Estruturas de Dados I

1-Conceitos Fundamentais

Lidio Mauro Lima de Campos
lidio@ufpa.br

Universidade Federal do Pará – UFPA
Faculdade de Computação

Referências

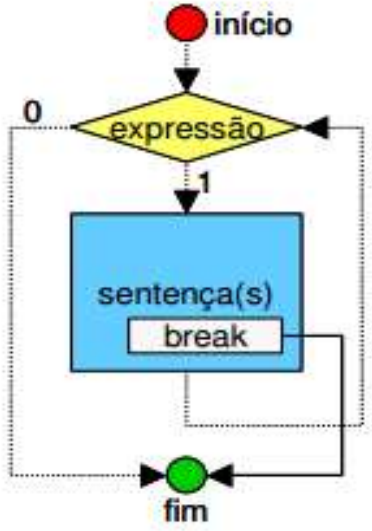
- Waldemar Celes, Renato Cerqueira, José Lucas Rangel, *Introdução a Estruturas de Dados, Editora Campus* (2004)
- Kernigan, B.W. Ritchie, D.M. A Linguagem de Programação C. Campus.
- Ascencio, Ana Fernades G., de Araújo, Graziela Santos. Estruturas de Dados. Prentice Hall, 2010.

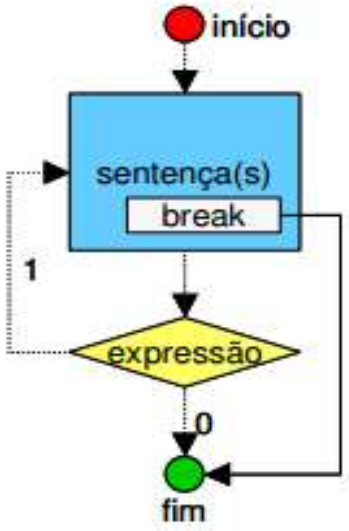
Controle de Fluxo - Interrupção de laços - Comando "break":

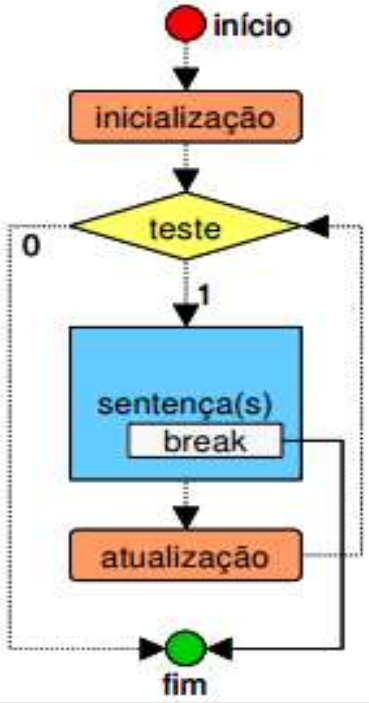
```
while (expressão) {
    sentenças(s);
    if (condição) {
        break;
    }
    sentenças(s);
}
```

```
do { sentenças(s);
    if (condição) {
        break;
    }
    sentenças(s);
} while (expressão);
```

```
for (inicialização;
     teste;
     atualização) {
    sentenças(s);
    if (condição) {
        break;
    }
    sentenças(s);
}
```







Controle de Fluxo

- Interrupção de laços - Comando “break”: termina a execução do comando de laço.

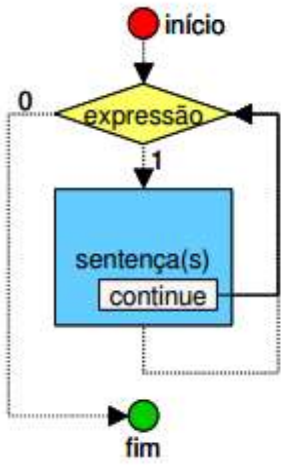
```
#include <stdio.h>
#include <conio.h>
int main (void)
{
  int i;
  for(i=1;i<=10;i++)
  {
     if(i==5)
      break;
     printf("%d",i);
  }
  printf("fim\n");

  return 0;
}
```

Saída=1234fim

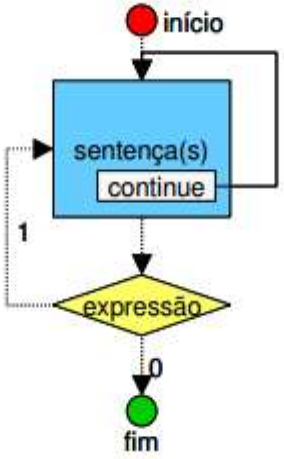
Controle de Fluxo-Interrupção de laços - Comando "continue":

- termina a iteração corrente e passa para a próxima.

