Programação (CK0226 – 2017.2)

Universidade Federal do Ceará
Departamento de Computação
Prof. Lincoln Souza Rocha
(lincoln@dc.ufc.br)

INTRODUÇÃO À PROGRAMAÇÃO NA LINGUAGEM C

Funções e Ponteiros

Sumário

- Definição de funções
- Pilha de execução
- Ponteiros
- Variáveis globais
- Variáveis estáticas
- Recursividade
- Préprocessador e macros

Declaração de Função

FUNÇÃO

```
<tipo de retorno> <nome da função>(<lista de parâmetros>) {
   <blocos de comando>
   return <valor de retorno>;
}
```

PROCEDIMENTO

```
void <nome da função>(<lista de parâmetros>) {
   <blocos de comando>
}
```

Declaração de Função

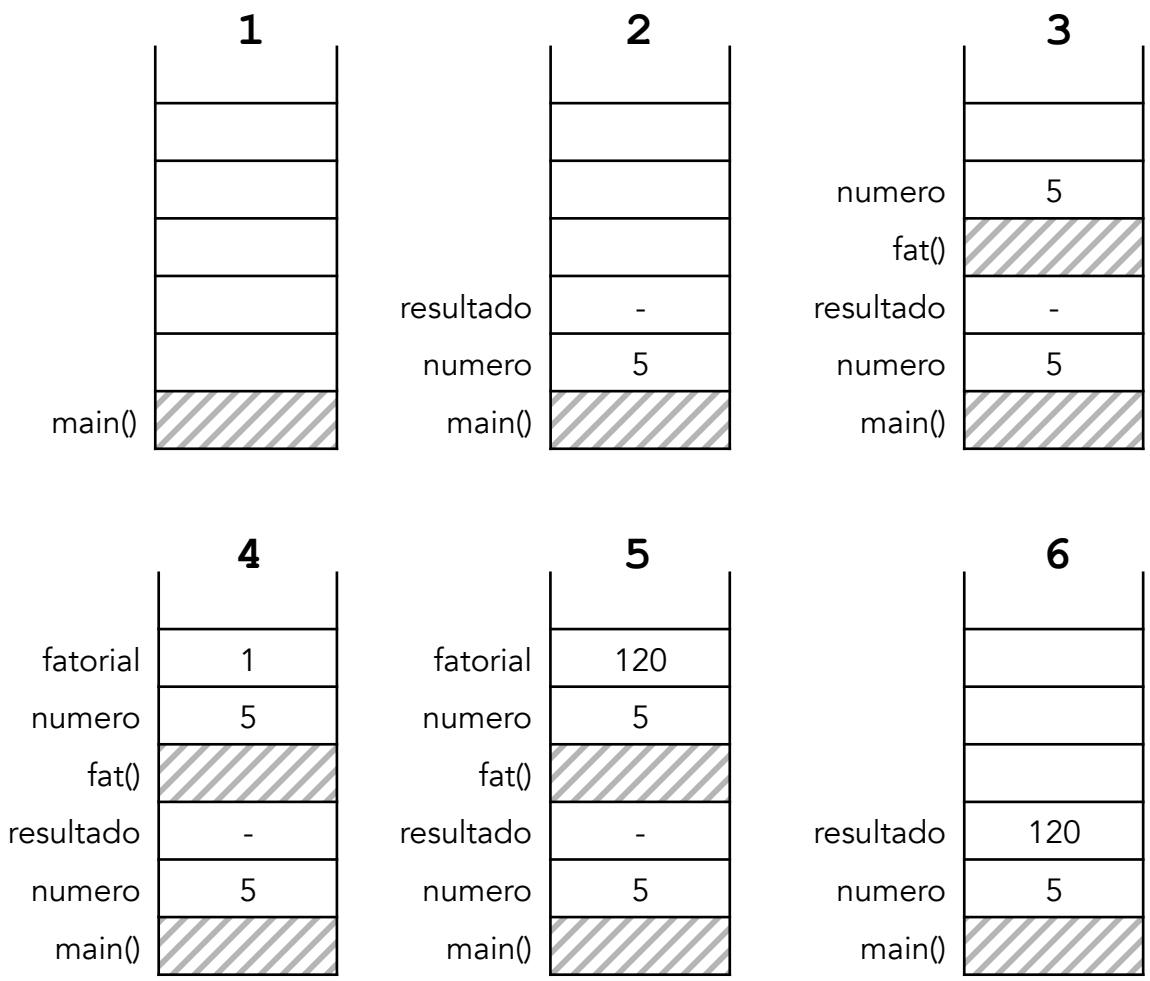
```
#include <stdio.h>

int fat (int numero);                                   protótipo da função

int main (void) {
    int numero, resultado;
    printf(“Digite um número nao negativo:”);
    scanf("%d", &numero);
    resultado = fat(numero);                            chamada da função
    printf("Fatorial = %d\n", resultado);
    return 0;
}

int fat (int numero) {
    int i;
    int fatorial = 1;
    for (i = 1; i <= numero; i++)                       declaração da função
        fatorial *= i;
    return fatorial;
}
```

