Programação de Computadores I
ICP131
Aula 12

Ronald Souza
Instituto de Computação - UFRJ
ronaldsouza@dcc.ufrj.br

Conteúdo de hoje

Variáveis globais e estáticas;
Argumentos para a main();
Operações com bits

⟶ Conceito de variáveis globais e estáticas;
⟶ Como passar argumentos para a função main
⟶ Apresentar as operações sobre bits em C

Variáveis globais

*Variáveis são **globais** quando declaradas **fora de uma função.***

→ São acessíveis **de qualquer parte do programa**, ou seja, podem ser usadas ou modificadas **dentro de todas as funções** do programa

→ Existem durante **toda a execução do programa**

*Variáveis declaradas dentro da função main() são **variáveis locais** (isto é, só podem ser vistas de dentro dessa função)*

Variáveis locais

*Variáveis são **locais** quando declaradas **dentro de uma função.***

→ Não podem ser usadas ou modificadas por outras funções

→ Somente existem enquanto a função onde foram declaradas estiver sendo executada

Os parâmetros de uma função também são variáveis locais da função

Exemplo de uso de variáveis globais

```
float nota1, nota2; //variaveis globais

void entrada () {
    printf("Digite as notas 1 e 2: ");
    scanf("%f %f", &nota1, &nota2);
}

int main() {
    float media;
    entrada();
    media = (nota1 + nota2) / 2;
    printf("\nMedia do aluno: %.2f\n", media);
    return 0;
}
```

Recomendações gerais

→ Deve-se usar variáveis globais apenas quando absolutamente necessário. Seu mal uso afeta a legibilidade e a manutenção do programa.

→ Também deve-se evitar usar o mesmo nome para variáveis locais e globais.

Variáveis locais estáticas (em C)

→ Acessíveis apenas dentro da função onde foram declaradas

→ ...porém são armazenadas na área de dados global, então **seu valor se mantém entre as chamadas da função.**

→ São declaradas com o atributo **static**.

Exemplo de uso de variáveis locais estáticas

```
float entrada () {
    static int cont = 1;
    float nota;
    printf("\nDigite a nota %d: ", cont);
    scanf("%f", &nota);
    cont++;
    return nota;
}
int main() {
    float media, nota1, nota2;
    nota1 = entrada(); nota2 = entrada();
    media = (nota1 + nota2) / 2;
    printf("\nMedia = %.2f\n", media);
    return 0;
}
```

Parâmetros da função main()

→ A função main também pode receber parâmetros

→ Esses parâmetros devem ser passados para a função quando o programa é executado (linha de comando)

→ Todos os parâmetros são armazenados no formato string

→ O primeiro argumento é sempre o nome do arquivo executável

Parâmetros da função main()

→ A função main também pode receber parâmetros

→ Esses parâmetros devem ser passados para a função quando o programa é executado (linha de comando)

→ **Todos os parâmetros são armazenados no formato string**

→ **O primeiro argumento é sempre o nome do arquivo executável**

Assinatura completa da função main()

```
<tipo> main (int argc, char *argv[])
```

onde: **argc** é o número de argumentos recebidos e **argv** é o vetor contendo cada um dos argumentos, **no formato de string**

Passagem de parâmetros para a main() - Exemplo 1/3

```
//Arquivo “teste.c”:
#include <stdio.h>
int main(int argc, char *argv[]) {

    for(int i = 0; i < argc; i++)
        printf("argv[%d] = %s\n", i, argv[i]);

    return 0;
}
```

Exemplo de chamada:

```
./teste.out Ronald 7.5 8.5
```

Saída após execução:

```
argv[0] = teste.out
argv[1] = Ronald
argv[2] = 7.5
argv[3] = 8.5
```

Passagem de parâmetros para a main() - Exemplo 2/3 - aspas!

//Arquivo “teste.c”:

```
#include <stdio.h>
int main(int argc, char *argv[]) {

    for(int i = 0; i < argc; i++)
        printf("argv[%d] = %s\n", i, argv[i]);

    return 0;
}
```

Exemplo de chamada:

```
./teste.out “Ronald 7.5” 8.5
```

Saída após execução:

```
argv[0] = teste.out
argv[1] = Ronald 7.5
argv[2] = 8.5
```

Exemplo 3/3 - parte I

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char* argv[]) {
    long long int matricula;
    double cra; int ano;
    char nome[100];
    if (argc < 5) {
      puts("Erro. Informe matricula, CRA, ano de ingresso e nome");
      return 1;
    }
    matricula = atoll(argv[1]);
    cra = atof(argv[2]);
    ano = atoi(argv[3]);
    sscanf(argv[4], "%[^\n]s", nome);
    printf(" matricula = %lld\n cra = %0.2f\n ano = %d\n nome =
%s\n", matricula, cra, ano, nome);
}
```

Exemplo 3/3 - parte II

Exemplo de chamada:

```
./teste.out 555555 7.1 2023 "Fulano de Tal"
```

Saída (impressão na tela):

```
matricula = 555555
cra = 7.10
ano = 2023
nome = Fulano de Tal
```

Operações com bits em C

→ Para operações com bits, a linguagem C dispõe de alguns operadores que podem ser usados nos tipos char, int, long e long long.

→ Não podem ser usados em float, double, long double e void

→ A diferença entre estes operadores e os lógicos é que estes operam em **pares de bits** enquanto que os operadores lógicos consideram o valor completo (mais detalhes a seguir)

Operadores de bit

&	“E” binário
^	“XOR” ou-exclusivo
|	“OU” binário
~	“NÃO” binário
>>	deslocamento de bits a direito
<<	deslocamento de bits a esquerda

Deslocamentos à esquerda e à direita

Nos deslocamentos à direita em variáveis unsigned e nos deslocamentos à esquerda, os bits que entram são zeros

Nos deslocamentos à direita em variáveis signed, os bits que entram correspondem ao sinal do número (1= sinal negativo, 0 = sinal positivo)

Deslocamento de bits - Exemplos

Unsigned char x;	X a cada execução	Valor de x
x = 7	00000111	7
x = x<<1	00001110	14
x = x<<3	01110000	112
x = x<<2	11000000	192
x = x>>1	01100000	96
x = x>>2	00011000	24

Deslocamentos à esquerda e à direita

Um deslocamento para a direita é equivalente a uma divisão por 2
Um deslocamento para a esquerda equivale a multiplicar por 2
Assim a = a * 2 e a = a << 1 são equivalentes!

Exemplos:

```
y = x >> 1; //y = x / 2
y = x << 1; //y = x * 2
y = x >> 2; //y = x / 4
y = x << 2; //y = x * 4
```

Por hoje é isso!

Slides baseados no material de Silvana Rossetto.