Aulas 17 e 18 - Procedimentos e Funções

Problema

Escreva um programa que lê dois valores n, k e imprime como saída:

1. O número de permutações $P_n = n!$
2. O número de arranjos $A_{n,k} = n!/(n-k)!$
3. O número de combinações $C_{n,k} = n!/(k! * (n-k)!)$

Quantos cálculos de fatorial?!?!

Porque utilizar modularização?

- Evitar que os blocos do programa fiquem grandes demais e, por conseqüência, mais difíceis de ler e entender.
- Separar o programa em partes que possam ser logicamente compreendidos de forma isolada.
- Permitir o reaproveitamento de código.
- Evitar que um trecho de código seja repetido várias vezes dentro de um mesmo programa, minimizando erros e facilitando alterações.

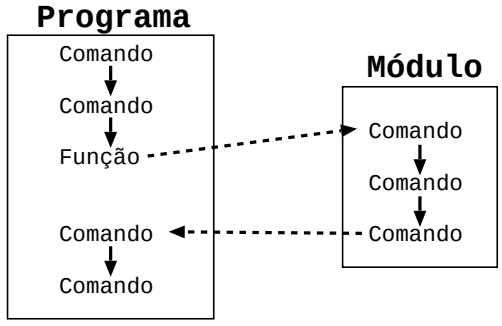
Programação Modular

- dividir para conquistar – resolver um problema complexo dividindo-o em partes menores
- módulos ou subprogramas – conjunto de comandos executados como bloco
- tipos: procedimentos ou funções
- objetivo – programas mais fáceis de ler, entender e manter

Como dividir o programa?

- um módulo deve:
  - ser o mais autosuficiente possível
  - fazer bem uma única tarefa
  - ser visto como uma "caixa-preta"
- apenas as informações necessárias são passadas de um módulo para outro (parâmetros)
- se algum dos subproblemas for ainda muito complexo, dividi-lo em outros subproblemas e assim sucessivamente

Programação Modular



Procedimentos

- módulos que agrupam um conjunto de comandos, que são executados quando o procedimento é chamado
- possuem: um nome; uma lista de parâmetros; variáveis e constantes próprias; bloco de comandos.
- lista de parâmetros (entrada de dados) pode não existir
- parâmetros são declarados e utilizados como se fossem variáveis normais do programa
- a ordem em que os parâmetros aparecem na lista é extremamente importante.

Funções

- procedimentos que retornam um valor
- toda função deve ter um tipo que determina qual será o tipo de seu valor de retorno.
- A expressão contida dentro do comando `return` é chamado de valor de retorno, e corresponde a resposta de uma determinada função.

O tipo `void`

- O tipo `void` é um tipo especial, utilizado principalmente em funções.
- Ele é um tipo que representa o "nada", ou seja, uma variável desse tipo armazena conteúdo indeterminado, e uma função desse tipo retorna um conteúdo indeterminado.
- Este tipo é utilizado para indicar que uma função não retorna nenhum valor.

Procedimentos e Funções

- Procedimentos – estruturas que agrupam um conjunto de comandos, que são executados quando o procedimento é chamado. Ex.: `scanf("%d", &x);`
- Funções – procedimentos que retornam um único valor ao final de sua execução. Ex.: `x = pow(2,3);`

Em C, não se usa a nomenclatura procedimentos.

Exemplos de Funções

- `int f1() {return (1);}`
- `int f2() {}` // erro
- `void f3() {}`
- `void f1() {return (1);}` //erro
- `int f1() {return;}`

Declaração

- Sintaxe:

```
<tipo> nome (<lista_parametros>) {
   comandos;
   return (valor);
}
```

- `<lista_parametros>`:
  `<tipo> param1, <tipo> param2, ...`

Exemplos de Funções

-
```
int soma (int a, int b) {
   return (a + b);
}
```

-
```
void imprime (int x) {
   printf("Resultado %d", x);
}
```

Matematicamente

Uma função é uma operação que recebe um ou mais valores (argumentos ou parâmetros) e produz um resultado.

