Dividir para Conquistar

Material da Prof. Ana Eliza

Modularização

- Situação:
  - Necessidade de resolução de problemas cada vez maiores e/ou mais complexos.
- Solução:
  - Dividir o problema grande e/ou complexo em problemas menores e mais simples;
  - Encontrar a solução para os problemas menores;
  - A solução do problema original é obtida através da composição das soluções dos problemas menores.

Modularização

Sub-Programa

- Utilização:
  - A solução dos problemas mais simples é implementada utilizando sub-programas.
  - Sub-programas podem ser vistos como blocos de construção com os quais montamos um programa.
  - Esta técnica de construção de programas é chamada de **MODULARIZAÇÃO**.

Modularização

Bloco

- Definição:
  - Um bloco consiste em um conjunto de declarações e um conjunto de comandos delimitados por indicadores de início e fim de bloco.
  - Em C, os delimitadores de bloco são **{** e **}**.
- Exemplo:

```
{
        int a,b,c;
        scanf ("%i",&a);
        scanf ("%i",&b);
        c = a + b;
        printf ("%i",c);
}
```

⇦ **Declarações de variáveis**

Comandos

Modularização

Sub-Programa

- Definição:
  - Um sub-programa é um bloco de programa constituído por um conjunto de declarações e por um conjunto de comandos e identificado por um nome.
- Exemplo:

```
void adicao(){
    int a,b,c;
    scanf ("%i",&a);
    scanf ("%i",&b);
    c = a + b;
    printf ("%i",c);
}
```

⇦ **Declarações de variáveis**

Comandos

Modularização

- **Sub-Programa**
  - Existem dois tipos de sub-programa:
    - **Funções**: Calculam valores.
    - **Procedimentos**: Executam tarefas.

Funções

- **Definição de Função:**
  - A definição de uma função associa um nome a um bloco de declarações e comandos.
  - Os comandos que compõem o bloco da função têm a "*missão*" de calcular um valor que deve ser informado pela função.

Funções

- **Sintaxe de Definição de uma Função:**

```
tipoRetorno nomeFunção (listaParâmetros)
{
    declaraçõesDeVariáveisDaFunção
    listaDeComandosDaFunção
}
```

Onde `listaParâmetros` é:

$tipoParam_1$ $nomeParam_1$,...,$tipoParam_N$ $nomeParam_N$

- **Exemplo:**

```
float soma(float a, float b) {
    float aux;
    aux = a + b;
    return aux;
}
```

Funções

☞Exemplo de Uso de uma Função:

```
#include <stdio.h>
float soma(float a,float b) {
     float aux;
     aux = a + b;
     return aux;
}
int main () {
     float x,y,z;
     printf("Digite X:");
     scanf ("%f", &x);
     printf("Digite Y:");
     scanf ("%f", &y);
     z = soma(x,y);
     printf("Soma: %f",z);
}
```

chamada

retorno

Funções

- **Chamada (execução) de Função:**
  - Uma referência a um nome de função dentro de um programa provoca a execução do bloco de comandos da função.
  - Ao término da execução da função, a execução do programa continua a partir do ponto onde a função foi chamada .

Funções

- **Resultado de uma Função:**
  - O comando `return` é utilizado pela função para retornar um valor calculado.
    - ⇨O valor retornado pela função deve ser "capturado" pelo programa e:
      - Armazenado em uma variável, ou
      - Exibido ao usuário, ou
      - Utilizado em uma expressão para a realização de um cálculo mais complexo.

Funções

- **Exemplo de Uso de uma Função:**

```
#include <stdio.h>

float soma(float a,float b) {
    float aux;
    aux = a + b;
    return aux;
}

int main () {
    float x,y;
    printf("Digite X:");
    scanf ("%f", &x);
    printf("Digite Y:");
    scanf ("%f", &y);
    printf("Soma: %f", soma(x,y));
}
```

Funções

☞ Exemplo de Uso de uma Função:

```
#include <stdio.h>
int fat (int x) {
     int i, f = 1;
     for (i = 2; i <= x; ++i)
          f = f * i;
     return f;
}
int main ( ) {
      int n, k, num;
      printf("Informe o total de elementos do conjunto: ");
      scanf("%i",&n);
      printf("Informe a qtd. elementos em cada combinação: ");
      scanf("%i",&k);
      num = fat (n) / (fat (k) * fat (n - k));
      printf("Eh possivel fazer %i combinações diferentes. \n",num);
}
```

$$C_{N,K} = \frac{N!}{K!(N-K)!}$$

Procedimentos

- **Definição de Procedimento:**
  - A definição de um procedimento associa um nome a um bloco de declarações e comandos.
  - Os comandos que compõem o bloco do procedimento têm a “missão” de executar uma determinada tarefa para o programa.

