada na tela.

A função printf

- O primeiro argumento de printf é um *string de controle*, uma seqüência de caracteres entre aspas. Esta *string*, que sempre deve estar presente, pode especificar através de caracteres especiais (as *seqüências de conversão*) quantos outros argumentos estarão presentes nesta invocação da função.

A FUNÇÃO PRINTF

- Por exemplo, se o valor de uma variável inteira x é 12, então a execução da função

printf("Valor de x = %d", x);

imprime na tela a frase Valor de x = 12.

Se y é uma variável do tipo caráter com valor 'A', então a execução de printf("x = %d e y = %c\n", x, y);
imprime na tela a frase x = 12 e y = A

A FUNÇÃO PRINTF

```
#include <stdio.h>

int main()
   {
     int nr_pos,nr_neg;

     nr_pos = 3;
     nr_neg = -3;

     printf("nr_pos = %+d\n",nr_pos);
     printf("nr_neg = %d\n",nr_neg);

     return(0);
   }
```

PRINTF – FORMATANDO A SAIDA

SINTAXE	EFEITO
printf (" %5d ",valor);	exibe valor com um mínimo de 5 caracteres
printf(" %05d ",valor);	exibe valor com um mínimo de 5 caracteres precedendo-o com zeros
#%o	exibe um valor octal precedido de 0 (zero)
#%x	exibe um valor hexadecimal precedido de 0x
#%X	

```
int x = 31;
printf("%d\n", x); // 31
printf("%o\n", x); // 37
printf("%x\n", x); // 1f

x = -1;
printf("%u\n", x); // 4294967295

x = 64;
printf("%c\n", x); // A

double d = 1200.0;
printf("%f\n", d); // 1200.000000
printf("%e\n", x); // 1.200000e+003
printf("%g\n", x); // 1200.000000
```

A FUNÇÃO PRINTF

```
#include <stdio.h>       // Necessário para usar a função printf
                         // A função printf exibe um ou mais dados na tela

void main ()
{
   printf("%s","Isto é uma string ....\n"); // note o '\n' no final da string;
   printf("%s","Outra string ....");
   printf("%s","Terceira string\n");

//Depois de Executar o programa, tecle ALT-F5 para ver o resultado na tela
}
```

Mais printf

```
#define UNIVERSIDADE "Pontifícia Universidade Católica do Rio Grande do Sul"
        // deve-se colocar entre aspas

#include <sdtio.h>
#include <conio.h>   // necessário para as funções clrscr e getch

void main ()
{
   clrscr();      // Limpa a tela
   printf("%s", UNIVERSIDADE); // Imprime o nome representado pela constante
   getch();       // espera que o usuário pressione uma tecla
}
```

+ PRINTF

```
// Impressão de Variáveis Inteiras
#include <stdio.h>
#include <conio.h>   // necessário para as funções clrscr e getch

void main ()
{
   int Contador;
   int NroDeFilhos;

   clrscr();     // Limpa a tela

   Contador = 10;
   printf("Valor da Variável: %d\n", Contador);          // No momento da execução sinal %d vai
                                                  // ser substituído pelo valor da
                                                  // variável Contador

   NroDeFilhos = 3;
   printf("Maria tem %d filhos", NroDeFilhos);  // o inteiro pode ficar no meio da string

   getch();      // espera que o usuário pressione uma tecla
}
```

PRINTF COM FÓRMULAS

```
// Impressão de Expressões aritméticas
#include <stdio.h>
#include <conio.h>   // necessário para as funções clrscr e getch

void main ()
{
   int NroDeAndares;
   int AlturaPorAndar;

   clrscr();     // Limpa a tela

   NroDeAndares = 7;
   AlturaPorAndar = 3;

   printf("Altura Total do Prédio: %d metros", NroDeAndares*AlturaPorAndar);
        // No momento da execução sinal %d vai ser substituído
        // pelo valor da multiplicação

   getch();      // espera que o usuário pressione uma tecla
}
```

