FATEC - SP

Faculdade de Tecnologia de São Paulo

Prof° Luiz Paulo Zanetti

E-mail: luizpaulozanetti@hotmail.com

Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas

Disciplina
Linguagem de Programação

Linguagem C – Comandos diversos

Comentários

Na linguagem C, os comentários são delimitados por:

/ e */ ou por //*

Os comentários podem vir em qualquer posição do programa e não apenas em linhas separadas.

Eles também podem começar em uma linha e terminar em outra.

Linguagem C – Comandos diversos

Exemplos de declaração de variável

int i, j, l;

short int si;

unsigned int ui;

long inteiro_grande;

double balanco, lucro, prejuizo;

Linguagem C – Comandos diversos

Constantes hexadecimais e octais

Podem ser declaradas constantes em hexadecimal ou octal conforme o exemplo a seguir:

```
int hex = 0xFF;              /* 255 em decimal */
/* as constantes em hexadecimal devem ser
   precedidas por 0x */
int oct = 011;               /*  9 em decimal */
/* as constantes em octal devem ser
   precedidas por 0 */
```

Linguagem C – Comandos diversos

Constantes strings

Uma string é um conjunto de caracteres entre aspas. Por exemplo, “esta é uma string” é uma string.

Não confundir strings com caracteres, ‘a’ é um carácter enquanto “a” é uma string.

Linguagem C – Comandos diversos

Incremento X Pós-incremento

```
    x ++;
    printf("%d\n", x);       /* exibirá 2*/
    printf("%d %d\n", x++, ++x);/* exibirá 2 e 3 */
    /* neste caso, x só é incrementado depois que o  comando
       é executado enquando y é incrementado antes */
}
```

Linguagem C – Comandos diversos

Operador de atribuição

O operador = é o operador de atribuição. Ao contrário de outras linguagens, C permite que o operador de atribuição seja usado em expressões com outros operadores.

```
int a, b, c;
a = b = c = 1;     /* atribui 1 às 3 variáveis */
((a = 2 * b) > c) /* a = 2 e a comparação resulta em 1 */
```

Linguagem C – Comandos diversos

Conversões de tipos

Quando você mistura constantes e variáveis de tipos diferentes em uma expressão, C as converte para o mesmo tipo.

O compilador C converterá todos os operandos para o tipo do maior operando, uma operação de cada vez, conforme descrito nestas regras de conversão de tipo:

Linguagem C – Comandos diversos

Regra 1

Todo char e short int é convertido para int.

Todo float é convertido para double.

Linguagem C – Comandos diversos

Regra 2

Para todos os pares de operandos, ocorre o seguinte, em sequência: se um dos operandos for um long double, o outro será convertido para long double.

Se um dos operandos for double, o outro será convertido para double.

Se um dos operandos for long, o outro será convertido para long. Se um dos operandos for unsigned, o outro será convertido para unsigned.

Linguagem C – Comandos diversos

Exemplo:

```
main()
{
    int x = 3;
    printf("%f %f\n", (float) x / 2, (float) (x / 2));
/* serão impressos na tela 1.5 e 1.0 pois no primeiro  caso, x é
    convertido para float e depois é dividido, já  no segundo, somente o
    resultado é convertido para  float */
}
```

Linguagem C – Comandos diversos

Type cast

Pode-se forçar o compilador a efetuar determinada conversão utilizando-se um type cast.

Nesse caso, o programador não precisa fazer nada, pois a conversão de dados é feita automaticamente sem que ele precise usar qualquer instrução extra:

```
main
{
    char a = 5;
    int i = a;
}
```

Linguagem C – Comandos diversos

Type cast

No código acima não existe nada demais, certo? O único detalhe é que quando fazemos i = a, essa não é uma atribuição comum, o compilador precisa gerar código extra para converter o char para int.

Esta conversão é feita de maneira implícita e silenciosa pelo compilador porque qualquer valor de char pode ser armazena em um int. Já um int, não pode ter todos os seus valores armazenados em um char.

Se pegarmos como exemplo um compilador 32 bits, onde o char geralmente tem 8 bits, os seus valores vão de -128 a 127, já um int (32 bits) pode possuir valores de -2147483648 a 2147483647.

