Programação I
PRG29002

Engenharia de Telecomunicações 2ª Fase

Professor: Cleber Jorge Amaral

2016-2

INSTITUTO FEDERAL
SANTA CATARINA

Revisão de estruturas de repetição

- while
- do while
- for
- for aninhados
- goto
- Loops infinitos

Programa C: Um conjunto de funções

- Um programa em C basicamente é um conjunto de funções.
- Uma função pode ser vista como um subprograma para o qual podemos repassar dados de entrada através de parâmetros e receber os resultados através do retorno da função.
- Um programa em C possui pelo menos uma função a “main()” que é a primeira instrução executada pelo programa (pelo menos do ponto de vista do programador). Da mesma forma, a última instrução desta função é a última instrução a ser chamada.
- Um programa normalmente vai apresentar um conjunto de funções

Exemplo

```
func2()
{
    printf("Esta é a função func2()\n");
}
 func1()
{
    printf("Esta é a função func1()\n");
    func2();
}
 main()
{
    printf("Esta é a primeira instrução da função main()\n");
    func1();
    printf("Esta é a última instrução da função main()\n");
}
```

Retorno das funções

- Uma função em C pode retornar um tipo de dado (char, int, double, float,...) ou não retornar nada (void)
- A chamada para retorno de um dado é realizado utilizando a palavra-chave “return”, observando que não se usa operador de atribuição “=”
- Exemplo:
  - return 0;
  - return a*b;
  - return ‘f’;

Parâmetros de funções

- Uma função normalmente resolve um determinado problema para um determinado conjunto de dados e produz uma saída.
- Estes dados podem ser passados como parâmetros e a saída pode ser retornada pela função.
- Os parâmetros são declarados similarmente a declaração de variáveis para uso em um processo, porém ao chamar uma função o código de chamada carrega estes dados na entrada da função que partirá com informações previamente parametrizadas
- Os parâmetros podem ser dos diversos tipos de dados suportados pela linguagem

Tornando a função visível ao main

- O código C roda de forma linear do topo na direção da última linha do arquivo de código
- Uma função para ser visível ao main, que é o código que roda no início do programa, deve ser declarada antes do main (podendo ser implementada ou apenas prototipada)

Vetores

- Vetor é uma representação de um conjunto de dados
- Em C a sintaxe usada para declarar um vetor é similar de declaração de variáveis simples, salvo pela necessidade de definir as dimensões entre colchetes que deve ser constantes inteiras
- Vetores são matrizes unidimensionais
- Declaração:

```
tipo_da_variável nome_da_variável [tamanho];
```

Vetores

- Exemplos de declarações válidas
  - float f[5] ;
  - int v[5] = {2, 51 , 6, 31, 0}; //declara o vetor e inicializa
  - Int n[2][5] = {{32, 64, 27, 18, 95},{12,15,43,17,67}};
  - char cidade[9] = {'S','a','o',’ ’,’J’,’o’,’s’,’é’,‘\0'};

Exemplo1: vetor de char (string)

```
int main()
{
 char s[20];
 printf("Digite uma string: ");
 scanf("%[^\n]s",s); //sem & antes de s
 printf("String digitada: %s",s);
 return 0;
}
```

Exemplo2: string alfabeto

```
#include <stdio.h>
int main()
{
 char letra;
 char alfabeto[27];
 int i = 0;
 for(letra = 'A' ; letra <= 'Z' ; letra =letra+1)
  {
    alfabeto[i] = letra;
    printf("%c ", alfabeto[i]);
    i++;
  }
  alfabeto[26] = 0;
  printf("\nalfabeto: %s\n",alfabeto);
}
```

Operadores aritméticos

- “+” soma (binário) / positivo (unário)
  - Ex1.: i = +1; //Unário para dizer que 1 é positivo
  - Ex2.: i = 2 + 5; //Binário somando 2 números
- “–” subtração (binário) / negativo (unário)
  - Ex3.: j = –5; //Unário para dizer que 5 é negativo
  - Ex4.: j = 8 – j; //Binário subtraindo 2 números
- “*” multiplicação (sempre binário)
  - Ex5.: k = 8 * 5;
- “/” divisão (sempre binário)
  - Ex6.: m = m / 4;
- “%” resto da divisão (sempre binário)
  - Ex7.: n = n % 2;

Operadores relacionais

- “>” maior que
  - Ex1.: if (i > j) printf(“i é maior que j”);
- “>=” maior ou igual que
  - Ex2.: if (i >= j) printf(“i é maior ou igual a j”);
- “<” menor que
  - Ex3.: if (i < j) printf(“i é menor que j”);
- “<=” menor ou igual que
  - Ex4.: if (i <= j) printf(“i é menor ou igual a j”);
- “==” igual a
  - Ex5.: if (i == j) printf(“i é igual a j”);
- “!=” diferente de
  - Ex6.: if (i != j) printf(“i é diferente de j”);