Pilha de Execução



1 - Início do programa: pilha vazia

2 - Declaração das variáveis: numero e resultado

3 - Chamada da função : cópia do parâmetro

4 - Declaração da variável local: fatorial

5 - Final do laço

6 - Retorno da função: desempilha

Exercício @ Classe

Implemente uma função para calcular o número de arranjos de **n** elementos, tomados **k** a **k**, dado pela fórmula abaixo.

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

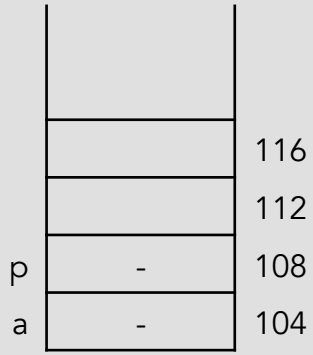
Variáveis do Tipo Ponteiro

A linguagem 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

```
/*variável inteiro */
int a;

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



Variáveis do Tipo Ponteiro

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.

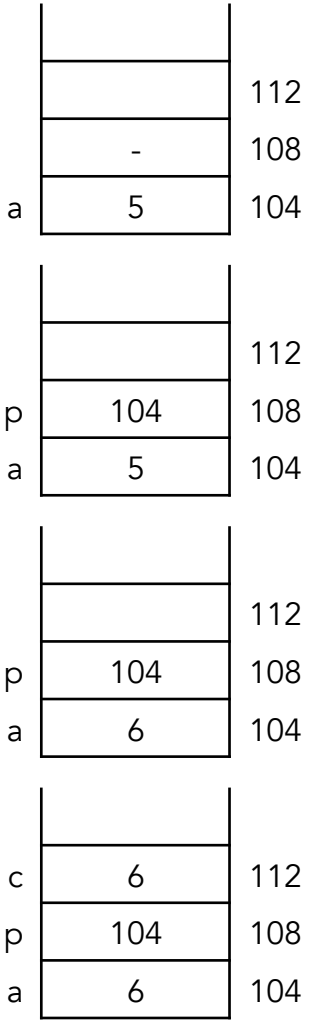
Ponteiros *in Action*!

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

/*variável p recebe o endereço de a */
int *p = &a;

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

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



Ponteiros *in Action*!

```
#include <stdio.h>

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

Imprime 2!

Ponteiros *in Action*!

```
#include <stdio.h>

int main ( void )  {
    int a, b, *p;
    a = 2;
    *p = 3;
    b = a + (*p);
    printf(" %d ", b);
    return 0;
}
```

ERRO!!!

Ponteiros *in Action*!

```
#include <stdio.h>

int main ( void )  {
    int a, b, *p;
    a = 2;
    *p = 3;
    b = a + (*p);
    printf(" %d ", b);
    return 0;
}
```

ERRO!!!

Erro na atribuição "*p = 3". Utiliza a memória apontada por p para armazenar o valor 3, sem que p tivesse sido inicializada, logo armazena 3 num espaço de memória desconhecido.

Ponteiros *in Action*!

```
#include <stdio.h>

int main ( void )  {
    int a, b, c, *p;
    a = 2;
    p = &c
    *p = 3;
    b = a + (*p);
    printf(" %d ", b);
    return 0;
}
```

OK!!!

A atribuição "*p = 3" agora está correta. O ponteiro p aponta para c. A atribuição armazena 3 no espaço de memória reservado para c.

Passagem de Ponteiros pra Funções

A 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() pode alterar, indiretamente, os valores das variáveis de g().