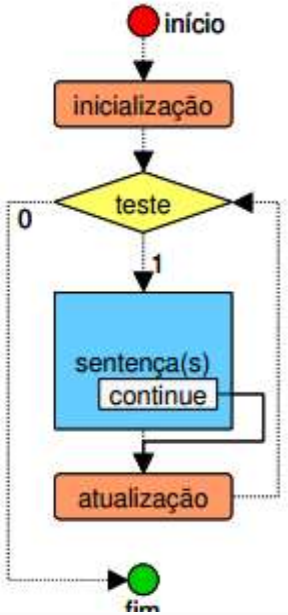
```
while (expressão) {
    comandos(s);
    if (condição) {
        continue;
    }
    comandos(s);
}
```




```
do {
    comandos(s);
    if (condição) {
        continue;
    }
    comandos(s);
} while (expressão);
```




```
for (inicialização;
     teste;
     atualização) {
    comandos(s);
    if (condição) {
        continue;
    }
    comandos(s);
}
```



Controle de Fluxo

- Construções de Laços
  - Interrupção de laços - Comando “continue”:

```
#include <stdio.h>
#include <conio.h>
int main (void)
{
  int i;
  for(i=1;i<=10;i++)
  {
     if(i==5)
      continue;
     printf("%d",i);
  }
  printf("fim\n");
  return 0;
}
```

Saída:1234678910fim

Controle de Fluxo

- Construção de Laços
  - Comando "switch":
  - seleciona uma entre vários casos.
    - ("op" deve ser um inteiro ou caractere).

```
switch ( expr )
{
  case op1: bloco de comandos 1; break;
  case op2: bloco de comandos 2; break;
...
  default:   bloco de comandos default; break;
}
```

Controle de Fluxo

```
/* calculadora de quatro operações */
#include <stdio.h>
int main (void)
{
  float num1, num2;
  char op;
  printf("Digite: numero op numero\n");
  scanf ("%f %c %f", &num1, &op, &num2);
  switch (op)
  {
    case '+':       printf(" = %f\n", num1+num2); break;
    case '-':       printf(" = %f\n", num1-num2);  break;
    case '*':       printf(" = %f\n", num1*num2);  break;
    case '/':       printf(" = %f\n", num1/num2);  break;
    default:        printf("Operador invalido!\n");  break;
  }
  return 0;
}
```

Controle de Fluxo - Exercícios

- 3)Fazer um programa para converter um conjunto de dados que representam temperaturas em graus fahrenheit (a partir de zero incrementadas de 20 ate 300 graus) em graus celsius de acordo com a fórmula:

  Celsius (c)=(5.0/9.0)*(fahr-32).

Fahrenheit	Celsius
0	-17.8
20	-6.7
40	4.4
60	15.6
....	....
300	148.9

Controle de Fluxo - Exercícios

```
main()
{
 int fahr;
 for(fahr=0;fahr<=300;fahr=fahr+20)
 {printf("%4d %6.1f\n",fahr,(5.0/9.0)*(fahr-32));}
 return 0;

}
```

Controle de Fluxo - Exercícios

4)Fazer um programa em C para imprimir os divisores de um número inteiro positivo n. Para um número inteiro n , o programa verifica se cada número de 1 até n é ou não um divisor de n.

5)Fazer um programa que calcule a média de uma quantidade fixa de valores fornecido como entrada, imprimir o valor calculado.

6)Dados uma seqüência de números inteiros não-nulos, calcular a soma dos números pares de cada seqüência.

7)Dados o número *n* de alunos de uma turma de EDI e suas notas da primeira prova, determinar a maior e a menor nota obtidas por essa turma (Nota máxima = 10 e nota mínima = 0).

