

Linguagem C - Resumão

Prof^a. Rose Yuri Shimizu

Programas C - Estrutura básica

TODA INSTRUÇÃO TERMINA COM “;”

```
1 /*
2     Estrutura básica de um programa em C
3     stdio.h : cabeçalho das funções de entrada e saída
4 */
5 #include <stdio.h> //inclusão da biblioteca
6
7 //função principal
8 int main() {
9     //declaração de variáveis
10    int a; //toda instrução termina com ;
11    int A; //case sensitive
12    char letra;
13    float real1;
14    double real2;
15
16    return 0; //padrao C: return 0 sucesso e nao-zero erro.
17 }
```

Declaração de variáveis - Tipos de dados - Intervalos

Tipo	Tamanho	Intervalo \approx
char	8 bits	-128 \rightarrow 127
short	16 bits	-32 mil \rightarrow 32 mil
unsigned short	16	0 \rightarrow 65 mil
int	32 bits	-2 bilhões \rightarrow 2 bilhões
unsigned int	32	0 \rightarrow 4 bilhões
long	64 bits	-9 quintilhões \rightarrow 9 quintilhões
long long	64 bits	-9 quintilhões \rightarrow 9 quintilhões
unsigned long	64	0 \rightarrow 18 quintilhões
float	32 bits	$-3.4 \times 10^{38} \rightarrow 3.4 \times 10^{38}$
double	64 bits	$-1.7 \times 10^{308} \rightarrow 1.7 \times 10^{308}$

- float: 6 dígitos de precisão decimal (restante é aproximação)
- double: 15 dígitos de precisão decimal (restante é aproximação)
- Exemplo:
 - ▶ float **1123456123456123456** = **1123456104810938368**
 - ▶ double **1123456123456123456** = **1123456123456123392**

Declaração de variáveis - Tipos de dados - Memória

'Realistic' 32-bit memory map

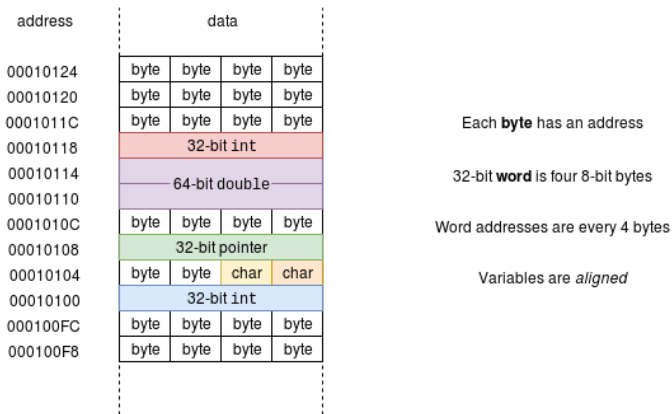


Figura: <https://xerxes.cs.manchester.ac.uk/comp251/kb/images/variables.png>

Inicialização de variáveis (atribuição)

```
1 int main()
2 {
3     //Declaração
4     int pera;
5     char qualidade;
6     float peso;
7
8     //Inicialização
9     pera = 3;
10    qualidade = 'A';
11    peso = 0.653;
12
13    //Inicialização na declaração
14    int maca = 1;
15
16    //Várias declarações
17    int a, b=9, c;
18 }
```

Função printf()

- Imprimir mensagens e valores de variáveis

```
1 int X = 3;
2 int Y = 2;
3
4 //           |-----|
5 printf("X vale %d e Y vale %d\n", X, Y);
6 //           |-----|
7
8 printf("Olá Mundo!\n");
```

- Tudo que está entre aspas dupla será impresso na tela
- **Especificadores de formato:** tipo do dado que será substituído pelo valor da variável
- **man 3 printf**

Tipo	Especificador
int	%d
long	%ld
char	%c
float	%f
double	%lf

Função scanf()

```
1 #include <stdio.h>
2
3 int main(){
4     //Declaração das variáveis
5     float y;
6     int x;
7
8     //Inicialização pela entrada padrão (teclado)
9     scanf("%f %d", &y, &x); //& : endereço da variável
10
11     //Saída
12     printf("Você digitou %f %d\n", y, x);
13     return 0;
14 }
```

```
1 scanf("%d %d",&a,&b);
2     //^ nao pode ter espaco entre o ultima entrada e o
   fecha aspas
```

Operadores Aritméticos

- Operador de Atribuição =:
 - ▶ (soma = $a + b$) \neq ($a + b$ = soma)
- Operadores Aritméticos:

Operador	Exemplo
+ (adição)	$x = y + 1$
- (subtração)	$x = y - 1$
/ (divisão)	$x = y/z$
* (multiplicação)	$x = y*10$
% (resto)	$x = y\%4$
++ (mais um)	$++x \rightarrow x = x + 1$
-- (menos um)	$--x \rightarrow x = x - 1$