Linguagem C – Comandos diversos

Register

Sempre que uma variável for declarada do tipo register, o compilador fará o máximo possível para mante-la num dos registradores do microprocessador, acelerando assim o acesso a seu valor. É pratica comum, declarar as variáveis de controle de loop como sendo register.

```
main( )
{
register int count;
for (count=0;count<10;count++)
        {
        ...
        }
return 0;
}
```

Linguagem C – Comandos diversos

Comprimento máximo do campo

Para especificar o comprimento máximo que um campo poderá ter, basta colocar um inteiro entre o sinal % e o comando de formatação.

Os caracteres que sobrarem serão utilizados nas próximas chamadas a scanf().

Caso não queira ler mais do que 20 caracteres na string nome, utilize

```
scanf("%20s", nome);
```

Linguagem C – Comandos diversos

Comando continue

O comando continue funciona de maneira parecida com o comando break.

Porém, em vez de forçar o encerramento, continue força a próxima iteração do loop e pula o código que estiver no meio.

Linguagem C – Comandos diversos

Exemplo:

```
/* programa para imprimir os números pares  entre 0 e 98 */

main()
{
   int x;
        for (x = 0; x < 100; x ++)
        {
        if (x % 2) continue;
             printf("%d ", x);
        }
}
```

Linguagem C – Comandos diversos

Comando return

O comando return possui duas utilidades básicas:

Causar a saída imediata da função na qual ele se encontra, retornando para o código de chamada.

Devolver um valor para a função chamadora.

Linguagem C – Comandos diversos

Exemplo

```
/* retorna 1 se o parâmetro > 10 */
int maior_que_dez(int x)
{
   return (x > 10);
}
```

Linguagem C – Comandos diversos

Argumentos

Quando é necessário passar alguma informação extra para uma função, esta informação será passada através de argumentos.

Para chamar uma função com argumentos, eles devem ser colocados entre parênteses após o identificador da função.

Veja:

puts(“Atenção”);

A string “Atenção” é o argumento passado para a função puts.

Linguagem C – Comandos diversos

Recursividade

A linguagem C permite que as próprias funções se chamem. A esta característica damos o nome de recursividade.

Existem muitos problemas que se tornariam extremamente difíceis de serem implementados sem a recursividade.

Linguagem C – Comandos diversos

Preprocessador

Antes do programa ser compilado, ele é submetido ao preprocessador.

Esta característica é muito útil.

Todos os comandos do preprocessador são iniciados por um sinal #, sendo os dois mais usados:

#define

#include

Linguagem C – Comandos diversos

#define

O comando define serve para definir um identificador e uma string.

O compilador substituirá o identificador pela string toda vez que for encontrado no arquivo- fonte.

O identificador é chamado de nome de macro, e o processo de substituição é chamado de substituição de macro.

#define indentificador string

Linguagem C – Comandos diversos

Exemplo

```
#define mensagem “Isto é um teste.\n”
#define verdadeiro 1
#define falso !verdadeiro
main()
{
   if (verdadeiro)
   printf(mensagem);
   if (falso)
   printf(mensagem);
}
```

Linguagem C – Comandos diversos

#include

Instrui o compilador a incluir um outro arquivo fonte com aquele que contém o comando #include.

O nome do arquivo a ser incluído deve estar entre aspas ou entre o sinal de maior e menor.

Se o nome do arquivo for colocado entre aspas, o compilador procurará pelo arquivo na seguinte sequência: diretório atual, diretórios configurados no compilador e diretórios padrões.

Caso o nome do arquivo esteja entre < >, não será procurado no diretório atual.

Linguagem C – Comandos diversos

Usuários

Em C podem ser criados diferentes dados personalizados, entre eles:

estrutura

campo de bit

enumeração

O uso de tipos definido pelo usuário facilita a programação e dá maior poder ao programador.

Linguagem C – Comandos diversos

Estruturas

Em C, uma estrutura é uma coleção de variáveis que são referenciadas pelo mesmo nome.

É uma forma conveniente de manter juntas informações relacionadas.

Forma geral:

```
struct nome_estrutura
{
        tipo1 var1;
        tipo2 var2;
}
var_estrutura;
```

Linguagem C – Comandos diversos

Campos de bit

C possui metodos para acessar somente um bit dentro de um byte. Isto é útil para:

Economizar memória declarando várias variáveis booleanas num só byte.

Comunicar com dispositivos que transmite informação diversas codificadas em bytes.