Controle de Fluxo - Exercícios

4)

```
int main(void)
{
 int numero,divisor,resto;
 printf("Entre com o numero:");
 scanf("%d",&numero);
 divisor=1;
 while(divisor<=numero)
 {
   resto=numero%divisor;
   if(resto==0)
   {
     printf("%d\n",divisor);

   }
   divisor++;
 }
 return 0;
}
```

Controle de Fluxo - Exercícios

5)

```
int main(void)
{
 int numero,contador,quantidade;
 float soma,media;
 printf("Entre com a quantidade de numeros:");
 scanf("%d",&quantidade);
 contador=1;
 while(contador<=quantidade)
 {
   printf("Entre com o numero=");
   scanf("%d",&numero);
   soma+=numero;
   contador++;
 }
 media=soma/quantidade;
 printf("\n MEDIA=%f", media);
 return 0;
}
```

```
7)
 int main() {
 int n,        /* guarda o numero de alunos      */
     nota,      /* usada para a leitura das notas */
     contador,  /* numero de notas ja' lidas      */
     notamaior, /* guarda a maior nota            */
     notamenor; /* gurada a menor nota             */

 printf("\n\tCalculo de maior e menor nota de uma turma\n");
 printf("\nDigite o numero de alunos: ");
 scanf("%d", &n);

 /* inicializacoes */
 contador = 0;
 notamaior = 0;
 notamenor = 100;

 while (contador <  n) {
   printf("Digite uma nota (0 a 100): ");
   scanf("%d", &a);
   contador = contador + 1;
   if (notamaior < nota)
     notamaior = nota;
   if (notamenor > nota)
     notamenor = nota;
 }

 printf("A maior nota obtida foi: %d\n", notamaior);
 printf("A menor nota obtida foi: %d\n", notamenor);

  return 0;
}
```

Controle de Fluxo

```
if ( expr ) { bloco de comandos } else { bloco de comandos }

condição ? expressão1 : expressão2;

while (expr) { bloco de comandos }

for ( expr_inicial; expr_booleana; expr_de_incremento) { bloco de comandos }

do { bloco de comandos } while ( expr );

switch ( expr ) {
   case op1: bloco de comandos 1; break;
   case op2: bloco de comandos 2; break;
...
   default:    bloco de comandos default; break;
}
```

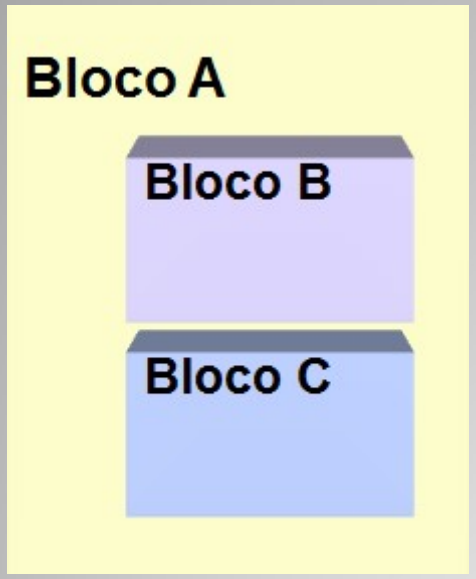
3-Funções

- Tópicos
  - Definição de Funções.
  - Pilha de Execução.
  - Passagem de Parâmetros por Valor e Referência.
  - Ponteiros.
  - Variáveis Globais.
  - Recursividade.

3-Funções

- Tópicos
  - Definição de Funções.
  - Pilha de Execução.
  - Passagem de Parâmetros por Valor e Referência.
  - Ponteiros.
  - Variáveis Globais.
  - Recursividade.

3-Funções



⊙ Propósito:

- **Decomposição de tarefas computacionais extensas em tarefas menores.**

⊙ Definição 1:

- ***Funções*** são **blocos** de programa com início e fim.

