Introdução à Programação

Programação C

Uso de Funções

Prof. Roberto M. de Faria/UASC/UFCG

Conteúdo

- Introdução – Motivação
- Exemplo: número de combinações
- Exemplo: número de combinações com uma função do usuário
- Módulos de Programas
- Módulos de Programas em C
- Biblioteca Padrão C
- Funções
- Benefícios do uso de funções
- Definições de Função

Conteúdo

- Protótipos de Funções
- Arquivos de Cabeçalho
- Chamada de Funções por Valor e por Referência
- Geração de Números Aleatórios
- Classes de Armazenamento
- Regras de Escopo
- Recursividade
- Recursividade vs. Iteração

Introdução - Motivação

- Uma combinação sem repetição, em análise combinatória, é um subconjunto com ***s*** elementos em um conjunto **U**, com ***n*** elementos. Como é um conjunto, não há repetição de membros dentro do conjunto.
- O número de subconjuntos s é dado por

$$C_r^n = \binom{n}{r} = \frac{n!}{r! \cdot (n-r)!}$$

- Um programa para calcular o número de subconjuntos poderia ser

Número de Combinações

```
#include <stdio.h>
#include <locale.h>
int main() {
    int r, n, n_menos_r, fator, , num_subconjuntos;
    long long int n_fatorial, r_fatorial,
                  n_menos_r_fatorial;
    setlocale(LC_ALL, "");
    printf("Calcula o número de subconjuntos s, de r"
           " elementos, numa combinação matemática simples"
           "\npara um conjunto U de n elementos:\n\n");
    printf("Informe o número de elementos do conjunto U(n):
");
    scanf("%d", &n);
    printf("Informe o número de elementos dos subconjuntos
s(r): ");
    scanf("%d", &r);
```

Número de Combinações

```
    n_fatorial = 1;
    for (fator = 1; fator <= n; fator++)
        n_fatorial *= fator;
    r_fatorial = 1;
    for (fator = 1; fator <= r; fator++)
        r_fatorial *= fator;
    n_menos_r = n - r;
    n_menos_r_fatorial = 1;
    for (fator = 1; fator <= n_menos_r; fator++)
        n_menos_r_fatorial *= fator;
    num_subconjuntos = n_fatorial / (r_fatorial *
                       n_menos_r_fatorial);
    printf("\nNúmeros de subconjuntos = %d\n\n",
           num_subconjuntos);
    return 0;
}
```

Número de Combinações - função

```
#include <stdio.h>
long long int fatorial(int); // Protótipo da função fatorial()
int main() {
    int r, n, num_subconjuntos;
    printf("Calcula o número de subconjuntos s, de r elementos, "
           "numa combinação matemática simples\npara um conjunto U"
           " de n elementos:\n\n");
    printf("Informe o número de elementos do conjunto U(n): ");
    scanf("%d", &n);
    printf("Informe o número de elementos dos subconjuntos s(r): ");
    scanf("%d", &r);
    num_subconjuntos = fatorial(n) / (fatorial(r) * fatorial(n -
r));
    printf("\nNúmeros de subconjuntos = %d\n\n", num_subconjuntos);
    return 0;
}
```

Número de Combinações - função

```
/*
  Função para cálculo de um fatorial
*/
long long int fatorial(int numero) {
    long long int fat = 1;
    int fator;

    for (fator = 1; fator <= numero; fator++)
        fat *= fator;
    return fat;
}
```

Módulos de Programas

- Divisão para a conquista - construção de programas a partir de partes ou componentes menores - ***módulos***
- Escreve o módulo uma vez e pode usar inúmeras vezes – possibilidade de reuso
- Cada módulo deve resolver um único problema – facilidade para a solução e maior possibilidade de reuso
- Os módulos podem ser escritos por pessoas diferentes – aumento de produtividade

Módulos de Programas em C

- Funções – módulos em C
- Possibilidade de combinação de funções definidas pelo usuário com funções das bibliotecas nos programas
- Existência de uma vasta gama de funções nas bibliotecas padrão de C
- Um programa é um conjunto de funções interoperantes

Módulos de Programas em C

- Chamadas de Funções
  - Invocação, ativação ou execução de funções
    - Explicitação do nome da função e passagem de argumentos (dados necessários à execução da função)
    - Realização de operações ou manipulações pela função
    - Retorno dos resultados pela função
- Analogia
  - Solicitação de execução de uma tarefa pelo patrão a um empregado
    - Aquisição de informações sobre a tarefa pelo empregado
    - Execução da tarefa
    - Retorno dos resultados
    - Ocultação da informação (patrão não precisa saber como a tarefa foi feita)

Bibliotecas Padrão C

- Execução de tarefas mais comuns aos programas
- Usar funções de uma biblioteca padrão
  - **`#include <nome_da_biblioteca.h>`**
- Formato para a chamada de funções
  - **`nome_da_função(arg_1, …, arg_n)`**
- Uso da vírgula como separador em listas com vários argumentos