Procedimentos

Sintaxe de Definição de um Procedimento:

```
void nomeProcedimento(listaParâmetros)
  {
     declaraçõesDeVariáveisDoProcedimento
     listaDeComandosDoProcedimento
  }
```

Onde `listaParâmetros` é:

$tipoParam_1$ $nomeParam_1$,...,$tipoParam_N$ $nomeParam_N$

Exemplo:

```
void soma(float a, float b) {
     float aux;
     aux = a + b;
     printf ("Soma = %f \n",aux);
}
```

Procedimentos

☞Exemplo de Uso de um Procedimento:

```
#include <stdio.h>
void soma(float a,float b) {
     float aux;
     aux = a + b;
     printf ("Soma = %f \n",aux);
}
int main () {
     float x,y;
     printf("Digite X:");
     scanf ("%f", &x);
     printf("Digite Y:");
     scanf ("%f", &y);
     soma(x,y);
}
```

Procedimentos

- **Chamada (Execução) de um Procedimento**:
  - Uma referência a um nome de procedimento dentro de um programa provoca a execução do bloco de comandos do procedimento.
  - Ao término da execução do procedimento, a execução do programa continua a partir do ponto onde este foi chamado.

Procedimentos

- **Exemplo de Uso de um Procedimento**:

```
#include <stdio.h>

void calcularNovoSalario(float s, float p) {
    float aumento, novoSal;
    aumento = s * p/100;
    novoSal = s + aumento;
    printf("Aumento: %f. Novo salario: %f
  \n",aumento,novoSal);
}

int main () {
    float sal, perc;
    printf("Digite o percentual de aumento: ");
    scanf("%f",&perc);
    printf("Digite o salario atual: ");
    scanf("%f",&sal);
    calcularNovoSalario(sal,perc);
}
```

chamada

retorno

Escopo de Variáveis

Definição de Escopo

- ✓ O escopo de uma variável é a parte do código do programa onde a variável é visível e, portanto, pode ser utilizada.
- ✓ Com relação ao escopo, as variáveis se dividem em:
  - Globais
  - Locais

Escopo de Variáveis

- **Variáveis Globais**
  - ✓ São as variáveis declaradas fora dos procedimentos e das funções;
  - ✓ São visíveis e podem ser utilizadas em toda a extensão do programa;
- **Variáveis Locais**
  - ✓ São as variáveis declaradas dentro dos procedimentos e das funções;
  - ✓ São visíveis e podem ser utilizadas **apenas** dentro do sub-programa que as declarou.

Escopo de Variáveis

☞ **Exemplo de Escopo:**

```
#include <stdio.h>
int num1,num2,num3; ⇦ Variáveis globais
int fat (int x) {
    int i, f = 1; ⇦ Variáveis locais a fat
    for (i = 2; i <= x; i++)
       f = f * i;
    return f;
}
int main () {
  printf("Digite o 1° número :");
  scanf ("%f", &num1);
  printf("Digite o 2° número :");
  scanf ("%f", &num2);
  if (num1 >= 0 && num2 >= 0)
  {
     num3 = fat(num1) + fat (num2);
     printf("%i",num3);
  }
}
```

Parâmetros

O Uso Matemático

- Exemplo: $f(x) = x^2 + 2x + 6$

  Se $x = 2$ então $f(x) = f(2) = 2^2 + 2 \times 2 + 6 = 14$

  Se $x = 3$ então $f(x) = f(3) = 3^2 + 2 \times 3 + 6 = 21$

- **x** é o parâmetro da função **f**.

Parâmetros

O Uso Computacional

- Os parâmetros têm por finalidade servir como uma interface (ponto de comunicação) entre os módulos que compõem um programa.
- Desta forma, é possível passar valores para um módulo utilizando os parâmetros.

