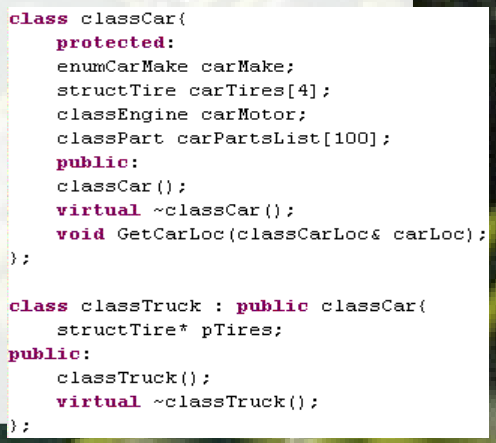
Curso de Programação C em Ambientes Linux – Aula 02

Centro de Engenharias da Mobilidade - UFSC

Professores Gian Berkenbrock e Giovani Gracioli

http://www.lisha.ufsc.br/C+language+course+ resources



```
class classCar{
    protected:
    enumCarMake carMake;
    structTire carTires[4];
    classEngine carMotor;
    classPart carPartsList[100];
    public:
    classCar();
    virtual ~classCar();
    void GetCarLoc(classCarLoc& carLoc);
};

class classTruck : public classCar{
    structTire* pTires;
public:
    classTruck();
    virtual ~classTruck();
};
```

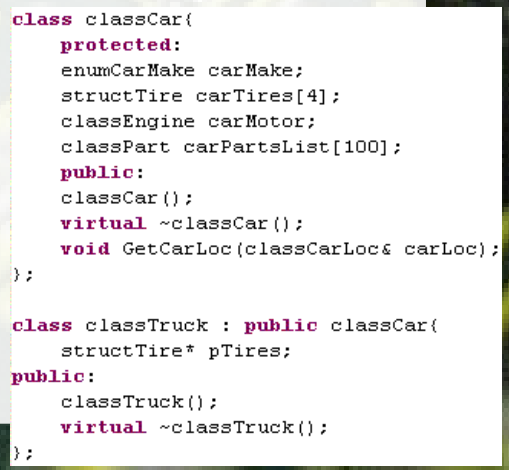
Conteúdo desta aula

- Estrutura de um programa em C
- Constantes e palavras reservadas
- Tipos primitivos
- Declaração de variáveis
- Operadores aritméticos, atribuição, relacionais e lógicos
- Funções de entrada e saída formatada
- Estruturas de controle de fluxo

Estrutura de um Programa C

Curso de Programação C em Ambientes Linux

Aula 02 de 5 – 05/08/2014



```
class classCar{
    protected:
    enumCarMake carMake;
    structTire carTires[4];
    classEngine carMotor;
    classPart carPartsList[100];
    public:
    classCar();
    virtual ~classCar();
    void GetCarLoc(classCarLoc& carLoc);
};

class classTruck : public classCar{
    structTire* pTires;
public:
    classTruck();
    virtual ~classTruck();
};
```

Estrutura básica de um programa em C

```
1  /* Figura 2.1: fig02_01.c
2     Primeiro programa em C */
3  #include <stdio.h>
4
5  /* função main inicia execução do programa */
6  int main( void )
7  {
8     printf( "Bem-vindo a C!\n" );
9
10    return 0; /* indica que o programa terminou com sucesso */
11 } /* fim da função main */
```

```
Bem-vindo a C!
```

Figura 2.1 ■ Primeiro programa em C.

Caracteres de escape

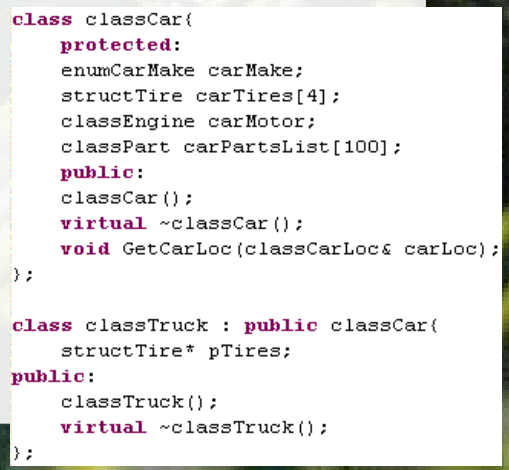
Sequência de escape	Descrição
\n	Nova linha. Posiciona o cursor da tela no início da próxima linha.
\t	Tabulação horizontal. Move o cursor da tela para a próxima posição de tabulação.
\a	Alerta. Faz soar o alarme do sistema.
\\	Barra invertida. Insere um caractere de barra invertida em uma string.
\"	Aspas. Insere um caractere de aspas em uma string.

Figura 2.2 ■ Algumas sequências comuns de escape.

