MC102 - Algoritmos e Progração de Computador

Prof. Alexandre Xavier Falcão

3° Aula: Variáveis simples, atribuições, e operações matemáticas.

1 Constantes

Assim como variáveis, constantes são usadas para armazenar números e caracteres. Porém, não podemos modificar o conteúdo de uma constante durante a execução do programa. Exemplos:

```
#define PI          3.1415926536 /* atribui 3.1415926536 para PI */
#define MSG        "O Conteúdo de a é " /* atribui o texto para MSG */
#define Epsilon     1E-05    /* atribui 0.00001 para Epsilon */

int main()
{
  float dois_PI;

  dois_PI = 2*PI; /* atribui 6.2831853072 para "a" */
  printf("%s %5.2f\n",MSG,dois_PI); /* imprime "O conteúdo de a é 6.28"
       na tela */
}
```

2 Definindo novos tipos

Podemos usar o comando **typedef** para definir novos tipos de variáveis ou abreviar tipos existentes.

```
typedef enum {false,true} bool; /* o tipo bool só armazena 0/false e 1/true */
typedef unsigned char uchar;    /* o tipo uchar é o mesmo que unsigned char */
int main()
{
  bool v1,v2; /* tipo boleano ou variável lógica */
  uchar a=10;

  v1 = true;  /* o mesmo que atribuir 1 para v1 */
  v2 = false; /* o mesmo que atribuir 0 para v2 */
  printf("%d %d %d\n",v1,v2,a); /* imprime na tela 1 0 10 */
}
```

3 Operações lógicas e expressões

Os caracteres "&&", "||", "!" indicam as seguintes operações lógicas: **and**, **or** e **not**, respectivamente. Essas operações quando aplicadas a variáveis lógicas reproduzem os resultados apresentados na Figura 1, dependendo do conteúdo das variáveis.

v1	v2	v1 and v2
true	true	true
true	false	false
false	true	false
false	false	false

v1	v2	v1 or v2
true	true	true
true	false	true
false	true	true
false	false	false

v1	not v1
true	false
false	true

Figura 1: Operações lógicas

```
typedef enum {false,true} bool;
int main()
{
  bool v1,v2,v3;

  v1 = true;
  v2 = false;
  v3 = v1&&v2; /* atribui false para ‘‘v3’’ */
  v3 = v1||v2; /* atribui true para ‘‘v3’’ */
  v3 = !v1;    /* atribui false para ‘‘v3’’ */
  v3 = ((v1&&v2)||(!v2)); /* atribui true para ‘‘v3’’ */
}
```

4 Funções matemáticas e resto da divisão inteira

Note que um programa consiste de uma seqüência de comandos apresentada na **função principal** (*main*). Quando várias tarefas são necessárias, as seqüências de comandos dessas tarefas podem ser agrupadas em outras **funções**. Esta forma de organização dos programas evita confusão na depuração do programa e provê uma melhor apresentação do código fonte. Estas funções podem ser chamadas a partir da função principal e devem retornar o resultado da tarefa correspondente, para que as demais tarefas possam ser executadas. A própria função main deve retornar o valor 0, quando termina com sucesso. Várias funções matemáticas, por exemplo, são disponibilizadas para o programador na linguagem C.

```
#include <math.h> /* Para usar as funções matemáticas precisamos
     incluir suas definições, que estão em math.h. */

#define PI 3.1415926536

int main()
{
  double a,b;
  int c,d,e;

  a = 1.0;
  b = exp(a); /* atribui 2.718282 para b */
  a = 4.0;
  a = pow(a,3.0); /* atribui 64.0 para a */
  b = log10(100); /* atribui 2.000000 para b */
  a = sin(PI/4.0); /* atribui 0.707107 para a */
  c = 5;
  d = 3;
  e = c/d; /* atribui 1 para c - divisão inteira */
  e = c%d; /* atribui 2 para e - resto da divisão inteira */

  return(0);
}
```

Além de incluir as definições de math.h, as funções matemáticas precisam ser compiladas e incorporadas ao programa executável. Isto é feito com o comando "gcc exemplo.c -o exemplo -lm". Outros exemplos de funções matemáticas são: raíz quadrada **sqrt(x)**, cosseno **cos(x)**, arco tangente **atan(x)**, logarítmo Neperiano **ln(x)**, e arredondamento **round(x)**. Essas funções podem ser encontradas em qualquer manual da linguagem C ou através do comando **man** para aqueles que usam linux.

5 Endereço das variáveis

Toda variável x tem um endereço na memória que pode ser acessado com $\&x$. Este endereço é necessário em algumas funções de entrada de dados, que veremos a sequir.