C permite aninhamento de blocos.
Delimitadores de blocos: **{ }**

3-Funções

- Programa Básico em C

```
/* Bibliotecas */
#include <stdio.h>
        :
/* Constantes, variáveis e tipos globais */
const float Pi=3.141592654;
        :
/* Programa Principal */
int main()
{
        /* Declaração das variáveis */
        int x,y;
        :
        /* Programa propriamente dito */
        scanf("%d",&x);
        :
        return 0; /* para indicação de execução bem-sucedida */
}
```

3-Funções

- **Definição**:
  - ***Funções*** **são blocos precedidos de um cabeçalho que consiste em:**
    - ***tipo do valor de retorno*** **(se houver algum).**
    - ***nome*** **da função;**
    - ***lista de parâmetros*** **ou** ***argumentos*** **com respectivos tipos;**

3-Funções

```
tipo_de_retorno
    nome_da_função(declaração_de_parâmetros)
    {
    corpo_da_função
    }
```

- **Declaração de parâmetros:**

  tipo nome1, tipo nome2, ... , tipo nomeN

3-Funções

```
/* programa que lê um número e imprime seu fatorial (versão 2) */
#include <stdio.h>
int fat (int n);
int main (void)
{  int n, r;
   printf("Digite um número nao negativo:");
   scanf("%d", &n);
   r = fat(n);
   printf("Fatorial = %d\n", r);
   return 0;
}

/* função para calcular o valor do fatorial */
int fat (int n)
{  int i;
   int f = 1;
   for (i = 1; i <= n; i++)
      f *= i;
   return f;
}
```

"protótipo" da função: deve ser incluído antes da função ser chamada

chamada da função

"main" retorna um inteiro:
0 : execução OK
≠ 0 : execução ¬OK

declaração da função: indica o tipo da saída e o tipo e nome das entradas

retorna o valor da função

3-Funções

- Exercício 8: implemente uma função para calcular o número de arranjos de n elementos, tomados k a k, dado pela fórmula:

$$a = \frac{n!}{(n-k)!}$$

3-Funções

```c
#include<stdio.h>
int fat(int n);
int main(void)
{
int n,k;
float a;
do
{
printf("Entre com dois numeros positivos (n,k):\n");
scanf("%d %d",&n, &k);
}while(!(n>=0 && k>0));

a=fat(n)/fat(n-k);
printf("(%d, %d) = %.2f\n",n,k,a);
system("pause");
return 0;

}
```

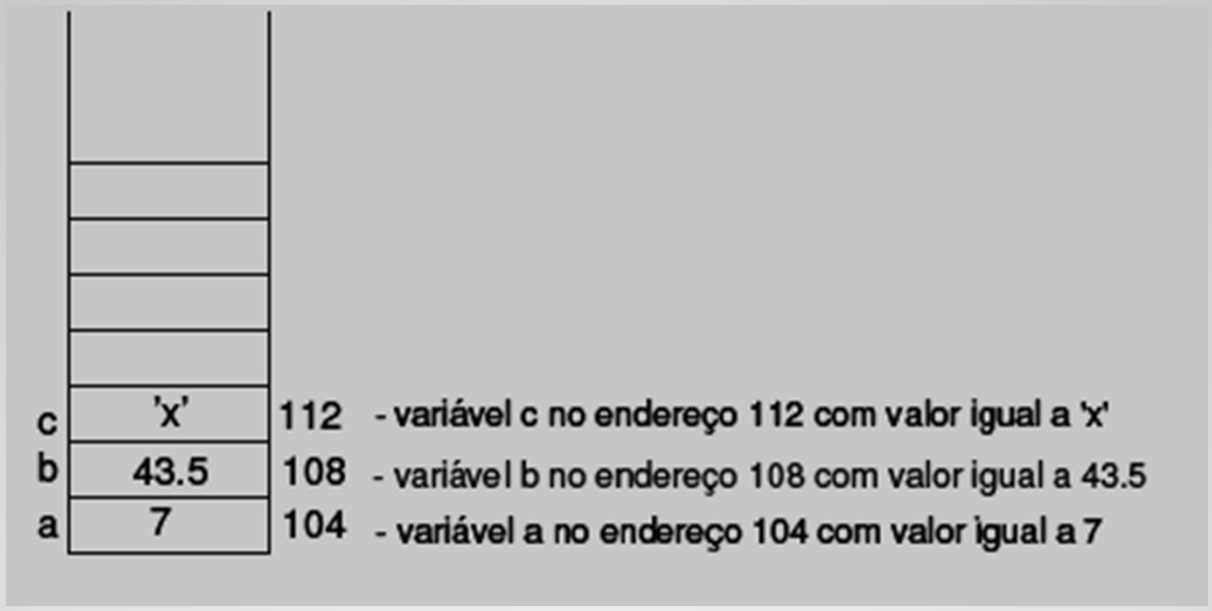
```c
int fat(int n)
{
   int k;
   int f=1;
   for(k=1;k<=n;k++)
   {
      f*=k;
   }
   return f;

}
```

