Linguagem C: Subprogramação

Prof. Paulo R. S. L. Coelho
paulo@facom.ufu.br

Faculdade de Computação
Universidade Federal de Uberlândia

Organização

Organização

Definição

- Um programa escrito em uma linguagem de programação pode ser organizado em um ou mais módulos.
- Um deles, denominado **programa principal**, obrigatório em todos os programas, é aquele pelo qual começa a execução.
- Os eventuais módulos auxiliares são chamados **subprogramas**.
- Durante sua execução, o programa principal poderá solicitar o trabalho de um deles fazendo uma **chamada** desse subprograma.

Argumentos e Parâmetros

- Quando um módulo de um programa faz uma chamada de subprograma, ele pode especificar os dados (variáveis) sobre os quais essa chamada deve atuar.
- Estes dados são chamados **argumentos de chamada**. Por exemplo, na atribuição:
  `x = b*sin(a) - a*sin(b) + (a + b)*sin(a + b);`
  `a`, `b`, e `a+b` são argumentos do subprograma `sin`.

Natureza dos subprogramas

- Um subprograma recebe geralmente dois nomes, dependendo de sua natureza: **função** e **procedimento**.
- É considerado **função** quando produzir um valor, emitindo-o (retornando-o) explicitamente ao módulo que o chamou. Esta emissão é realizada através do comando **return**. Normalmente uma chamada de função aparece em expressões:

```
x = pow(c, d) / (a + fat(b));
```

- Um subprograma é considerado um **procedimento** quando sua principal finalidade é a execução de uma tarefa relacionada com eventuais argumentos, não emitindo (retornando) nenhum valor ao módulo que o chamou. Por exemplo:

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

Subprogramas em C I

- Na linguagem C, todos os módulos de um programa são denominados **função**, sendo que aquela que corresponde ao programa principal tem o nome de **main** e tem presença obrigatória em qualquer programa.
- A organização dos módulos de um programa em C segue os seguinte modelo:

Declarações Globais
Funções Auxiliares
Função Main

- A forma geral de uma função na linguagem C é a seguinte:

Subprogramas em C II

```
Tipo Nome (Lista de Parâmetros) {
  Declarações;
  Comandos;
}
```

- O fluxo de execução retorna de uma função ao módulo que a chamou de duas maneiras: **retorno natural** ou **retorno explícito**.
- No retorno natural, isso ocorre após a execução do último comando do corpo da função.
- No retorno explícito, isso ocorre através do comando **return**.

Exemplo

- Considere a seguinte função para cálculo do fatorial:

```
#include <stdio.h>

int fatorial(int x) {
  int i; fat = 1;
  for (i = 2; i <= x; i++) {
    fat *= i;
  return fat;
}

int main() {
  int n, resp;
  printf("entre número inteiro maior positivo: ");
  scanf("%d", &n);
  resp = fatorial(n);
  printf("O fatorial de %d eh %d\n", n, resp);
  return 0;
}
```

Organização

Declarações locais e globais

- **Declaração local** a uma função é aquela feita no corpo dessa função. Dessa forma, ao ser referenciado, um identificador só é reconhecido pelo compilador se a referência for feita na mesma função de sua declaração.
- O trecho do programa em que um identificador é reconhecido é chamado de seu **escopo de validade**.
- **Declaração global** é feita fora do corpo de qualquer função. O escopo de validade de um identificador declarado globalmente abrange todo o trecho de programa após sua declaração.

Exemplo

- Considere a seguinte programa:

```
#include <stdio.h>
int a = 33;
void sss() {
  int b = 88;
  printf("sss: a = %d, b = %d\n", a, b);
}

int main() {
  int a = 77, b = 55;
  printf("main1: a = %d, b = %d\n", a, b);
  sss();
  printf("main2: a = %d, b = %d\n", a, b);
  return 0;
}
```

- Sua saída será:
  main1: a = 77, b = 55
  sss: a = 33, b = 88
  main2: a = 77, b = 55

Organização

Passagem de Argumentos

- A linguagem C apresenta dois importantes modos de passagem de argumentos para as respectivas funções:
  - Passagem por valor;
  - Passagem por referência.
- Na passagem por valor, o argumento é tipicamente uma expressão, sendo seu valor calculado e carregado no parâmetro.
- Na passagem por referência, o argumento deve ser uma variável, sendo que o respectivo parâmetro é alocado coincidindo com o endereço da mesma. Argumento do tipo vetor sempre é passado por referência.