- $f(x) = \frac{x}{1+x^2}$

  $f(3) = \frac{3}{1+9} = \frac{3}{10} = 0,3$

- $f(x,y) = \frac{x-y}{\sqrt{x}-\sqrt{y}}$

Invocando uma função

- atribuindo o seu valor a uma variável:
  `x = soma(4, 2);`
- o resultado da chamada de uma função é uma expressão e pode ser usada em qualquer lugar que aceite uma expressão:
  `printf("Soma: %d\n", soma(a, b));`
- Outro exemplo:
  `imprime(soma(a, b));`

Invocando um módulo

- O número de argumentos e os tipos de cada argumento devem ser compatíveis com os parâmetros especificados no cabeçalho do módulo. O valor destas expressões são copiados para os parâmetros do módulo.
- Os argumentos não necessariamente possuem os mesmos nomes que os parâmetros que o módulo espera.
- Cada argumento pode ser qualquer expressão válida: uma constante, uma variável, uma expr. aritmética, ou uma outra função que retorne algum valor.
- O valor dos argumentos não é afetado por alterações nos parâmetros dentro do módulo.

Onde declarar funções?

1. Antes do `main`
2. Depois do `main`: protótipo obrigatório
   Protótipo da função:
   - informa ao compilador sobre a função
   - idêntico ao cabeçalho, substituindo as chaves e seu conteúdo por ponto-e-vírgula.
   - deve ser descrito antes de qualquer chamada à essa função
   - conveniente agrupar todos os protótipos

Exemplos de invocação

Para a função:

```
void f1 (int a, float b, char c) {}
```

Podemos ter as seguintes chamadas:

- `f1(10, 2.5, 'a');`
- `int x, float y, char z;`
  `f1(x, y, z);`
- `f1(20, y, 'z');`

Antes do `main`

```
#include <stdio.h>

int soma (int op1, int op2) {
  return (op1 + op2);
}

int main () {
  int a = 0, b = 5;
  printf ("%d\n", soma (a, b));
  return 0;
}
```

Depois do `main`: protótipo

```
#include <stdio.h>

int soma (int op1, int op2);

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

int soma (int op1, int op2) {
  return (op1 + op2);
}
```

Variável global

```
#include <stdio.h>
int global;
void imprime_global () {
  printf ("%d\n", global);
}
void le_global () {
  printf ("Digite o valor da variÃ¡vel global: ");
  scanf ("%d", &global);
}

main () {
  le_global();
  imprime_global();
  printf ("%d\n", global);
}
```

Imprime

```
#include <stdio.h>

void imprime (int numero) {
  printf ("Numero %d\n", numero);
}

int main () {
  int a = 6;
  imprime (10);
  imprime (a);

  return 0;
}
```

Cuidado com variáveis globais

- visível por qualquer módulo
- qualquer módulo pode alterá-las
- ocupa espaço de memória durante toda a execução do programa
- por enquanto **NÂO** vamos utilizá-las

Variáveis locais e globais

- Uma variável é chamada **local** se foi declarada dentro de um módulo. Nesse caso, ela existe somente dentro daquele módulo e após o término da execução do mesmo, a variável deixa de existir.
- Uma variável é chamada **global** se for declarada fora de qualquer módulo. Essa variável é visível por todos os módulos, qualquer módulo pode alterá-la e ela existe durante toda a execução do programa.

Exercícios

1. Escreva uma função calcula e imprime a média de três valores em ponto flutuante.
2. Escreva uma função calcula a média de três valores em ponto flutuante.
3. Escreva uma função que imprime o maior de dois números.
4. Escreva uma função que retorna o maior de dois números.

5. Implemente uma função que receba 3 valores inteiros e retorne o maior dentre eles. Esta função deve chamar a função do exercício anterior para executar esta tarefa.

6. Altere a função do exercício anterior de forma que ela simplesmente imprima o seu valor e não retorne nada.

7. Escreva uma função que calcula e imprime o fatorial de um número.

8. Escreva uma função que calcula o fatorial de um número.

9. Escreva um programa que lê dois valores n, k e imprime como saída o número de:
   (a) permutações $P_n = n!$
   (b) arranjos $A_{n,k} = n!/(n-k)!$
   (c) combinações $C_{n,k} = n!/(k! * (n-k)!)$