3-Funções

- Pilha de Execução
  - Comunicação entre funções:
    - funções são independentes entre si
    - transferência de dados entre funções:
      - através dos parâmetros e do valor de retorno da função chamada.
- passagem de parâmetros é feita ***por valor***
    - variáveis locais a uma função:
- definidas dentro do corpo da função (incluindo os parâmetros)
    - não existem fora da função.
    - são criadas cada vez que a função é executada.
    - deixam de existir quando a execução da função terminar.

3-Funções

- Comunicação Entre Funções
  - *Pergunta: Como implementar a comunicação entre funções?.*
  - *Resposta: Através de uma pilha.*



3-Funções

- Pilha de Execução
  - implementação da função *fat*
  - simulação da chamada *fat(5)*
    - a variável *n possui valor 0 ao final da execução de fat, mas* o valor de *n no programa principal ainda será 5.*

3-Funções

```
/* programa que lê um numero e imprime seu fatorial (versão 3) */
#include <stdio.h>
int fat (int n);
int main (void)
{  int n = 5;
   int r;
   r = fat ( n );
   printf("Fatorial de %d = %d \n", n, r);
   return 0;
}
int fat (int n)
{  int f = 1;
   while (n != 0) {
      f *= n;
      n--;
   }
   return f;
}
```

declaração das variáveis *n* e *r*, locais à função *main*

declaração das variáveis *n* e *f*, locais à função *fat*

alteração no valor de *n* em *fat* não altera o valor de *n* em *main*

3-Funções

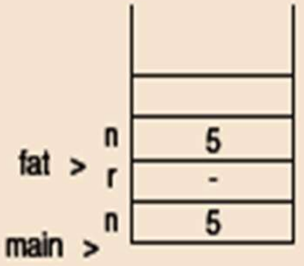
- Pilha de Execução

- Exemplo (cont.): comportamento da pilha de execução

1 - Início do programa: pilha vazia

2 - Declaração das variáveis: n, r

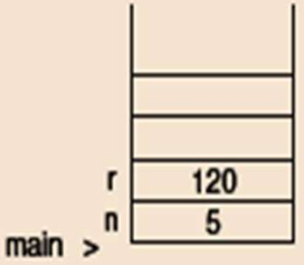
3 – Chamada da função : cópia do parâmetro



4 - Declaração da variável local: f

5 - Final do laço

6 - Retorno da função: desempilha



3-Funções

9)Fazer um Função em C que receba dois valores float e retorne o maior dos números, passar os parâmetros pela função main().

10)Implemente uma função que calcule a distância d entre dois pontos conforme a equação abaixo , a função deve obedecer ao seguinte protótipo: float distancia(float x0, float y0, float x1, float y1).

3-Funções

9)
```
#include <stdio.h>
float maior(float a, float b);
/* Programa principal */
int main(void){
   float x,y, m;
   scanf("%f",&x);
   scanf("%f",&y);
   m = maior(x,y);
   printf("Maior = %f\n", m);
   return 0;
}
/* Corpo das funções */
float maior(float a, float b){
   if (a>b) return a;
   else return b;
}
```

3-Funções

10) #include<stdio.h>
#include<stdlib.h>
#include<math.h>

float distancia(float x0, float y0, float x1, float y1);

