




Disciplina:

Programação Computacional

Prof. Fernando Rodrigues
e-mail: fernandorodrigues@sobral.ufc.br

Aula 15: Programação em C

❖ Conceitos Avançados:

- Configurando a localidade;
 - Geração de valores aleatórios;
 - Passar argumentos na linha de comando;
 - Diretivas de compilação;
 - O Padrão C99;
 - Funções de Manipulação de Strings;
 - Exercícios.
- 

Configurando a localidade

- ▶ Usamos a função *setlocale()* da biblioteca *locale.h*:
 - `char* setlocale(int categoria, const char* local);`
- ▶ onde **categoria** é uma das macros definidas para localidade e **local** é o identificador de localidade do sistema especificado.
- ▶ Os valores possíveis de macro para **categoria** são:

Categoria	Descrição
LC_ALL	Afeta todo o local da linguagem.
LC_COLLATE	Afeta as funções strcoll() e strxfrm() que trabalham com strings.
LC_CTYPE	Afeta as funções que manipulam caracteres.
LC_MONETARY	Afeta a informação de formatação monetária.
LC_NUMERIC	Afeta a formatação numérica da localidade C.
LC_TIME	Afeta a função de formatação de data e hora strftime() .

Configurando a localidade

- ▶ Com relação ao **local**, trata-se de um valor específico do sistema. Porém, dois valores sempre existirão:
 - 1º) “C” : para a localidade mínima da linguagem.
 - 2º) “” : localidade-padrão do sistema.
 - Além destes, “ptb” é usado para Português do Brasil.
- ▶ Para configurar a localidade de todo o ambiente da linguagem para Português do Brasil, fazemos:
 - `setlocale(LC_ALL, “ptb”);` ou
 - `setlocale(LC_ALL, “portuguese”);`
- ▶ Lembrando que a biblioteca *locale.h* precisa ser importada: `#include <locale.h>`

Geração de valores (pseudo-)aleatórios

- ▶ Para geração de valores aleatórios (entre 0 e `RAND_MAX`, onde `RAND_MAX` é uma constante definida na biblioteca *stdlib.h*) usamos a função:
 - `int rand(void)`
- ▶ Por exemplo, para a variável inteira 'z' receber um valor aleatório, devemos fazer:
 - `int z = rand();`
- ▶ Para inicializarmos a semente (seed) de geração de números aleatórios, utilizamos a função:
 - `void srand(unsigned int)`
- ▶ Antes da chamada a função *rand()*.

Geração de valores (pseudo-)aleatórios

- ▶ Para que seja gerada uma “boa semente” randômica, a dica é utilizar a função *time(NULL)* (da biblioteca *time.h*) como parâmetro da função *srand()*, da seguinte forma:

- `srand(time(NULL));`

▶ Ex:

```
#include <time.h>
int main(){
    int i, aleatorio[10];
    srand(time(NULL));
    for(i=0 ; i < 10 ; i++)
        aleatorio[i] = rand();
}
```

```
aleatorio[0] = 13056
aleatorio[1] = 11423
aleatorio[2] = 12394
aleatorio[3] = 7366
aleatorio[4] = 21356
aleatorio[5] = 21367
aleatorio[6] = 25043
aleatorio[7] = 23545
aleatorio[8] = 19399
aleatorio[9] = 14287
```

- ▶ Para escolher a faixa de valores, vamos usar operações matemáticas, principalmente o operador de módulo, também conhecido como “resto da divisão”: %

Geração de valores (pseudo-)aleatórios

- ▶ Para fazer com que um número 'x' receba um valor entre 0 e 9, fazemos:
 - $x = \text{rand()} \% 10;$
- ▶ Para fazer com que um número 'x' receba um valor entre 1 e 10, fazemos:
 - $x = 1 + (\text{rand()} \% 10);$
- ▶ Para fazer com que um número 'x' receba um valor entre 0.00 e 1.00, primeiro geramos números inteiros, entre 0 e 100:
 - $x = \text{rand()} \% 101;$
- ▶ Para ter os valores decimais, dividimos por 100:
 - $x = x/100;$

Passando argumentos na linha de comando

- ▶ A função main pode ser definida de tal maneira que o programa receba parâmetros que foram dados na linha de comando do sistema operacional. Para receber esses parâmetros, a função main adquire a seguinte forma:

```
int main(int argc, char *argv[ ]) {  
    //sequência de comandos }
```

- ▶ Agora a função main recebe dois parâmetros de entrada:
 - int argc: trata-se de um valor inteiro que indica o número de parâmetros com os quais a função main foi chamada na linha de comando.