Passagem de Ponteiros pra Funções

```
#include <stdio.h>

void troca (int *px, int *py);

int main ( void ) {
    int a = 5, b = 7;
    troca(&a, &b);
    printf("%d %d \n", a, b);
    return 0;
}

void troca (int *px, int *py) {
     int temp;
     temp = *px;
     *px = *py;
     *py = temp;
}
```

Passagem de Ponteiros pra Funções

1

		120
		116
		112
b	7	108
a	5	104
main()		-

2

		120
py	108	116
px	105	112
troca()		-
b	7	108
a	5	104
main()		-

3

temp	-	120
py	108	116
px	105	112
troca()		-
b	7	108
a	5	104
main()		-

4

temp	5	120
py	108	116
px	104	112
troca()		-
b	7	108
a	5	104
main()		-

5

temp	5	120
py	108	116
px	104	112
troca()		-
b	7	108
a	7	104
main()		-

6

temp	5	120
py	108	116
px	104	112
troca()		-
b	7	108
a	5	104
main()		-

1 - Declaração das variáveis: a e b

2 - Chamada da função : passa endereços

3 Declaração da variável local: temp

4 – A variável temp recebe conteúdo de px

5 - Conteúdo de px recebe conteúdo de py

6 - Conteúdo de py recebe temp

Resumo

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 */

Variáveis Globais

- Declarada fora do corpo das funções
  - Visível por todas as funções subsequentes
- Não é armazenada na pilha de execução
  - 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

Variáveis Globais

```
#include <stdio.h>

void somaprod (int a, int b);

int s, p; /* variáveis globais */

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

void somaprod (int a, int b) {
     s = a + b;
     p = a * b;
}
```

Variáveis Estáticas

- Declarada no corpo de uma função
  - Visível apenas dentro da função em que foi declarada
- Não é armazenada na pilha de execução
  - Armazenada em uma área de memória estática
  - Continua existindo antes ou depois de a função ser executada
- Utilização de variáveis estáticas
  - Quando for necessário recuperar o valor de uma variável atribuída na última vez que a função foi executada

Variáveis Estáticas

```
void imprime ( float a) {
      static int n = 1;
      printf(" %f ", a);
      if ((n % 5) == 0)
         printf(" \n ");
      n++;
}
```

Função para imprimir números reais. Imprime um número por vez, separando-os por espaços em branco e colocando, no máximo, cinco números por linha.

Comentário sobre Variáveis

Variáveis estáticas e variáveis globais são inicializadas com zero, se não forem explicitamente inicializadas.

Variáveis globais estáticas são visíveis para todas as funções subsequentes, mas não podem ser acessadas por funções definidas em outros arquivos.

Funções Recursivas

- Recursão direta
  - A função ***f*** faz uma chamada a se própria
- Recursão indireta
  - A função ***f*** faz uma chamada à função ***g*** que, por sua vez, faz uma chamada à função ***f***

OBS. Quando uma função é chamada recursivamente, cria-se um ambiente local para cada chamada e as variáveis locais de chamadas recursivas são independentes entre si, como se estivéssemos chamando funções diferentes.

Funções Recursivas

- Função fatorial recursiva

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

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

Préprocessador de Macros

Reconhece algumas diretivas pré-estabelecidas e altera o código fonte antes de enviá-lo ao compilador.

Exemplo de Macros

Diretiva de inclusão (#include <nome do arquivo>): o préprocessador substitui o #include pelo corpo do arquivo especificado por <nome do arquivo>.

O texto do arquivo passa a fazer parte do código fonte. Colocando o nome do arquivo entre aspas “nome do arquivo", o préprocessador procura o arquivo primeiro no diretório local e, caso não o encontre, o procura nos diretórios de include especificados para compilação. Por outro lado, se o nome do arquivo é informado entre os sinais de menor e maior (<nome do arquivo>), o préprocessador não procura o arquivo no diretório local (os arquivos da biblioteca padrão de C devem ser incluídos com <>).

Exemplo de Macros

Diretiva de definição (#define <expressão>): é utilizada para definir constantes e expressões parametrizadas. O préprocessador é responsável de substitui as macros definidas pelo código associado.

```
#define PI 3.14159F
float area (float r) {
    float a = PI * r * r; /* PI é substituído por 3.14159F */
    return a;
}
```

```
#define MAX(a,b) ((a) > (b) ? (a) : (b))
...
v = 4.5;
c = MAX(v, 3.0); /* MAX é substituído por ((v) > (3.0) ? (v) : (3.0)) */
```

Programação (CK0226 – 2017.2)

Universidade Federal do Ceará
Departamento de Computação
Prof. Lincoln Souza Rocha
(lincoln@dc.ufc.br)