Tipos Primitivos

Curso de Programação C em Ambientes Linux

Aula 02 de 5 – 05/08/2014



```
class classCar{
    protected:
    enumCarMake carMake;
    structTire carTires[4];
    classEngine carMotor;
    classPart carPartsList[100];
    public:
    classCar();
    virtual ~classCar();
    void GetCarLoc(classCarLoc& carLoc);
};

class classTruck : public classCar{
    structTire* pTires;
public:
    classTruck();
    virtual ~classTruck();
};
```

Tipos da Linguagem C (1)

- C é uma linguagem fortamente “tipada”, ou seja, cada variável declarada deve ter seu tipo informado
- Existem 5 tipos básicos em C
  - char, int, float, void, double
- Para cada tipo existem ainda os modificadores de tipo
  - signed, unsigned, long, short e long long
- Ao float não pode-se aplicar nenhum modificador e ao double apenas o long

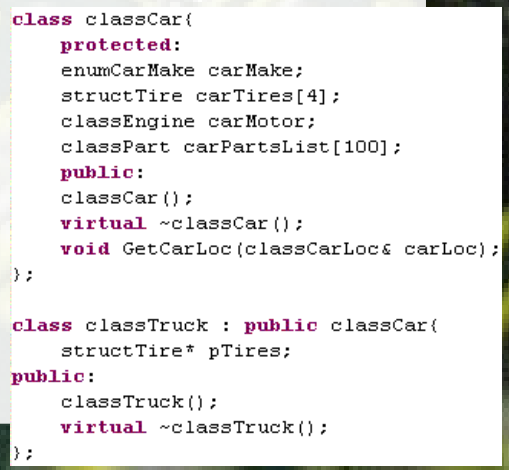
Tipos da Linguagem C (2)

- Os modificadores de tipo estão atrelados ao tamanho das variáveis
- Um inteiro normalmente possui 32 bits, logo os valores podem variar de -2^16 a 2^16 – 1
- Um inteiro sem sinal (unsigned) pode variar de 0 a 2^32 – 1
- Para descobrir o tamanho de um tipo, é possível usar o comando sizeof:
  - printf(“tamanho de um inteiro %d\n”, sizeof(int));

Constantes e Palavras Reservadas

Curso de Programação C em Ambientes Linux

Aula 02 de 5 – 05/08/2014



```
class classCar{
    protected:
    enumCarMake carMake;
    structTire carTires[4];
    classEngine carMotor;
    classPart carPartsList[100];
    public:
    classCar();
    virtual ~classCar();
    void GetCarLoc(classCarLoc& carLoc);
};

class classTruck : public classCar{
    structTire* pTires;
public:
    classTruck();
    virtual ~classTruck();
};
```

Constantes

- Constantes são valores que são mantidos fixos pelo compilador
- Constantes de tipos básicos
  - char: 'b', '\n'
  - int: 1202, -102
  - unsigned int 100, 24812
- Hexadecimal
  - 0xEF
  - 0x92
- String: “João”

Palavras chave

- Algumas palavras têm significado especial para o compilador C, de modo que você precisa ter cuidado para não usá-las como nomes de variáveis

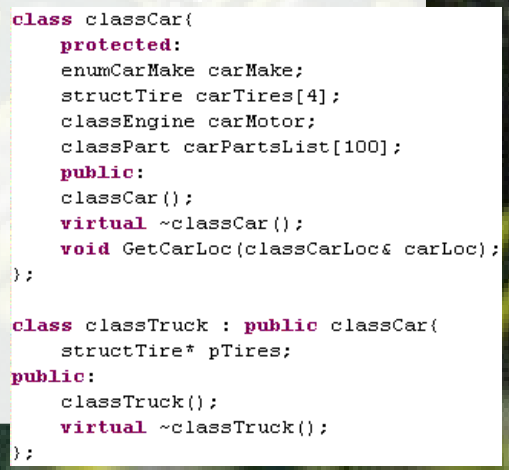
Palavras-chave			
auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while
Palavras-chave acrescentadas na C99			
_Bool _Complex _Imaginary inline restrict			

Figura 2.15 ■ Palavras-chave em C.

Declaração de Variáveis

Curso de Programação C em Ambientes Linux

Aula 02 de 5 – 05/08/2014



```
class classCar{
    protected:
    enumCarMake carMake;
    structTire carTires[4];
    classEngine carMotor;
    classPart carPartsList[100];
    public:
    classCar();
    virtual ~classCar();
    void GetCarLoc(classCarLoc& carLoc);
};

class classTruck : public classCar{
    structTire* pTires;
public:
    classTruck();
    virtual ~classTruck();
};
```

Declaração e Inicialização de Variáveis (1)

- As variáveis em C devem ser declaradas antes de serem usadas
- A forma geral de declaração é:
  - tipo_da_variável lista_de_variáveis ; (não esqueça do ;)
- As variáveis da lista terão todas o mesmo tipo e deverão ser separadas por vírgula:
  - char ch, letra;
- Alternativamente:
  - char ch;
  - char letra;

Declaração e Inicialização de Variáveis (2)

- As variáveis declaradas podem ser inicializadas na declaração ou durante o programa:
  - int a = 100;
    int b;

    float c;

    char d = 'u';

    ....

    b = 100;

    c = 5.10;