Operadores Relacionais

- Compara a relação entre dois valores ou expressões;
- Tipos
 - ▶ $<$ Menor
 - ▶ $>$ Maior
 - ▶ \leq Menor ou igual
 - ▶ \geq Maior ou igual
 - ▶ $==$ Igual
 - ▶ $!=$ Diferente
- Em C: $==$ é diferente de $=$
 - ▶ $=$ é atribuição, $==$ é comparação
- O resultado é 1 (verdade) ou 0 (falso)

Operadores Lógicos

- Compara a relação entre dois valores ou expressões
- Lógica proposicional através da álgebra booleana (George Boole, matemático inglês)
- Operadores lógicos em C
 - ▶ && E : “todos”
 - ▶ || OU : “algum”
 - ▶ ! Não : inverso

Operadores Lógicos

- P: Está chovendo
- Q: Solo está seco
- Equivalência lógica em C:

```
1 int chuva = 1;  
2 int seco = 1;  
3  
4 int ativar_sensor = !chuva && seco;
```

- Tabela verdade (2^2 linhas)

p	q	$\neg p \wedge q$
V	V	F
V	F	F
F	V	V
F	F	F

Expressão aritmética, relacional e lógica: precedências

- 1 ()
 - 2 !
 - 3 * / %
 - 4 + -
 - 5 < > <= >=
 - 6 == !=
 - 7 &&
 - 8 ||
 - 9 =
- a || ! (b && c)

Controle de fluxo : IF

```
1 #include <stdio.h>
2 int main(){
3     char c;
4     scanf("%c",&c);
5
6     if(c=='a' || c=='A' || c=='e' || c=='E' ||
7         c=='i' || c=='I' || c=='o' || c=='O' ||
8         c=='u' || c=='U')
9     {
10        if(c>='A' && c<='U') //intervalo: tabela ASCII
11            printf("%c:vogal maiúscula\n",c);
12        else
13            printf("%c:vogal minúscula\n",c);
14    }
15    else if(c>='0' && c<='9')
16    {
17        printf("%c:digito\n",c);
18    }
19    else
20    {
21        printf("%c:consoante\n",c);
22    }
23    return 0;
24 }
```

Estrutura de Repetição: WHILE e FOR

```
1 //n! = n*(n-1)*(n-2)*(n-3)*...*3*2*1
2 //declarar
3 int fat, n, i;
4
5 //inicializar
6 scanf("%d", &n);
7 fat = 1;
8 i = n;
9
10 //repetir - verificar
11 while(n >= 1) {
12     fat = fat * n;
13
14     //atualizar
15     n--;
16 }
17
18 printf("%d! = %d\n", i, fat);
19
20 for(fat=1; i >= 1; i--) {
21     fat = fat * i;
22 }
```

Estrutura de Repetição: WHILE e FOR

```
inicializações
while(condições)
{
    ...
    atualizações
}
```

```
for( inicializações; condições; atualizações )
{
    ...
}
```

Estrutura de Repetição: leituras seguidas

```
1 scanf("%d", &a);
2 while(a > 0){
3     printf("%d\n", a);
4     scanf("%d", &a);
5 }
```

```
1 while(scanf("%d", &a), a > 0){
2     printf("%d\n", a);
3 }
```

```
1 while(scanf("%d", &a) && a > 0){
2     printf("%d\n", a);
3 }
```

```
1 // | espaço para leituras seguidas de caracteres
2 while(scanf(" %c", &a)){
3     printf("%c\n", a);
4 }
```


Função e Procedimento

- Definido por um conjunto de instruções que fazem uma tarefa específica
- Somente é executada ao ser utilizada/chamada/invocada
- Funções podem utilizar outras funções
- Exemplos: printf, scanf, strlen, strcmp, etc.
- Componentes:

```
1 tipo_retorno nome(parametros)
2 {
3     ...
4     tipo_retorno a;
5     ...
6     return a;
7 }
```

- ▶ **tipo_retorno**: o que a função devolve para o código que chamou:
 - ★ int, double, float, char;
 - ★ void (vazio, nada);
- ▶ **nome_funcao**: o nome utilizado para chamar a função;
- ▶ **parametros**: lista de declarações de variáveis separadas por vírgulas;
- ▶ **return**: palavra reservada que indica o que será retornado;

Retorno de funções

Retorno do scanf

- A função **scanf** retorna:
 - ▶ `man scanf`
 - ▶ **Número de itens** de entrada **combinados e atribuído** com sucesso;
 - ▶ O valor **EOF** é retornado se o final da entrada é alcançado antes da primeira leitura ou falha de correspondência ou erro de leitura.

```
1 while (scanf("%d", &d) != EOF) { } //ctrl+d = end of file
2 while (scanf("%d", &d) == 1) { }
3
```

```
1 int d;
2 while (scanf("%d", &d) != EOF) //ctrl+d = end of file
3 {
4     printf("%d\n", d);
5 }
```