PRINTF COM VARIÁVEIS REAIS

```
// Impressão de números reais
#include <stdio.h>
#include <conio.h>   // necessário para as funções clrscr e getch

void main ()
{
   float NotaDaP1, NotaDaP2;
   float Media;


   clrscr();     // Limpa a tela

   NotaDaP1 = 6.6;  // Atribuição do Valores das médias
   NotaDaP2 = 8.2;

   Media = (NotaDaP1 + NotaDaP2) / 2.0;

   printf("Média Final : %f", Media);
        // No momento da execução sinal %f vai ser substituído
        // pelo valor da variável Media com SEIS casas decimais
        // Média Final : 7.400000
   getch();      // espera que o usuário pressione uma tecla
}
```

Formatando números reais na saida

```c
#include <stdio.h>
#include <conio.h>
void main()
{
   float Numero;
   Numero = -2.5;
   clrscr();
   printf("1234567890\n");
   printf("%7f\n", Numero);
   printf("%7.0f\n", Numero);
   printf("%7.3f\n", Numero);
   printf("%8.3f\n", Numero);
   printf("%9.3f\n", Numero);
   printf("\n");
   printf("%8.4f\n", Numero);
   printf("%8.1f\n", Numero);
   printf("%6.12f\n", Numero);

   getch();
}
```

```
// Resultados

1234567890
-2.500000
     -2
 -2.500
  -2.500
   -2.500

 -2.5000
    -2.5
-2.500000000000
```

Variáveis tipo string

- Uma variável capaz de armazenar uma string deve ser declarada informando-se qual o número máximo de caracteres que ela poderá armazenar.

- **Exemplo:** char Nome[30]; // isto define que a variável poderá armazenar uma // string de até **29** caracteres**.**

- As atribuições de valores a strings **devem** ser feitas através da função **strcpy**

- Ao trabalharmos com strings deve-se incluir o arquivo de cabeçalho ***string.h***

String exemplo

```
// Exemplo com strings

#include <stdio.h>
#include <conio.h>
#include <string.h>  // arquivo de cabeçalho para trabalhar com strings

void main()
{
   char Nome[30];  // declara uma string que poderá armazenar até 29 caracteres !!

   clsrscr();

   strcpy(Nome, "Jose da Silva");  // atribui "Jose da Silva" para a variável Nome
   printf("O funcionário %s foi tranferido", Nome);  // no lugar de %s aparecerá o
                                                     // conteúdo da variável Nome

   getch();
}
```

Entrada via scanf

Entrada formatada

```
scanf("<formato>", &var1, &var2, ..., varN);
```

- Lê da entrada padrão (teclado) uma sequência de caracteres, interpreta de acordo com <formato>, e grava os valores obtidos nas variáveis (na ordem encontrada)

```
int i, r; float d;
r = scanf("%d, %f", &i, &d);
// espera inteiro e real separados p/ vírgula
```

SCANF

- Espera que o usuário digite um inteiro. O valor digitado será o conteúdo da variável `n`.

```
scanf("%d", &n);
```

- Espera que o usuário digite dois inteiros. O primeiro valor digitado será o conteúdo da variável `m` e o segundo valor será o conteúdo da variável `n`.

```
scanf("%d %d", &m, &n);
```

- O usuário deve digitar `n` números. Note que o `printf` tem como finalidade somente orientar o usuário para a digitação dos números.

```
for (i = 0; i < n; i++){
  printf("Digite mais um número: ");
  scanf("%d", &num);
}
```

Comando if

- O comando IF serve para **alterar o fluxo de execução de um programa** em C baseado no valor, **verdadeiro ou falso**, de uma **expressão lógica.**

Formato 1:

```
if (expr_log)
   comando1; // executado se "expr_log" for verdadeira
comando2; // executado sempre independente da condição
```