Passando argumentos na linha de comando

- `char *argv[]`: trata-se de um ponteiro para uma matriz de strings. Cada uma das strings contidas nessa matriz é um dos parâmetros com os quais a função `main` foi chamada na linha de comando.
- Observe que o valor de `argc` é sempre maior ou igual a 1. Esse parâmetro vale 1 se o programa foi chamado sem nenhum parâmetro (o nome do programa é contado como argumento da função), e é somado +1 em `argc` para cada parâmetro passado para o programa.
- Ao todo, existem `argc` strings guardadas em `argv`.

Passando argumentos na linha de comando

Exemplo: parâmetros da linha de comando

```
01  #include <stdio.h>
02  #include <stdlib.h>
03  int main(int argc, char* argv[]){
04      if(argc == 1){
05          printf("Nenhum parametro passado para o programa %s\n",argv[0]);
06      }else{
07          int i;
08          printf("Parametros passados para o programa %s:\n",argv[0]);
09          for(i=1; i<argc; i++)
10              printf("Parametro %d: %s\n",i,argv[i]);
11      }
12      system("pause");
13      return 0;
14  }
```

Diretivas de compilação

- ▶ As diretivas de compilação são instruções incluídas dentro do código-fonte do programa, mas que não são compiladas. Sua função é fazer alterações no código-fonte antes de enviá-lo para o compilador.
- ▶ Um exemplo dessas diretivas de compilação é o comando `#define`, que usamos para declarar uma constante.
 - Basicamente, essa diretiva informa ao compilador que ele deve procurar todas as ocorrências de determinada palavra e substituí-la por outra quando o programa for compilado.

Diretivas de compilação

- ▶ As principais diretivas de compilação são:

Lista de diretivas de compilação			
#include	#define	#undef	#ifdef
#ifndef	#if	#endif	#else
#elif	#line	#error	#pragma

- ▶ A diretiva / comando `#include`:
 - É utilizado para declarar as bibliotecas que serão utilizadas pelo programa. Basicamente, esse comando diz ao pré-processador para tratar o conteúdo de um arquivo especificado como se houvesse sido digitado no programa no ponto em que o comando `#include` aparece.

Diretivas de compilação

- ▶ A diretiva / comando `#define`:
 - essa diretiva informa ao compilador que ele deve procurar todas as ocorrências de determinada expressão e substituí-la por outra quando o programa for compilado.
 - O comando `#define` permite três sintaxes:
`#define nome`
`#define nome_da_constante valor_da_constante`
`#define nome_da_macro(lista_de_parâmetros)`
`expressão`

Diretivas de compilação

- ▶ O primeiro uso do comando `#define` é simplesmente definir um nome que poderá ser testado mais tarde com os comandos de inclusão condicional:

Exemplo: inclusão condicional com `#define`

	Com <code>#define</code>	Sem <code>#define</code>
01	#include <stdio.h>	#include <stdio.h>
02	#include <stdlib.h>	#include <stdlib.h>
03	#define valor	
04	int main(){	int main(){
05	#ifdef valor	#ifdef valor
06	printf("Valor definido\n");	printf("Valor definido\n");
07	#else	#else
08	printf("Valor NAO definido\n");	printf("Valor NAO
09	#endif	definido\n");
10	system("pause");	#endif
11	return 0;	system("pause");
12	}	return 0;
		}

Diretivas de compilação

- ▶ O segundo uso do comando `#define` é para declarar uma constante. Essa diretiva informa ao compilador que ele deve procurar todas as ocorrências de determinada expressão `nome_da_constante` e substituí-la por `valor_da_constante` quando o programa for compilado.

