Linguagens de Programação I

Tema # 6

Geração de Números Aleatórios
Introdução a Funções

Susana M. Iglesias

NUMEROS ALEATÓRIOS

- Uma aplicação divertida e popular da programação é sua utilização, em criar jogos e simulações.
- Na maioria dos jogos de azar, o *fator sorte* atrai a maioria das pessoas, o mesmo pode ser introduzido em aplicações em C utilizando a função **`rand()`** para gerar números aleatórios.
- **`i = rand()`**
- a função **`rand()`** gera um inteiro entre `0` e **`RAND_MAX`**,

NUMEROS ALEATÓRIOS

- **`RAND_MAX`** é uma constante simbólica definida no arquivo de cabeçalho **`<stdlib.h>`**
- na maioria dos compiladores **`RAND_MAX = 32767`**,
- em muitas aplicações o conjunto de valores gerados com **`rand()`**, é diferente do necessário para uma determinada aplicação
- lançamento de uma moeda (0 ou 1),
- jogo de dados (1 a 6),

NUMEROS ALEATÓRIOS

- em tais casos é necessário fazer um ajuste de escala ou deslocamento da escala,
- para gerar números no intervalo [a, b] utilize a expressão:
- `i = a + rand() % (b-a+1)`
- a primeiro número do intervalo desejado,
- b último número do intervalo

NUMEROS ALEATÓRIOS

- Exemplo: *Crie um programa que simule o lançamento de um dado 20 vezes e imprima o valor de cada lançamento.*

```
int main()
{
  int i;

  for(i=1; i<=20; i++){
    printf("%8d", 1 + (rand() % 6));

    if (i%5==0) printf("\n");

  }

  return 0;
}
```

NUMEROS ALEATÓRIOS

- na realidade a função `rand()` gera números pseudo-aleatórios,
- ao chamar `rand()` repetidamente produz números aparentemente aleatórios, a mesma seqüência se repete cada vez que o programa for executado.
- para gerar números realmente aleatórios a função `srand()` deve ser utilizada,

NUMEROS ALEATÓRIOS

- a função **`srand()`**, utiliza um argumento inteiro sem sinal para ser a *semente* da função **`rand()`**, de forma que seja produzida uma seqüência diferente de números aleatórios cada vez que o programa for executado.
- o protótipo da função **`srand()`** encontrasse em **`<stdlib.h>`**

NUMEROS ALEATÓRIOS

- Exemplo: *Modifique o exemplo anterior para gerar números verdadeiramente aleatórios.*

```
int main()
{
  int i, semente;

  printf("Entre com a semente:");
  scanf("%d", &semente);
  srand(semente);

  for(i=1; i<=20; i++){
    printf("%8d", 1 + (rand() % 6));

    if (i%5==0) printf("\n");
  }
  return 0;
}
```

NUMEROS ALEATÓRIOS

- se desejássemos randomizar sem necessidade de introduzir uma semente cada vez, devemos procurar uma semente dentro do programa.
- Geralmente é utilizado:

```
srand(time(NULL));
```

- a função `time()` retorna o valor do relógio do computador em segundos,
- o protótipo da função `time()` se encontra em `<time.h>`
- **Exercício**: *Modifique o programa do exemplo anterior para randomizar lendo o relógio do sistema.*

INTRODUÇÃO A FUNÇÕES

- A maioria dos programas de computador que resolvem problemas do mundo real são MUITO maiores que os programas vistos neste curso.
- A melhor maneira de desenvolver e manter um programa é construí-lo a partir de pequenas partes ou módulos.
- Na linguagem C estes módulos independentes são chamados de funções.

INTRODUÇÃO A FUNÇÕES

- As funções permitem uma melhor reutilização do código, aplicam o conceito de abstração de processos, e evitam a repetição de código em um programa.
- Existem dois tipos de funções em C:

1. as funções da biblioteca padrão (I/O, matemáticas, manipulação de arquivos, manipulação de caracteres)
2. funções definidas pelo programador.

INTRODUÇÃO A FUNÇÕES

- Ao criar nossas próprias funções devemos considerar:
    1. o protótipo da função,
    2. a definição da função,
    3. os parâmetros da função,
    4. a chamada a função.