Um exemplo

- Somando dois números inteiros

```
#include <stdio.h>
int main(void) {
    int inteiro1 = 100, inteiro2 = 1000, soma;
    soma = inteiro1 + inteiro2;
    printf("Valor da soma = %d\n", soma);
    return 0;
}
```

Conceitos de memória

- Nomes de variáveis, tais como inteiro1, inteiro2 e soma correspondem, na realidade, a posições na memória do computador
- Toda variável tem um nome, um tipo e um valor

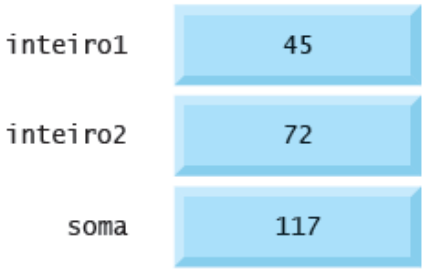


Figura 2.8 ■ Posições de memória após um cálculo.

Operadores Aritméticos, Atribuição, Relacionais e Lógicos

Curso de Programação C em Ambientes Linux

Aula 02 de 5 – 05/08/2014

```
class classCar{
    protected:
    enumCarMake carMake;
    structTire carTires[4];
    classEngine carMotor;
    classPart carPartsList[100];
    public:
    classCar();
    virtual ~classCar();
    void GetCarLoc(classCarLoc& carLoc);
};

class classTruck : public classCar{
    structTire* pTires;
public:
    classTruck();
    virtual ~classTruck();
};
```

Aritmética em C

Operação em C	Operador aritmético	Expressão algébrica	Expressão em C
Adição	+	$f + 7$	f + 7
Subtração	–	$p - c$	p – c
Multiplicação	*	bm	b * m
Divisão	/	x / y ou $\frac{x}{y}$ ou $x \div y$	x / y
Módulo ou resto da divisão entre 2 inteiros	%	r mod s	r % s

Figura 2.9 ■ Operadores aritméticos.

Precedência dos Operadores Aritméticos

Operador(es)	Operação(ões)	Ordem de avaliação (precedência)
( )	Parênteses	Avaliados em primeiro lugar. Se os parênteses forem aninhados, a expressão no par mais interno é a primeira a ser avaliada. Se houver vários pares de parênteses 'no mesmo nível' (ou seja, não aninhados), eles serão avaliados da esquerda para a direita.
* / %	Multiplicação Divisão Módulo	Avaliados em segundo lugar. Se houver vários, serão avaliados da esquerda para a direita.
+ –	Adição Subtração	Avaliados por último. Se houver vários, serão avaliados da esquerda para a direita.

Figura 2.10 ■ Precedência de operadores aritméticos.