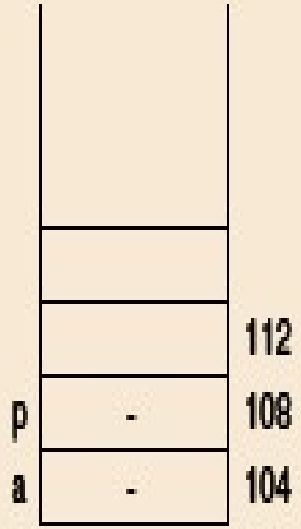
int main(void)
{
  int x0,y0,x1,y1;
  scanf("%d",&x0);
  scanf("%d",&y0);
  scanf("%d",&x1);
  scanf("%d",&y1);
  float dist=distancia(x0,y0,x1,y1);
  printf("DISTANCIA %f",dist);
  system("pause");
  return 0;
}
float distancia(float x0, float y0, float x1, float y1)
{
  // float distancia=sqrt((x0-x1)*(x0-x1)+(y0-y1)*(y0-y1));
    float distancia=sqrt(pow((x0-x1),2)+pow((y0-y1),2));
    return distancia;
}

4-Ponteiros

- Variável do tipo ponteiro:
  - C permite o armazenamento e a manipulação de valores de endereços de memória.
- para cada tipo existente, há um tipo ponteiro que pode armazenar endereços de memória onde existem valores do tipo correspondente armazenados.

4-Ponteiros

```
/*variável inteiro */
 int a;

/*variável ponteiro p/ inteiro */
 int *p;
```

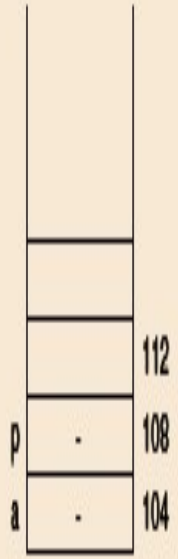


4-Ponteiros

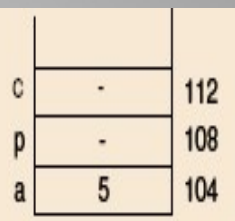
- Operador unário & (“endereço de”):
  - aplicado a variáveis, resulta no endereço da posição de memória reservada para a variável.
- Operador unário * (“conteúdo de”):
  - aplicado a variáveis do tipo ponteiro, acessa o conteúdo do endereço de memória armazenado pela variável ponteiro.

4-Ponteiros

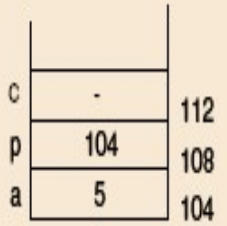
```
/*variável inteiro */
 int a;

/*variável ponteiro p/ inteiro */
 int *p;
```



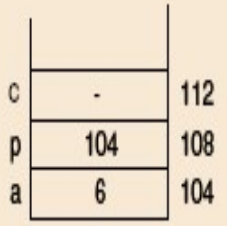

```
/ * a recebe o valor 5 */
a = 5;
```



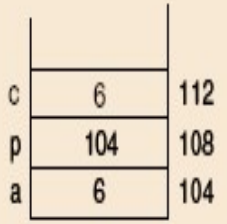

```
/* p recebe o endereço de a
   ou seja, p aponta para a */
p = &a;
```




```
/* posição de memória apontada por p
   recebe 6 */
*p = 6;
```




```
/* c recebe o valor armazenado
   na posição de memória apontada por p */
c = *p;
```



4-Ponteiros

```
int main ( void )
{
  int a;
  int *p;
  p = &a;
  *p = 2;
  printf(" %d ", a);
  return 0;
}
```

- imprime o valor 2

4-Ponteiros

```
int main(void){
   int x=4,y=7;
    int *px,*py;

   printf("x e %d, y e %d.\n",x,y);
   px=&x;
   py=&y;

    *px=*px+10;
    *py=*py+10;

    printf("x e %d, y e %d.\n",x,y);

    getch();
    return 0;
}
```

4-Ponteiros

- Passagem de ponteiros para funções:
  - função g chama função f.
  - f não pode alterar diretamente valores de variáveis de g.
  - porém se g passar para f os valores dos endereços de memória onde as variáveis de g estão armazenadas, f poderá alterar, indiretamente, os valores das variáveis de g.