INTRODUÇÃO A FUNÇÕES

- O protótipo (ou cabeçalho) da função diz ao compilador:
  1. o tipo de dado retornado pela função
  2. o número de parâmetros que a função espera receber, os tipos dos parâmetros e a ordem na qual os parâmetros são esperados.
- O compilador utiliza os protótipos para validar as chamadas a funções.

INTRODUÇÃO A FUNÇÕES

- Definição de função:

```
tipo_de_retorno nome_função(lista_parâmetros){
    declarações
    instruções
    return value;
}
```

- Funções não podem ser definidas dentro do corpo de outras funções.
- Os parâmetros de uma função são utilizados para enviar informação (dados) a função.

INTRODUÇÃO A FUNÇÕES

- Os parâmetros enviados a uma função são variáveis locais da função.
- As declaradas dentro de uma função são variáveis locais dessa função.
- Uma função não tem acesso a variáveis locais de outra função.
- A chamada a função é utilizada para invocar uma função, uma chamada a função transfere o controle do programa a primeira instrução da função.

FUNÇÕES - EXEMPLOS

```
int cuadrado(int);    /* protótipo da função */

int main()
{
  int i;

  for(i=1; i<=10; i++)
    printf("%d", cuadrado(i));   /* chamada a função */
  printf("\n");

  return 0;
}

/* definição da função */
int cuadrado(int num){
  return num*num
}
```

FUNÇÕES - EXEMPLOS

- **Exemplo**: *Crie uma função para gerar números aleatórios num intervalo [a, b]. Use a função para:*

  a)Imprimir três números entre 1 e 3.

  b)Imprimir um número entre 1 e 6.

  c)Imprimir 10 números entre 3 e 10.

```
#Tres numeros entre 1 e 5
4  5  2
#Um numero entre 1 e 6
4
#Dez numeros entre 3 e 10
7  7  9  8   3  6  6   6   4  10
```

```
int gera_num(int, int);

int main(){
  int a, b, i;

  srand(time(NULL));

  printf("#Tres numeros entre 1 e 5\n");
  for(i=0;i<3;i++)
    printf("%d\t", gera_num(1, 5));
  a = 1; b = 10;
  printf("\n#Um numero entre 1 e 6\n");
  printf("%d", gera_num(a, 6));
  a = 3;
  printf("\n#Dez numeros entre 3 e 10\n");
  for(i=0;i<10;i++)
    printf("%d\t", gera_num(a, b));
  return 0;
}

int gera_num(int ei, int ed){
  return (ei + rand() % (ed-ei+1));
}
```

FUNÇÕES - EXEMPLOS

- **Exemplo**: *Escreva um programa que receba um número inteiro e imprima o seguinte padrão (n=4)*

```
****
****
****
****

****
*  *
*  *
****

*
**
***
****
```

a) Crie uma função para imprimir o quadrado. Crie uma função para imprimir o quadrado vazado.

b) Crie uma função para imprimir o triangulo.

c) Crie outras funções se forem necessárias.

d) Modifique o programa para imprimir os padrões entre n e n+2, isto é, se receber o numero 3 imprima o padrão para 3, 4 e 5.

```c
void prn_quadrado(int);
void prn_quadrado_vaz(int);
void prn_triang(int);
void prn_linha(int);
void prn_linha_vaz(int);

int main(){
  int n, i;
  printf("Digite n:");
  scanf("%d", &n);
  for(i=n; i<n+3; i++){
    prn_quadrado(i);
    prn_quadrado_vaz(i);
    prn_triang(i);
  }
  return 0;
}

void prn_linha(int n){
  int j;
  for(j=0;j<n;j++)
    printf("*");
  printf("\n");
}
```

```c
void prn_linha_vaz(int n){
  int j;
  printf("*");
  for(j=1;j<n-1;j++)
    printf(" ");
  printf("*\n");
}
void prn_quadrado(int n){
  int i;
  for(i=0;i<n;i++)
    prn_linha(n);
  printf("\n");
}
void prn_quadrado_vaz(int n){
  int i;
  prn_linha(n);
  for(i=1;i<n-1;i++)
    prn_linha_vaz(n);
  prn_linha(n);
  printf("\n");
}
void prn_triang(int n){
  int i;
  for(i=0;i<n;i++)
    prn_linha(i+1);
  printf("\n");
}
```