Exemplo: constantes com `#define`

```
01  #include <stdio.h>
02  #include <stdlib.h>
03  #define PI 3.1415
04  int main(){
05      printf("Valor de PI = %f\n",PI);
06      system("pause");
07      return 0;
08  }
```

Diretivas de compilação

- ▶ O terceiro uso do comando `#define` é para declarar funções macros: uma espécie de declaração de função em que são informados o nome e os parâmetros da função como sendo o nome da macro e o trecho de código equivalente a ser

Exemplo: criando uma macro com `#define`

	Com macro	Sem macro
01	#include <stdio.h>	#include <stdio.h>
02	#include <stdlib.h>	#include <stdlib.h>
03	#define maior(x,y) x>y?x:y	
04	int main(){	int main(){
05	int a = 5;	int a = 5;
06	int b = 8;	int b = 8;
07	int c = maior(a,b);	int c = a>b?a:b;
08	printf("Maior valor = %d\n",c);	printf("Maior valor = %d\n",c);
09	system("pause");	system("pause");
10	return 0;	return 0;
11	}	}

O Padrão C99

- ▶ O padrão ISO/IEC 9899:1999, informalmente conhecido como C99, é uma revisão do padrão ANSI de 1989 da linguagem C. Ele foi reconhecido como um novo padrão da linguagem pela ANSI em 2000, e, com ele, uma série de recursos úteis foi incorporada à linguagem C.
- ▶ Características simples, como os comentários iniciados com `//`, o tipo `long long` e a possibilidade de declarar variáveis ao longo do código, como, por exemplo, dentro de um laço “for” (e não apenas no começo do bloco de comandos), são incorporadas à linguagem C com o padrão C99.
- ▶ A maioria dos compiladores C possui suporte para, pelo menos, algum dos novos recursos do padrão C99. É dever do programador se certificar de quais recursos o seu compilador suporta.

Habilitando o Padrão C99

- ▶ Na linha de comando:
 - Digite a chamada para o compilador seguida por “-std=c99” e em seguida o nome do seu código-fonte a ser compilado.
 - Por exemplo, para compilar um programa de nome “meuprograma.c” usando o compilador GCC com o padrão C99 habilitado, digite:
 - `gcc -std=c99 meuprograma.c`

Habilitando o Padrão C99

- ▶ No DEV C++ (válido para a versão 5.11):
 - Abra o menu “Ferramentas → Opções do Compilador”;
 - Aba “Configurações”;
 - Sub-aba “Geração de Código”;
 - Na opção “Padrão da linguagem (-std)”, selecione a opção “ISO C99”.

Detectando a versão do compilador

Exemplo: detectando a versão do compilador

```
01  #include <stdio.h>
02  #include <stdlib.h>
03  int main(){
04      #ifndef __STDC_VERSION__
05          printf("Versao: C89\n");
06      #else
07          #if(__STDC_VERSION__ == 199409L)
08              printf("Versao: C94\n");
09          #endif
10          #if(__STDC_VERSION__ == 199901L)
11              printf("Versao: C99\n");
12          #endif
13      #endif
14      system("pause");
15      return 0;
16  }
```

Operações em string: biblioteca string.h

► Tamanho:

- `size_t strlen(const char * str)`: retorna o número de caracteres da string passada.

► Cópia:

- `char * strcpy(char *, const char *)`: cópia a segunda para a primeira string.
- `char * strncpy(char *, const char *, size_t)`: cópia de `n` caracteres da segunda para a primeira string.
- `int sprintf(char *, const char *, ...)`: grava dados formatados em uma string.

► Concatenação:

- `char * strcat(char *, const char *)`: concatenação de strings.
- `char * strncat(char *, const char *, size_t)`: adiciona “`n`” caracteres de uma string no final de outra string.

Exercícios

1) Escreva um programa que receba via linha de comando dois nomes: nome1 e nome2, respectivamente. Em seguida, concatene os dois nomes na ordem inversa (nome2+nome1) de acordo com os parâmetros passados, e exiba na tela.

2) Crie um programa que defina três constantes: N, M e A (via diretivas de compilação), onde N é o número de valores que deverão ser gerados aleatoriamente no intervalo de 0 até M e A é o nome de um vetor e que armazenará os N valores inteiros aleatórios que serão gerados.

3) Reescreva o programa do item 2 (acima) de tal forma que seja possível exibir caracteres acentuados.

4) Defina os parâmetros do compilador para a versão C99 de tal forma que seja possível se declarar variáveis dentro do cabeçalho (protótipo) do laço “for”. Escreva um programa (do item 2, por exemplo) mostrando tal característica.

FIM