Formato 2:

```
if (expr_log)
  comando1; // executado se "expr_log" for verdadeira
else comando2; // executado se "expr_log" for falsa
comando3; // executado sempre, independente
          // do resultado do teste
```

COMANDO IF

```
#include <stdio.h>
int main ()
{
        int num;
        printf ("Digite um numero: ");
        scanf ("%d",&num);
        if (num>10)
                printf ("\n\nO numero e maior que 10");
        if (num==10)
        {
                printf ("\n\nVoce acertou!\n");
                printf ("O numero e igual a 10.");
        }
        if (num<10)
                printf ("\n\nO numero e menor que 10");
        return(0);
}
```

Comando If-else

```
#include <stdio.h>
int main ()
{
        int num;
        printf ("Digite um numero: ");
        scanf ("%d",&num);
        if (num==10)
        {
                printf ("\n\nVoce acertou!\n");
                printf ("O numero e igual a 10.\n");
        }
        else
        {
                printf ("\n\nVoce errou!\n");
                printf ("O numero e diferente de 10.\n");
        }
        return(0);
}
```

COMANDO IF-ELSE-IF

```
#include <stdio.h>
int main ()
{
        int num;
        printf ("Digite um numero: ");
        scanf ("%d",&num);
        if (num>10)
                printf ("\n\nO numero e maior que 10");
        else if (num==10)
        {
                printf ("\n\nVoce acertou!\n");
                printf ("O numero e igual a 10.");
        }
        else if (num<10)
             printf ("\n\nO numero e menor que 10");
        return(0);
}
```

Comando while

Sintaxe

- while (condição) comando
  onde comando pode corresponder a uma instrução simples ou a uma seqüência de instruções entre chaves separados por " ; " e condição é uma expressão lógica, cujo resultado pode ser verdadeiro ou falso.

Descrição

- Enquanto a condição for verdadeira, o comando é repetido. O comando pode ser apenas uma instrução do C ou um bloco de instruções entre chaves.

WHILE - EXEMPLO

```
#include <stdio.h>
#include <stdlib.h>
int main () {
  /* declaracoes */
  int num;  /* variavel utilizada para  leitura da   sequencia */
  int quad; /* variavel que armazena o quadrado de um   numero */
  /* programa */
  printf("Digite uma sequencia terminada por zero\n");
  scanf("%d", &num);
  while (num != 0) /* os simbolos '!=' significam diferente */
    {
      quad = num * num ;
      printf ("O quadrado de %d = %d\n", num, quad);
      scanf("%d", &num);
    }
  /* fim do programa */
  system ("pause");
  return 0;
}
```

While – exemplo 2

```
#include <stdio.h>
#include <stdlib.h>
int main ( )
{
  /* Declaracoes */
  int num, soma, digito;
  printf ("Entre com um numero: ");
  scanf ("%d", &num);
  soma = 0;
  while (num != 0) /* enquanto houver digitos para  processar */
    {
      digito = num % 10; /* pega o digito mais a direita */
      num = num / 10; /* atualiza o numero, mantedo apenas os digitos não processados */
      soma = soma + digito; /* atualiza a soma dos digitos */
    }
  printf ("Soma dos digitos: %d.\n", soma);
  system("pause");
  return 0;
}
```

Comando de repetição for

for (<inicialização>; <condição>; <incremento>) <comandos>

- A parte de <inicialização> é realizada apenas 1 vez, no início da execução do comando.
- A seguir, a <condição> é testada, e caso verdadeira, os <comandos> são executados.
- Após a execução dos <comandos> mas antes de testar a <condição>, a parte <incremento> do comando for é executada.

FOR - EXEMPLO

```
#include <stdio.h>
#int main ()
{
   int n, cont, fat;
   printf("Entre com um numero para calculo do fatorial: ");
   scanf("%d", &n);
   fat = 1;
   for (cont = 1; cont <= n; cont=cont+1)
     fat = fat * cont;
   return 0;
}
```