4-Ponteiros

```
/* função troca */
#include <stdio.h>
void troca (int *px, int *py )
{
   int temp;
   temp = *px;
  *px = *py;
  *py = temp;
}
int main ( void )
{
   int a = 5, b = 7;
   troca(&a, &b); /* passamos os endereços das variáveis */
   printf("%d %d \n", a, b);
   return 0;
}
```

4-Ponteiros

Exercício 11: implementar um programa em c para calcular o fatorial usando funções ponteiros (passagem por referência).

Exercício 12: implementar um programa em c para calcular a soma e o produto de duas variáveis usando a função `void somaprod(int a,int b,int *s,int *p)` como protótipo "a" e "b" são os valores a serem somados e *s e *p são ponteiros que apontam para a soma e produtos respectivamente.

4-Ponteiros

```
#include<stdio.h>
void fat(int *n,int *f);
int main(void)
{
  int n;
  int f=1;
 do
 {
  printf("Entre com um numero positivo (n):\n");
  scanf("%d",&n);
 }while(!(n>=0));

 fat(&n,&f);
 printf("%d",f);

 return 0;

}
```

```
void fat(int *n1,int *f1)
{
  int i;

  for(i=1;i<=*n1;i++)
  {
    *f1=*f1*i;
  }
 }
```

4-Ponteiros

```
# include<stdio.h>
#include<stdlib.h>
void somaprod(int a,int b,int *s,int *p)
{
  *s=a+b;
  *p=a*b;
 }

int main(void)
{
  int s,p,a,b;
  printf("Entre com o valor de a?");
  scanf("%d",&a);
  printf("Entre com o valor de b?");
  scanf("%d",&b);
  somaprod(a,b,&s,&p);
  printf("soma=%d produto=%d",s,p);
  getche();
  return 0;
}
```

4-Ponteiros

- **Exercício 13)**Considere uma função movimento para calcular a posição (s) e a velocidade (v) de uma partícula em um dado instante t, dada a sua aceleração a, a posição so, e a velocidade vo, de acordo com as fórmulas abaixo:

  $s = so+vot+at^2/2$, $v = vo+at$

  Implemente uma função em C com o seguinte protótipo: void movimento (float s0, float v0, float a, float t, float* s, float* v), após isso implemente o código completo em C, que calcula a posição e a velocidade da partícula em um dado instante de tempo.

4-Ponteiros

```
void movimento (float s0, float v0, float a, float t, float*
s, float* v)
{
  *s = s0 + v0 * t + (a * t * t)/2.0;
  *v = v0 + a * t;
}
/* Função principal */
int main (void) {
float s0 = 3.0, v0 = 10.0, a = 5.0, t = 3.0;
float s, v;
movimento(s0, v0, a, t, &s, &v);
printf("%f %f\n", s, v);
getch();
return 0;
}
```

4-Ponteiros

Operador unário & (“endereço de”)

```
p = &a;    /* p aponta para a */
```

Operador unário * (“conteúdo de”)

```
b = *p;    /* b recebe o valor armazenado na posição apontada por p */

*p = c;    /* posição apontada por p recebe o valor da variável c */
```

4-Ponteiros

- **Variáveis Globais**
- Variável global:
  - declarada fora do corpo das funções:
    - visível por todas as funções subsequentes.
    - não deixa de existir quando a execução de uma função termina.
    - existe enquanto o programa estiver sendo executado.
- – utilização de variáveis globais:
  - deve ser feito com critério.
  - pode-se criar um alto grau de interdependência entre as funções.
  - dificulta o entendimento e o reuso do código.

4-Ponteiros

```
#include <stdio.h>
int s, p; /* variáveis globais */
void somaprod (int a, int b)
{
   s = a + b;
   p = a * b;
}

int main (void)
{
   int x, y;
   scanf("%d %d", &x, &y);
   somaprod(x,y);
    printf("Soma = %d produto = %d\n", s, p);
    return 0;
}
```

5-Recursividade

- **Exemplo de definição de Função Recursiva**

$$n! = \begin{cases} 1, \text{ se } n = 0 \\ n \times (n-1)!, \text{ se } n > 0 \end{cases}$$

- /* Função recursiva para cálculo do fatorial */

```
int fat (int n)
{
  if (n==0)
    return 1;
 else
    return n*fat(n-1);
}
```