Avaliação aritmética

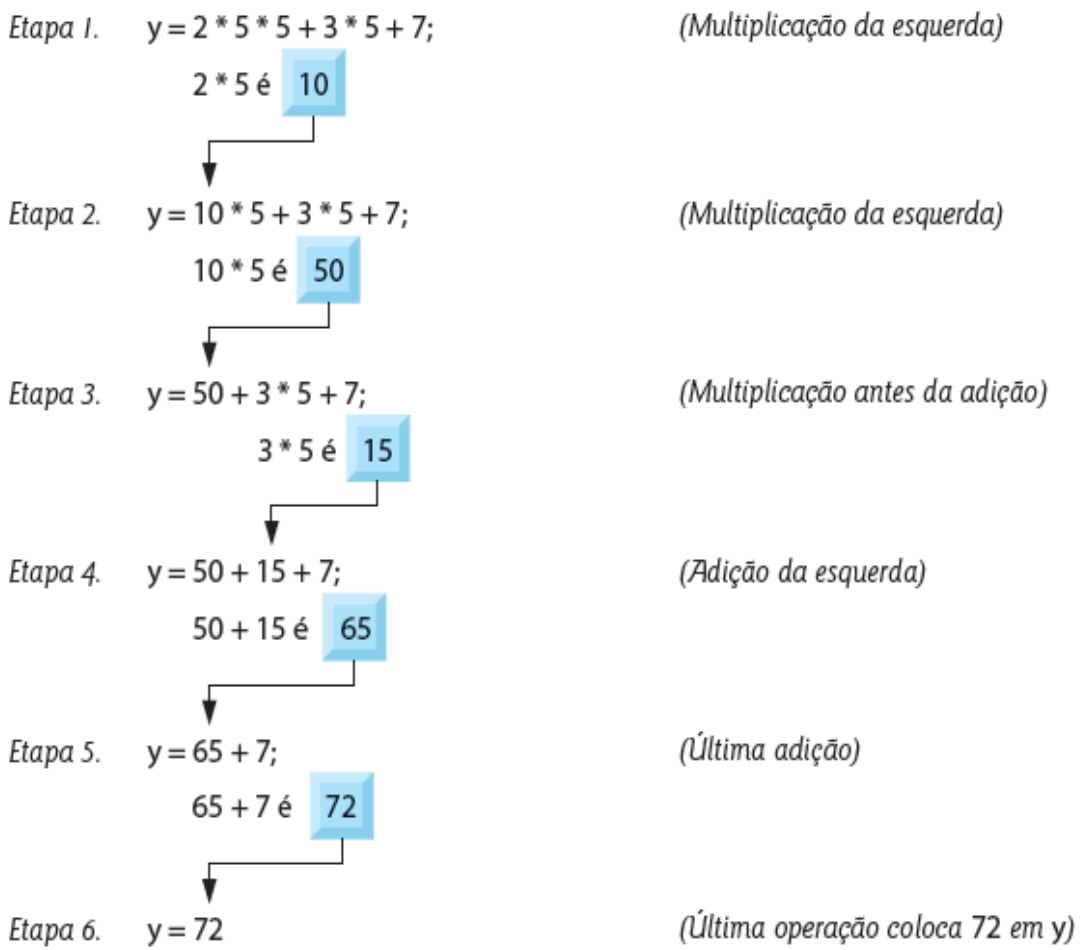


Figura 2.11 ■ Ordem em que um polinômio de segundo grau é avaliado.

Operadores de atribuição e incremento

Operador de atribuição	Exemplo de expressão	Explicação	Atribui
Considere: **int** c = 3, d = 5, e = 4, f = 6, g = 12;			
+=	c += 7	c = c + 7	10 a c
-=	d -= 4	d = d - 4	1 a d
*=	e *= 5	e = e * 5	20 a e
/=	f /= 3	f = f / 3	2 a f
%=	g %= 9	g = g % 9	3 a g

Figura 3.11 ■ Operadores aritméticos de atribuição.

Operador	Exemplo de expressão	Explicação
++	++a	Incrementa a em 1, e então usa o novo valor de a na expressão em que a estiver.
++	a++	Usa o valor corrente de a na expressão em que a estiver, e então incrementa a em 1.
--	--b	Decrementa b em 1, e então usa o novo valor de b na expressão em que b estiver.
--	b--	Usa o valor corrente de b na expressão em que b estiver, e então decrementa b em 1.

Figura 3.12 ■ Operadores de incremento e decremento.

Operadores de igualdade e atribuição (1)

- Existe um erro comum: troca acidental dos operadores == (igualdade) e = (atribuição)
- O que torna essas trocas tão prejudiciais é o fato de elas normalmente não causarem erros de compilação
- Em vez disso, instruções com esses erros normalmente são compiladas corretamente e permitem a execução dos programas até o fim, mas é provável que gerem resultados incorretos por causa de erros lógicos durante a execução

Operadores de igualdade e atribuição (2)

- Suponha que queiramos escrever

```
if ( codPgto == 4 )
   printf( "Você ganha um bônus!" );
```

- Porém, por engano escrevemos

```
if ( codPgto = 4 )
   printf( "Você ganha um bônus!" );
```

- A operação de atribuição sempre retorna o valor atribuído (4 neste caso). Como qualquer valor != 0 é verdadeiro, o if é executado
- Para “prevenir” tal erro, basta inverter: 4 == codPgto

Operadores Relacionais

Operador de igualdade ou relacional na álgebra	Operador de igualdade ou relacional em C	Exemplo de condição em C	Significado da condição em C
Operadores de igualdade			
=	==	x == y	x é igual a y
≠	!=	x != y	x não é igual a y
Operadores relacionais			
>	>	x > y	x é maior que y
<	<	x < y	x é menor que y
≥	>=	x >= y	x é maior ou igual a y
≤	<=	x <= y	x é menor ou igual a y

Figura 2.12 ■ Operadores de igualdade e relacionais.

Precedência dos Operadores

Operadores	Associatividade
()	esquerda para direita
* / %	esquerda para direita
+ -	esquerda para direita
< <= > >=	esquerda para direita
== !=	esquerda para direita
=	direita para esquerda

Figura 2.14 ■ Precedência e associatividade dos operadores discutidos até aqui.

Operadores Lógicos (1)

- C oferece operadores lógicos, que podem ser usados para formar condições mais complexas ao combinar condições simples
- Os operadores lógicos são && (AND lógico), || (OR lógico) e ! (NOT lógico, também chamado de negação lógica)
- Vejamos alguns exemplos de cada um desses operadores
- Suponha que queiramos garantir que duas condições sejam verdadeiras antes de escolher certo caminho de execução

Operadores Lógicos (2)

- Neste caso, podemos usar o operador &&

```
if ( sexo == 1 && idade >= 65 )
 ++idosoFeminino;
```

- Essa estrutura if contém duas condições simples
- A condição sexo == 1 poderia ser avaliada, por exemplo, para determinar se uma pessoa é do sexo feminino
- A condição idade >= 65 é avaliada para determinar se uma pessoa é idosa
- As duas condições simples são avaliadas primeiro, pois as precedências de == e >= são ambas maiores que a precedência de &&