Exemplo

- Considere a seguinte programa:

```
#include <stdio.h>

void ff(int a) {
  a++;
  printf("Durante ff, a = %d\n", a);
}

int main() {
  int a = 5;
  printf("Antes de ff, a = %d\n", a);
  ff(a);
  printf("Depois de ff, a = %d\n", a);
  return 0;
}
```

- Sua saída será:
  Antes de ff: a = 5
  Durante ff: a = 6
  Depois de ff: a = 5
- O valor da variável `a` da função `main` é copiado para a variável `a` local à função `ff()`.

Exemplo

- Considere a seguinte programa:

```
#include <stdio.h>

void trocar(int *p, int *q) {
  int aux;
  aux = *p; *p = *q; *q = aux;
}

int main() {
  int i = 3, j = 8;
  printf("Antes de trocar: i= %d; j = %d\n", i, j);
  trocar(&i, &j);
  printf("Depois de trocar: i= %d; j = %d\n", i, j);
  return 0;
}
```

- Sua saída será:
  Antes de trocar: i = 3; j = 8
  Depois de trocar: i = 8; j = 3
- Os endereços das variáveis `i` e `j` são passados como argumentos. Dessa forma, `p` e `q` apontam para `i` e `j` e uma alteração em `*p` implica em alteração em `i`.

Organização

Variáveis Indexadas

- Em C, a passagem de uma variável indexada como argumento é sempre por referência.
- Assim, caso algum elemento indexado do parâmetro seja alterado, o elemento do argumento sofre a referida alteração.
- Outro ponto importante é que quando se deseja produzir uma variável indexada dentro da função e retorná-la ao módulo que a chamou, deve-se usar ponteiros.
- Considere o exemplo:

```
int *NovoVetor(int B[]) {
  int i, *C;
  C = (int *) malloc(10*sizeof(int));
  for (i = 0; i < 10; i++)
    C[i] = B[i] + 4;
  return C;
}
```

Estruturas

- Em C, estruturas podem ser passadas como parâmetros, passadas como argumentos, e seus valores podem ser retornados de funções.
- Na passagem como argumento e no retorno de uma função, há uma cópia de toda a estrutura, de um módulo para outro.
- Considere o exemplo:

```
typedef struct matriz matriz;
struct matriz { int nlin; int ncol; int elem[10][10]; };

matriz NovaMatriz(matriz B) {
  int i,j;
  matriz C;
  for (i = 0; i < B.nlin; i++)
    for (j = 0; j < B.ncol; j++)
      C.elem[i][j] = B.elem[i][j] + 4;
  C.nlin = B.nlin;
  C.ncol = B.ncol;
  return C;
}
```

Organização

Exercícios I

1. Faça um programa que leia dois vetores A e B, 4 x 4, e passe-os como parâmetros para uma função chamada `soma`, a qual colocará no vetor A o valor da soma. Em seguida, imprima A e B.
2. Faça um programa que calcule se uma matriz é simétrica horizontalmente. Esse cálculo deve ser feito por uma função denominada `verificaSimetria`, a qual recebe a matriz e suas dimensões como parâmetros e retorna 1 se ela for simétrica e 0, caso contrário.

Exercícios II

3. Um polinômio **P(x)** de grau **n** dado pela fórmula: $P(x) = A_0 + A_1x^1 + A_2x^2 + ... + A_nx^n$ pode ser armazenado em uma estrutura definida pela seguinte declaração:

```
typedef struct polin polin;
struct polin {
  float coef[10];
  int grau;
};
```

onde `coef` é um vetor com os coeficientes e `grau` armazena o grau do polinômio.
Fazer uma função que tenha como parâmetros um polinômio do tipo `polin` e uma variável `x` do tipo `float` e que calcule o valor de P(x).