Operadores lógicos

- “&&” lógica E (AND)
  - Ex1.: if ((i > j) && (i > 0))
    printf(“i é maior que j e positivo”);
- “||” lógica OU (OR)
  - Ex2.: if (i > j) || (i == 0)
    printf(“i é maior que j ou é igual a zero”);
- “!” lógica negação (NOT)
  - Ex3.: if !(i > j)
    printf(“i não é maior que j”);

Operadores lógicos de bit a bit

- “&” lógica E (AND)

  Ex1.: char i = 0x01 & 0x0F; //”E” de dois números

  printf("0x%02hhX %d\n", i, i); //0x01 1

- “|” lógica OU (OR)

  Ex2.: char j = 0x01 | 0x0F; //”OU” de dois números

  printf("0x%02hhX %d\n", j, j); //0x0F 15

- “~” lógica complemento (NOT)

  Ex3.: char k = ~0x0F; //Negação de um “signed”

  printf("0x%02hhX %d\n", k, k); //0xF0 -16

  Ex4.: unsigned char q = ~0x0F; //Negação de um “unsigned”

  printf("0x%02hhX %d\n", q, q); //0xF0 240

Operadores lógicos de bit a bit OU exclusivo e de deslocamento

- “^” lógica diferença (EXCLUSIVE OR)

  Ex5.: char m = 0x0F ^ 0x88; //lógica diferença
  printf("0x%02hhX %d\n", m, m); //0x87 -121

- “<<” deslocamento para a esquerda

  Ex6.: char n = 0x02 << 2; //desloca 2 bit esquerda
  printf("0x%02hhX %d\n", n, n); //0x08 8

- “>>” deslocamento para a direita

  Ex7.: char p = 0x02 >> 1; //desloca 1 bit a direita
  printf("0x%02hhX %d\n", p, p); //0x01 1

Operadores de atribuição

- “=” Atribuição simples

  Ex1.: i = 0x0F; //i “recebe” o valor 0x0F

- “+=” e “–=” Soma e subtração e depois atribui

  Ex2.: n = 0x02; //n recebe 0x02

  n += 0x06; //n recebe n + 0x06 == 0x08

- “*=”, “/=” e “%=” Produto, divisão e resto depois atribui

  Ex3.: p = 0x05; //p recebe 0x05

  p %= 2; //p resulta com 1 que é o resto de 5/2

- “<<=” e “>>=”

  Ex4.: p <<= 2; //p tinha 1, rotacionando ficou 0x04

- “&=”, “|=” e “^=”

  Ex5.: p |= 0x05; //p tinha 0x04, com a OU ficou 0x05

Operadores de pré e pós incremento e decremento

- “++variavel” incrementa antes
- “––variavel” decrementa antes

  Ex1.: i = 2;//i começou com 2

  i++; //i agora é == 3

  Ex2.: j = ++i * 4; //resultado é j == 16 e i == 4
- “variavel++” incrementa depois
- “variavel––” decrementa depois

  Ex3.: k = 3;

  m = k++ * 4; //resultado é m == 12 e k == 4

Tabela de precedência

Precedence	Operator	Description	Associativity
1	++ --	Suffix/postfix increment and decrement	Left-to-right
	()	Function call	
	[]	Array subscripting	
	.	Structure and union member access	
	->	Structure and union member access through pointer	
	(*type*){*list*}	Compound literal(C99)	
2	++ --	Prefix increment and decrement	Right-to-left
	+ -	Unary plus and minus	
	! ~	Logical NOT and bitwise NOT	
	(*type*)	Type cast	
	*	Indirection (dereference)	
	&	Address-of	
	sizeof	Size-of	
	_Alignof	Alignment requirement(C11)	
3	* / %	Multiplication, division, and remainder	Left-to-right
4	+ -	Addition and subtraction	
5	<< >>	Bitwise left shift and right shift	
6	< <=	For relational operators < and ≤ respectively	
	> >=	For relational operators > and ≥ respectively	
7	== !=	For relational = and ≠ respectively	
8	&	Bitwise AND	
9	^	Bitwise XOR (exclusive or)	
10	|	Bitwise OR (inclusive or)	
11	&&	Logical AND	
12	||	Logical OR	
13[note 1]	?:	Ternary conditional[note 2]	Right-to-Left
14	=	Simple assignment	
	+= -=	Assignment by sum and difference	
	*= /= %=	Assignment by product, quotient, and remainder	
	<<= >>=	Assignment by bitwise left shift and right shift	
	&= ^= |=	Assignment by bitwise AND, XOR, and OR	
15	,	Comma	Left-to-right

Fonte: http://en.cppreference.com/w/c/language/operator_precedence

Obrigado pela atenção e participação!