Precedência de Todos os Operadores

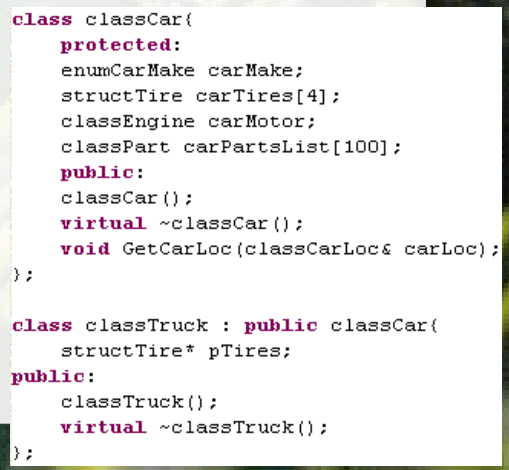
Operadores	Associatividade	Tipo
++ (*pós-fixo*) -- (*pós-fixo*)	direita para esquerda	pós-fixo
+ - ! ++ (*prefixo*) -- (*prefixo*) (*tipo*)	direita para esquerda	unário
* / %	esquerda para direita	multiplicativo
+ -	esquerda para direita	aditivo
< <= > >=	esquerda para direita	relacional
== !=	esquerda para direita	igualdade
&&	esquerda para direita	AND lógico
||	esquerda para direita	OR lógico
?:	direita para esquerda	condicional
= += -= *= /= %=	direita para esquerda	atribuição
,	esquerda para direita	vírgula

Figura 4.16 ■ Precedência e associatividade dos operadores.

E/S formatada

Curso de Programação C em Ambientes Linux

Aula 02 de 5 – 05/08/2014



```
class classCar{
    protected:
    enumCarMake carMake;
    structTire carTires[4];
    classEngine carMotor;
    classPart carPartsList[100];
    public:
    classCar();
    virtual ~classCar();
    void GetCarLoc(classCarLoc& carLoc);
};

class classTruck : public classCar{
    structTire* pTires;
public:
    classTruck();
    virtual ~classTruck();
};
```

E/S Formatada

- As duas principais funções que realizam E/S formatada em C são printf() e scanf()
- A função printf recebe uma string e os valores a serem impressos como parâmetros
  - printf(“Primeiro inteiro = %d, segundo inteiro = %d\n”, 10, 20);
- O %d indica que será impresso um valor inteiro
- Existe uma letra para cada tipo
  - %c = caracter
  - %f = ponto flutuante
  - %s = string

Mais exemplos de printf

- É possível reservar um tamanho específico para o valor impresso:

```
int a = 100;
printf("Valor = %5d\n", a);
```

- O campo %5d indica que o valor terá 5 caracteres de comprimento no mínimo
- %05d preenche com zeros caso o valor tenha menos que 5 dígitos
- %-5d permite o alinhamento à esquerda
- %5.8d indica o número mínimo e máximo de dígitos
- %5.2f indica ponto flutuante de comprimento 5 e 2 casas depois da vírgula

scanf

- Realiza a leitura de valores. Exemplo:

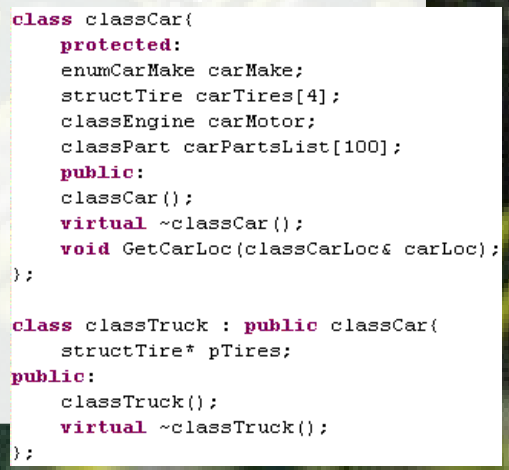
```
int a;
printf(“Digite um valor inteiro: “);
scanf(“%d”, &a);
printf(“Valor digitado = %d\n”, a);
```

- Note que %d informa que scanf está esperando um inteiro
- %f para ponto flutuante, %c para caracter, %s para string
- Faça um programa que leia dois números inteiros do usuário, some-os e imprima o resultado

Estruturas de Controle e Repetição

Curso de Programação C em Ambientes Linux

Aula 02 de 5 – 05/08/2014



```
class classCar{
    protected:
    enumCarMake carMake;
    structTire carTires[4];
    classEngine carMotor;
    classPart carPartsList[100];
    public:
    classCar();
    virtual ~classCar();
    void GetCarLoc(classCarLoc& carLoc);
};

class classTruck : public classCar{
    structTire* pTires;
public:
    classTruck();
    virtual ~classTruck();
};
```

Estruturas de controle

- Estruturas de sequência
- Estruturas de seleção (*if*, *if...else* e *switch*)
- Estruturas de repetição (*while*, *do...while* e *for*)

A estrutura de seleção if...else

- Exemplo:

```
if ( nota >= 60 ) {
   printf( "Aprovado\n" );
} /* fim do if */
else {
   printf( "Reprovado\n" );
} /* fim do else */
```

Operador ternário ?