Parâmetros

O Uso Computacional - Exemplo

```
#include <stdio.h>
float div (int n1, int n2) {
    float aux;
    aux = (float) n1 / (float) n2;
    return aux;
}
int main()
{
     float c;
     int a, b;
     printf("Informe o primeiro numero: ");
     scanf("%i",&a);
     printf("Informe o segundo numero: ");
     scanf("%i",&b);
     c = div(a,b);
     printf("%i dividido por %i = %f \n",a,b,c);
}
```

Parâmetros

- **Parâmetros Formais**
  - Parâmetros formais são as variáveis declaradas no cabeçalho do sub-programa.
- **Parâmetros Reais**
  - Parâmetros reais são as variáveis utilizadas no instante da chamada do sub-programa.

Parâmetros

- **O Uso Matemático**
  - Exemplo: $f(x) = x^2 + 2x + 6$

  (i) Se $x = 2$ então $f(x) = f(2) = 2^2 + 2 \times 2 + 6 = 14$

  (ii) Se $x = 3$ então $f(x) = f(3) = 3^2 + 2 \times 3 + 6 = 21$

  - **x** é o **parâmetro formal** da função **f**
  - **2** é o **parâmetro real** no caso (i), e
  - **3** é o **parâmetro real** no caso (ii).

Parâmetros

O Uso Computacional

```
float div (int n1, int n2) {
    float aux;
    aux = float (n1) / float (n2);
    return aux;
}
void main ( )  {
...  c = div(a,b); ...
}
```

- **n1** e **n2** são os parâmetros formais da função **div**.
- **a** e **b** são os parâmetros reais.

Mecanismos de Passagem de Parâmetros

- Passagem de Parâmetros por Valor
  - Mecanismo unidirecional (entrada)
- Passagem de Parâmetros por Referência
  - Mecanismo bidirecional (entrada / saída)

Mecanismos de Passagem de Parâmetros

- Passagem de Parâmetros por Valor
  - Na passagem de parâmetros por valor, no instante da chamada do sub-programa, o parâmetro formal recebe uma cópia do valor do parâmetro real correspondente.
  - Alterações feitas nos parâmetros formais não refletem nos parâmetros reais correspondentes.
  - A passagem de parâmetros por valor caracteriza-se por ser um mecanismo de entrada de dados para o sub-programa.

Mecanismos de Passagem de Parâmetros

Passagem de Parâmetros por Valor - Exemplo

```
#include <stdio.h>
int fat (int n) {
      int i, f = 1;
      for (i = 2; i <= n; i++)
            f = f * i;
      return f;
}
int main ( ) {
       int total, qtd, num;
       printf("Informe o total de elementos do conjunto: ");
       scanf("%i",&total);
       printf("Informe a qtd. elementos em cada combinação: ");
       scanf("%i",&qtd);
       num = fat (total) / (fat (qtd) * fat (total - qtd));
       printf("Eh possivel fazer %i combinações diferentes. \n",num);
}
```

Mecanismos de Passagem de Parâmetros

- **Passagem de Parâmetros por Referência**
  - A passagem de parâmetros por referência caracteriza-se por ser um mecanismo de comunicação bidirecional com o sub-programa, ou seja, é um mecanismo de entrada e saída de dados.

Mecanismos de Passagem de Parâmetros

- **Passagem de Parâmetros por Referência**
  - Na passagem de parâmetros por referência, no instante da chamada do sub-programa, o parâmetro formal recebe o endereço (posição na memória) do parâmetro real correspondente.

Mecanismos de Passagem de Parâmetros

Passagem de Parâmetros por Referência

- O que caracteriza que um parâmetro está utilizando o mecanismo de passagem por referência é a colocação de um * (asterisco) imediatamente antes do nome do **parâmetro formal**, na declaração do subprograma, indicando que o mesmo é uma variável do tipo **apontador**.
- Na chamada do subprograma, é passado o endereço do **parâmetro real**, colocando-se um & (operador de endereço) antes no nome do parâmetro.

Programa Exemplo

```
#include <stdio.h>
void troca(int *a,int *b) {
    int temp;
    temp = *a;
    *a = *b;
    *b = temp;
}
int main () {
     int x,y;
     printf("Informe o valor de X: ");
     scanf("%i",&x);
     printf("Informe o valor de Y: ");
     scanf("%i",&y);
     troca(&x,&y);
     printf("Novo valor de X: %i \n",x);
     printf("Novo valor de Y: %i \n",y);
}
```