Retorno de funções

Retorno do printf

- A função **printf** retorna:
 - ▶ `man 3 printf`
 - ▶ Sucesso: **número de caracteres impressos**
 - ▶ Erro: **número negativo**

```
1 int a = printf("alo\n");
2 printf("%d\n", a);
3 /*
4 Saida
5 alo
6 4
7 */
8
```

```
1 int somas() {
2     int a, s=0;
3     while(scanf("%d", &a) != EOF)
4         s=s+a;
5     return s;
6 }
7
8 void imprime() {
9     printf("Ola mundo!\n");
10 }
11
12 float obtem_valor() {
13     float valor;
14     printf("Entre um valor:");
15     scanf("%f", &valor);
16     return valor;
17 }
18
19 void imprime_olas() {
20     int a;
21     while(scanf("%d", &a) == 1 && a>0){
22         if(a>1000) return;
23         while(a-->0)
24             printf("Ola mundo!\n");
25     }
26 }
```

Parâmetros: passando valores para as funções

- Passar argumentos (valores) através dos parâmetros (variáveis);

```
1 int acumulador(int n, int m) { //parâmetros
2     int s=0, i;
3     while(n){
4         scanf("%d", &i);
5         if(i<m)
6             s+=i; //s=s+i;
7         n--;
8     }
9     return s;
10 }
11
12 int funcao() {
13     int n, max;
14     scanf("%d %d", &n, &max); //entrada 10 100
15     return acumulador(n, max); //argumentos n e max
16                                 //valor de n no retorno?
17 }
```

Passagem de argumentos por valor

- **Cópia** do valor original
- Não altera a variável original
- Variável da função \neq variável original

```
1 void teste (char k) {
2     printf("%c\n", k); //podemos utilizar o valor de k
3     k='a';             //podemos alterar o valor de k
4     printf("%c\n", k); //a
5 }
6
7 int main() {
8     teste('p'); //passando o valor direto
9
10    char k = 'b';
11
12    teste(k); //passando uma copia do valor de k
13    printf("%c\n", k); //b - nao altera o valor de k
14
15    return 0;
16 }
```

Passagem de argumentos por referência

- **Passar o endereço (referência) de uma variável**
- As **alterações** são feitas **diretamente no endereço** da variável original
- **Ponteiro:**
 - ▶ Variável especial que armazena endereços
 - ▶ Identificado pelo * (asterisco)
 - ▶ Armazena o endereço do local onde está o conteúdo

```
1 void f1(int x, int *i){
2     printf("%d %d\n", x, *i); //1 0
3     *i = 4; //alterando o conteúdo no endereço apontado por i
4     x = 5; //alterando o conteúdo da variável x
5 }
6
7 int main(){
8     int a=1, c=0;
9
10    f1(a, &c); //conteudo de a e endereço de c
11    printf("%d %d\n", a, c); //1 4
12    return 0;
13 }
```

Função e Procedimento

```
1 //função
2 void soma(int a, int b)
3 {
4     int c = a+b;
5     printf("%d\n", c);
6 }
7
8 //procedimento
9 int menu(){
10     char ch;
11     printf("Escolha Sim (s) ou Não (n):\n");
12     scanf("%c", &ch);
13     return ch;
14 }
```


Definindo a função: antes da “invocadora”

- Dever ser definida/criada antes de quem invoca a função

```
1 float f1() {
2     float a, b;
3     scanf("%f %f", &a, &b);
4     if(b!=0) return a/b;
5     else return 0;
6 }
7
8 int main() {
9     char x;
10    scanf("%c", &x);
11
12    while(x!='s') {
13        printf("%f\n", f1());
14        scanf(" %c", &x);
15    }
16    return 0;
17 }
```

Definindo a função: depois da “invocadora”

- “Tudo que usa, tem que declarar”;
- Onde declarar?
 - ▶ Antes da função (antes da main) ou em arquivos cabeçalho .h
- Como declarar? Através de seu protótipo:

```
1 #include <stdio.h>
2 //prototipo da funcao soma()
3 //tipo_retorno nome_funcao();
4 float soma(float, float); //ponto e virgula no final
5
6 int main() {
7     printf("%f\n", soma(1.2, 3.0));
8     return 0;
9 }
10 float soma(float a, float b) {
11     return a+b;
12 }
```

Definindo a função: depois da “invocadora”

```
1 //prototipos
2 void imprime();           //procedimento
3 void soma(int a, int b); //void soma(int, int);
4 int quad(int n);         //int quad(int);
5
6 int main() {
7     imprime();
8     quad(10);
9
10    return 0;
11 }
12
13 int quad(int n) {
14     return n*n;
15 }
16
17 void imprime() {
18     printf("Ola mundo!\n");
19 }
20
21 void soma(int a, int b) {
22     int c=a+b;
23     printf("%d\n", c);
24 }
```

Exemplo

Faça uma função que leia, repetidamente até EOF, a quantidade de pessoas que entra e sai, respectivamente, de um elevador e imprima quantas pessoas restaram.

```
1 #include <stdio.h>
2 //declaracao do prototipo
3 void elevador();
4
5 int main() {
6     elevador();
7     return 0;
8 }
9
10 //definicao da funcao elevador
11 void elevador() {
12     int entra, sai, e=0;
13     while( scanf("%d%d", &entra, &sai) != EOF ) {
14         e = e + entra - sai;
15     }
16     printf("%d\n", e);
17 }
```