Bibliotecas Padrão C

- Exemplos de chamadas já realizadas de funções
  - **`printf("Raiz quadrada = %.2f",`**
    `sqrt(900.0));`
    - Chamada da função **`sqrt()`**, que retorna a raiz quadrada de seu argumento; chamada da função **`printf()`** que mostra uma cadeia de caracteres na tela e retorna um código de execução
  - **`pow(base, expoente)`**
    - Chamada da função **`pow()`**, que recebe valores para base e expoente e retorna a potência correspondente calculada
  - **`scanf("%d %f", &valor1, &valor2);`**
    - Chamada da função **`scanf()`**, que recebe valores digitados no teclado, guarda os valores nos endereços de memória especificados e retorna um código de execução

Bibliotecas Padrão C

- **`system("comando_do_so");`**
  - Chamada da função **`system()`**, que executa um comando do sistema operacional a partir do programa
- **`variavel = getchar();`**
- Chamada da função **`getchar()`**, que recebe um caractere digitado no teclado e retorna o código inteiro do caractere de acordo com a Tabela ASCII

- Os argumentos podem ser constantes, variáveis, expressões ou vazios
- Todas as funções matemáticas retornam valores do tipo ***double***

Funções

- Modularização de um programa
- Todas as variáveis declaradas dentro de funções são ***variáveis locais*** - conhecidas apenas no contexto da função
- ***Argumentos*** – dados necessários para execução da função
- ***Parâmetros*** – variáveis de comunicação que recebem uma cópia dos valores dos argumentos – variáveis locais
- ***Retorno*** – valor final do problema solucionado pela função

Benefícios do uso de funções

- Desenvolvimento gerenciável de programas
- Divisão para conquista – problemas menores são mais fáceis de serem resolvidos
- Uso de funções existentes como blocos para a construção de novos programas
- Reusabilidade de software
- Abstração – ocultação de detalhes internos (p.e.: funções da biblioteca padrão) – quem usa a função não precisa saber como ela foi feita
- Repetição de código evitada
- Pessoas diferentes podem desenvolver as funções de um mesmo programa

Definição de Funções

- Formato de Definição de uma Função

```
tipo nome(lista de parâmetros) {
    declarações e comandos
}
```

- ***Tipo*** – tipo da função – tipo do dado que a função retorna
- ***Nome*** – nome da função – qualquer identificador válido (mesmas regras para nomes de variáveis)
- ***Lista de parâmetros*** – declarações das variáveis que receberão os argumentos – um tipo deve ser listado explicitamente para cada parâmetro, caso contrário o parâmetro será considerado do tipo ***int***

Definição de Funções

- ***Declarações e comandos*** – corpo da função (bloco de código)
- Variáveis podem ser declaradas dentro dos blocos
- Funções não podem ser definidas dentro de outras funções

Definição de Funções

- ***Retorno*** – retorno do controle de execução para a função que a chamou
- Quando não há retorno (o tipo da função é **`void`**)

  `return;`

- Se algo for retornado

  `return expressão;`

Definição de Funções

```
// Determinação do máximo de três inteiros

#include <stdio.h>

int maximo(int, int, int); // protótipo da função maximo

void mostra_maximo(int, int, int); // protótipo da função mostra_maximo

int main() {
    int int1, int2, int3;


    printf("Digite três inteiros: ");
    scanf("%d%d%d", & int1, & int2, & int3);
    mostra_maximo(int1, int2, int3);
    return 0;
}
```

Definição de Funções

```
// Definição da função maximo
int maximo(int intx, int inty, int intz) {
    int max = intx;
    if (inty > max )
        max = inty;
    if (intz > max)
        max = intz;
    return max;
}
// Definição da função mostra_maximo
void mostra_maximo(int int1, int int2, int int3) {
    printf("\nO maximo eh: %d\n", maximo(int1, int2,
            int3));
    return;
}
```

Definição de funções

```
//  Programa que recebe um inteiro e dá uma mensagem informando se o inteiro
//  é ou não primo
#include <stdio.h>
#include <stdlib.h>
#include <locale.h>
int eh_primo(int);
int conta_divisores(int);
int eh_divisor(int, int);
int main() {
    int numero;
    setlocale(LC_ALL, "");
    printf("Verifica se um número inteiro positivo é ou não primo:\n\n");
    printf("Qual o número a verificar?\n");
    scanf("%d", &numero);
    if (eh_primo(numero))
        printf("\n%d é primo!\n\n", numero);
    else
        printf("\n%d não é primo!\n\n", numero);
    return 0;
}
```

Definição de funções

```
// Verifica se o número recebido é ou não primo
// Retorna verdadeiro (um inteiro diferente de 0) ou falso (0)
int eh_primo(int numero) {
    return conta_divisores(numero) == 2 ? 1 : 0;
}
// Conta os divisores do número recebido
// Retorna o número de divisores
int conta_divisores(int inteiro) {
    int divisor, contador = 0;
    for (divisor = 1; divisor <= inteiro;
         divisor++)
        if (eh_divisor(inteiro, divisor))
            contador++;
    return contador;
}
// Verifica se o primeiro argumento é divisor do segundo
// Retorna verdadeiro (um inteiro diferente de 0) ou falso (0)
int eh_divisor(int inteiro, int divisor) {
    return inteiro % divisor == 0 ? 1 : 0;
}
```