- Uma expressão como

```
if(a > 0)
    b = -150;
eles
    b = 150;
```

- Pode ser simplificada usando o operador ternário ?

```
b = a > 0 ? -150 : 150;
```

- O operador ternário ? tem a seguinte forma:
  - condição ? expressão 1 : expressão 2;
  - printf(“%s\n”, nota >= 60 ? “Aprovado” : “Reprovado”);

if...else aninhados

```
if ( nota >= 90 )
   printf( "A\n" );
eles if ( nota >= 80 )
   printf("B\n");
eles if ( nota >= 70 )
   printf("C\n");
else if ( nota >= 60 )
   printf( "D\n" );
else
   printf( "F\n" );
```

A estrutura de seleção switch (1)

- Ocasionalmente, um algoritmo conterá uma série de decisões em que uma variável, ou expressão, será testada separadamente para cada um dos valores inteiros constantes que ela possa vir a assumir, e diferentes ações serão tomadas
- Isso é chamado de seleção múltipla
- C nos oferece a estrutura de seleção múltipla switch para lidar com essa tomada de decisão
- Consiste em uma série de rótulos *case* com um *default*

A estrutura de seleção switch (2)

```
1  /* Fig. 4.7: fig04_07.c
2     Contando notas de letra */
3  #include <stdio.h>
4
5  /* função main inicia a execução do programa */
6  int main( void )
7  {
8     int nota; /* uma nota */
9     int aCont = 0; /* número de As */
10    int bCont = 0; /* número de Bs */
11    int cCont = 0; /* número de Cs */
12    int dCont = 0; /* número de Ds */
13    int fCont = 0; /* número de Fs */
14
15    printf(  "Digite as notas em letra.\n"  );
16    printf(  "Digite o caractere EOF para encerrar a entrada.\n"  );
17
18    /* loop até que o usuário digite sequência de fim de arquivo */
19    while ( ( nota = getchar() ) != EOF ) {
20
21       /* determina qual nota foi digitada */
22       switch ( nota ) { /* switch aninhado no while */
23
```

A estrutura de seleção switch (3)

```
24          case 'A':   /* nota foi 'A' maiúsculo */
25          case 'a':   /* ou 'a' minúsculo */
26             ++aCount; /* incrementa aCount */
27             break;   /* necessário para sair do switch */
28
29          case 'B':   /* nota foi 'B' maiúsculo */
30          case 'b':   /* ou 'b' minúsculo */
31             ++bCount; /* incrementa bCount */
32             break;   /* sai do switch */
33
34          case 'C':   /* nota foi 'C' maiúsculo */
35          case 'c':   /* ou 'c' minúsculo */
36             ++cCount; /* incrementa cCount */
37             break;   /* sai do switch */
38
39          case 'D':   /* nota foi 'D' maiúsculo */
```

Figura 4.7 ■ Exemplo de switch (Parte 1 de 2.)

A estrutura de repetição while

- Uma estrutura de repetição permite que você especifique que uma ação deverá ser repetida enquanto alguma condição permanecer verdadeira
- Como exemplo de um *while* real, considere um segmento de programa projetado para encontrar a primeira potência de 3 maior que 100

```
produto = 3;

while(produto <= 100){
   produto = 3 * produto ;
} /* fim do while */
```

Condição controlada por contador (1)

- Considere o seguinte problema

Uma turma de dez alunos realiza um teste. As notas (inteiros, no intervalo de 0 a 100) dadas aos alunos estão à sua disposição. Determine a média das notas da turma.

- *Resolução:*

```
1  Define total como zero
2  Define contador de notas como um
3
4  Enquanto contador de notas é menor ou igual a dez
5      Lê a próxima nota
6      Soma a nota ao total
7      Soma um ao contador de notas
8
9  Define a média da turma como o total dividido por dez
10 Imprime a média da turma
```

Figura 3.5 ■ Algoritmo de pseudocódigo que usa a repetição controlada por contador para resolver o problema da média da turma.

Condição controlada por contador (2)

```c
/* Figura 3.6: fig03_06.c
   Programa de média da turma com repetição controlada por contador */
#include <stdio.h>

/* função main inicia execução do programa */
int main( void )
{
   int contador; /* número da nota a digitar em seguida */
   int nota;     /* valor da nota */
   int total;    /* soma das notas inseridas pelo usuário */
   int média;    /* média das notas */

   /* fase de inicialização */
   total = 0;    /* inicializa total */
   contador = 1; /* inicializa contador do loop */

   /* fase de processamento */
   while ( contador <= 10 ) {      /* loop 10 vezes */
      printf( "Digite a nota: " ); /* prompt para inserção */
      scanf( "%d", &nota );        /* lê a nota do usuário */
      total = total + nota;        /* soma nota ao total */
      contador = contador + 1;     /* incrementa contador */
   } /* fim do while */

   /* fase de término */
   média = total / 10; /* divisão de inteiros */

   printf( "Média da turma é %d\n", média ); /* exibe resultado */
   return 0; /* indica que programa foi concluído com sucesso */
} /* fim da função main */
```