Rotinas de codificação que precisam acessar bits dos bytes.

Linguagem C – Comandos diversos

Como declarar campos de bit

Os campos de bit só podem ser declarados dentro de estruturas.

Forma geral:

```
struct nome_estrutura
{
   tipo1 var1 : comprimento;
   tipo2 var2 : comprimento;
} nome_var ;
```

Os tipos podem ser: int, signed e unsigned.

Quando o tamanho é 1, ele deve ser obrigatoriamente unsigned.

Linguagem C – Comandos diversos

Exemplo

```
struct dispositivo
{
unsigned ativo : 1;
unsigned pronto : 1;
unsignet erro : 1;
unsigned : 2;
unsigned ultimo_erro : 3;
}
```

Linguagem C – Comandos diversos

Enumerações

Enumeração é um conjunto de constantes inteiras com nome e e especifica todos os valores legais que uma variável daquele tipo pode ter.

Para declarar:

```
enum nome_tipo { lista de constantes }
   nome_var;
```

Todas as constantes receberão valores inteiros a começar por zero, a não ser que seja especificado o contrário.

Linguagem C – Comandos diversos

Exemplo

```
enum tamanhos {pequeno, medio, grande =  5} tamanho;
main()
{
tamanho = pequeno;
printf("%d", tamanho);
tamanho = medio;
printf("%d", tamanho);
tamanho = grande;
printf("%d", tamanho);
}
```

Linguagem C – Comandos diversos

Operadores avançados

C possui vários operadores especiais que aumentam em muito sua força e flexibilidade, especialmente na programação a nível de sistema.

Linguagem C – Comandos diversos

Operadores bit a bit

Como C foi projetada para substituir a linguagem assembly na maioria das tarefas de programação, ela possui um completo arsenal de operadores bit a bit.

Os operadores bit a bit só podem ser usados nos tipos char e int.

Linguagem C – Comandos diversos

Operador vírgula

O operador vírgula é usado para juntar várias expressões. O compilador sempre avalia o lado esquerdo da vírgula como void. Assim, a expressão do lado direito ficará sendo o valor de toda expressão separada por vírgula.

Por exemplo:

```
x = (y = 3, y + 1);
```

Atribuirá 3 a y e 4 a x.

Linguagem C – Comandos diversos

Funções comuns

A biblioteca padrão de funções da linguagem C é muito ampla.

Os programas em C fazem uso intenso das funções da biblioteca.

Os programadores iniciantes tendem a reescrever funções já existentes nas bibliotecas.

Linguagem C – Comandos diversos

Protótipos das funções matemáticas

Os protótipos das funções matemáticas ficam no arquivo math.h. Veja alguns protótipos:

```
double sin(double arg);
double cos(double arg);
double tan(double arg)
double exp(double arg);
double log(double num);
double log10(double num);
double pow(double base, double exp);
double sqrt(double num);
```

Linguagem C – Comandos diversos

Alguns erros comuns de programação

Por ser uma linguagem que dá muito poder ao programador, também é muito fácil de errar em C.

Como o compilador aceita praticamente tudo o que se escreve, o programador deve ter atenção redobrada na hora de programar.

Linguagem C – Comandos diversos

Erros de ordem de processamento

Os operadores de incremento e decremento são usados na maioria dos programas em C, e a ordem de ocorrência da operação é afetada pelo fato de esses operadores precederem ou sucederem a variável. Logo, se y=10

```
x = y++;
```

será diferente de

```
x = ++y;
```

Linguagem C – Comandos diversos

Erros de limite

Muitas funções em C (inclusive as das bibliotecas) não possuem (ou possuem pouca) verificação de limites. Assim, a chamada a gets abaixo pode gerar um problema caso o usuário digite mais que 20 caracteres.

```
main()
{ char texto[21]; gets(texto);
}
```

Linguagem C – Comandos diversos

Omissões de declaração de função

Esquecer de definir o protótipo de uma função pode causar erro.

O compilador estará esperando sempre que as funções não declaradas retornem um inteiro.

O mesmo problema ocorre também com os parâmetros.

Linguagem C – Comandos diversos

Erros de argumentos de chamada

Os argumentos passados a uma função devem ser do mesmo tipo do esperado por elas. O programa abaixo está errado:

```
main()
{
   int x;
   char string[10];
   scanf("%d%s", x, string);
}
```