A estrutura de repetição for (1)

- O formato geral do for é:

```
for (expressão1; expressão2; expressão3) {
   instruções

}
```

- Na maior parte dos casos, a estrutura for pode ser representada com uma estrutura while equivalente, da seguinte forma:

```
expressão1;

while (expressão2) {
   instruções
   expressão3;

}
```

A estrutura de repetição for (2)

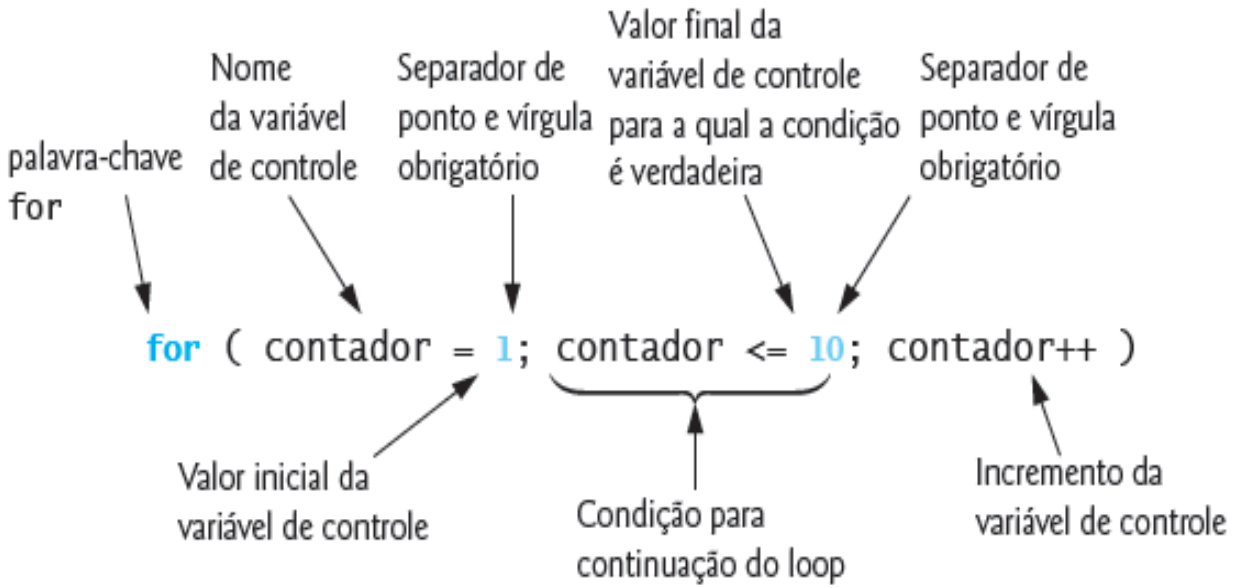


Figura 4.3 ■ Componentes do cabeçalho da estrutura `for`.

A estrutura de repetição for (3)

```
1  /* Fig. 4.2: fig04_02.c
2     Repetição controlada por contador com a estrutura for */
3  #include <stdio.h>
4
5  /* função main inicia a execução do programa */
6  int main( void )
7  {
8     int contador; /* declara o contador */
9
10    /* inicialização, condição de repetição e incremento
11       são todos incluídos no cabeçalho da estrutura for. */
12    for ( contador = 1; contador <= 10; contador++ ) {
13       printf( "%d\n", contador );
14    } /* fim do for */
15
16    return 0; /* indica que o programa terminou com sucesso */
17 } /* fim da função main */
```

Figura 4.2 ■ Repetição controlada por contador com a estrutura `for`.

Exemplos da estrutura for

Alterne a variável de controle de 1 a 100 em incrementos de 1.

```
for ( i = 1; i <= 100; i++ )
```

Alterne a variável de controle de 100 a 1 em incrementos de -1 (decrementos de 1).

```
for ( i = 100; i >= 1; i-- )
```

Alterne a variável de controle de 7 a 77 em intervalos de 7.

```
for ( i = 7; i <= 77; i += 7 )
```

Alterne a variável de controle de 20 a 2 em intervalos de -2.

```
for ( i = 20; i >= 2; i -= 2 )
```

Alterne a variável de controle na seguinte sequência de valores: 2, 5, 8, 11, 14, 17.

```
for ( j = 2; j <= 17; j += 3 )
```

Alterne a variável de controle na seguinte sequência de valores: 44, 33, 22, 11, 0.

```
for ( j = 44; j >= 0; j -= 11 )
```

A estrutura de repetição do..while (1)

- Similar à estrutura while
- Na instrução while a condição da continuação de loop é testada no início do loop, antes que seu corpo seja executado
- A estrutura do…while testa a condição da continuação do loop depois que o corpo do loop é executado
- Portanto, o corpo do loop será executado pelo menos uma vez

A estrutura de repetição do..while (2)

- Estrutura
  do {

  instruções;
  } while(condição);

```
1  /* Fig. 4.9: fig04_09.c
2     Usando a estrutura de repetição do/while */
3  #include <stdio.h>
4
5  /* função main inicia a execução do programa */
6  int main( void )
7  {
8     int contador = 1; /* inicializa contador */
9
10    do {
11       printf( "%d  ", contador ); /* exibe contador */
12    } while ( ++contador <= 10 ); /* fim de do...while */
13
14    return 0; /* indica que o programa foi concluído com sucesso */
15 } /* fim da função main */
```

```
1  2  3  4  5  6  7  8  9  10
```

Figura 4.9 ■ Exemplo de estrutura do...while.

O comando break

- O comando break, quando executado em uma estrutura while, for, do…while ou switch causa uma saída imediata dessa estrutura
- A execução do programa continua com a próxima instrução
- Os usos comuns do comando break são para escapar mais cedo de um loop ou para ‘pular’ o restante de uma estrutura switch

Exemplo do comando break

```
1  /* Fig. 4.11: fig04_11.c
2     Usando o comando break em uma estrutura for */
3  #include <stdio.h>
4
5  /* função main inicia a execução do programa */
6  int main( void )
7  {
8     int x; /* contador */
9
10    /* loop por 10 vezes */
11    for ( x = 1; x <= 10; x++ ) {
12
13       /* se x é 5, encerra o loop */
14       if ( x == 5 ) {
15          break; /* sai do loop somente se x é 5 */
16       } /* fim do if */
17
18       printf( "%d ", x ); /* exibe valor de x */
19    } /* fim de for */
20
21    printf( "\nSaiu do loop em x == %d\n", x );
22    return 0; /* indica que o programa foi concluído com sucesso */
23 } /* fim da função main */
```

```
1 2 3 4

Saiu do loop em x == 5
```

Figura 4.11 ■ Usando o comando break em uma estrutura for.

O comando continue

- O comando continue, quando executado em uma estrutura while, for ou do…while, ‘pula’ as instruções restantes no corpo dessa estrutura de controle e realiza a próxima iteração do loop
- Nas estruturas while e do…while, o teste de continuação do loop é avaliado imediatamente após o comando continue ser executado
- Na estrutura for a expressão de incremento é executada e, depois, o teste de continuação do loop é avaliado

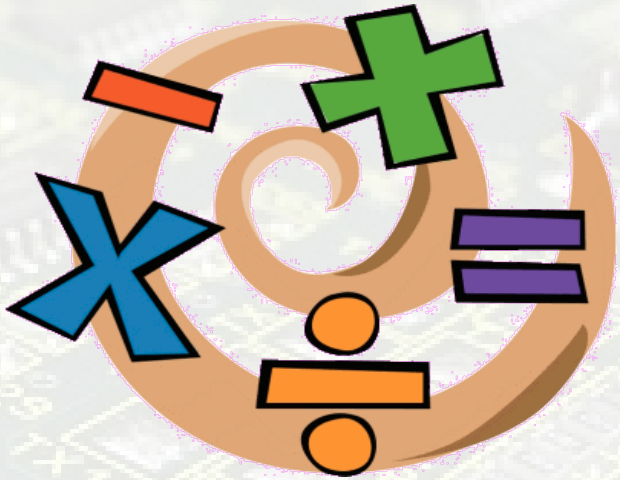
Exemplo do comando continue

```
 7  /* Fig. 4.12: fig04_12.c
 8     Usando o comando continue em uma estrutura for */
 9  #include <stdio.h>
10
11  /* função main inicia a execução do programa */
12  int main( void )
13  {
14     int x; /* contador */
15
16     /* loop por 10 vezes */
17     for ( x = 1; x <= 10; x++ ) {
18
19        /* se x é 5, continua com a próxima iteração do loop */
20        if ( x == 5 ) {
21           continue; /* salta código restante no corpo do loop */
22        } /* fim do if */
23
24        printf( "%d ", x ); /* exibe valor de x */
25     } /* fim do for */
26
27     printf( "\nUsou continue para pular a exibição do valor 5\n" );
28     return 0; /* indica que o programa foi concluído com sucesso */
29  } /* fim da função main */
```

```
1 2 3 4 6 7 8 9 10
Usou continue para pular a exibição do valor 5
```

Figura 4.12 ■ Usando o comando continue em uma estrutura for.

Finalizando



Revisão

- Estruturas de seleção: if, if/else e switch
- Estruturas de repetição: while, do..while e for
- Os comandos break e continue
- Operações aritméticas
- Operadores de atribuição e incremento
- Operadores de igualdade e atribuição

Sugestões Finais

- Resolvam a lista de exercícios relacionada com os temas da aula
  - Só se aprende a programar, programando
- Dúvidas sobre os exercícios podem ser enviadas por e-mail
- Leiam o material de apoio
  - Curso de C da UFMG: http://mico.ead.cpdee.ufmg.br/cursos/C/

Referências Bibliográficas

- Paul Deitel e Harvey Deitel, C: como programar, 6a edição, Ed. Prentice Hall Brasil, 2011.
- Curso de C da UFMG: http://mico.ead.cpdee.ufmg.br/cursos/C/