Linguagens de Programação

Capítulo 3: Tipos de Dados em C

Programas trabalham com dados. Nós fornecemos números, letras e palavras ao computador e esperamos que ele faça algo com estes dados, como por exemplo, calcular o valor de um pagamento ou mostrar os dados ordenados.

Resumo da aula:

- Palavras-chave: int, long, unsigned, char, float, Double, _Bool,_Complex,_Imaginary
- Operador: sizeof
- Função: scanf()
- Tipos de dados básicos
- Diferença entre tipos inteiros e reais (ponto-flutuante)
- Escrevendo constantes e declarando variáveis
- Usando printf() e scanf()

Exemplo: ouro.c

```c
/* ouro.c -- seu peso em ouro */
#include <stdio.h>
int main (void)
{
	float peso;
	float valor;

	printf("Você vale seu peso em ouro?\n");
	printf("Vamos verificar.\n");

	printf("Digite seu peso em kilos: ");
	scanf("%f", &peso);

	// 1 grama de ouro vale R$45
	valor = peso * 1000 * 45;

	printf("Seu peso em ouro vale R$%.2f.\n", valor);

	return 0;
}
```

Novidades do programa:

- Declaração de um novo tipo de variável: float
- Novas constantes de tipo real (ponto-flutuante)
- Uso do formatador %f e %.2f com a função printf()
- Uso da função scanf()
- O programa é interativo

Esta aula explora em detalhes os dois primeiros itens dessa lista e faz uso de scanf() e printf().

Tipos de dados

- Int (short, long, signed, unsigned)
- char
- float, double

Inteiro vs Real

Mostrar como os dados são armazenados em bits e bytes.

O tipo de dado está diretamente relacionado com a quantidade de memória que ele ocupa. A memória é um conjunto de células identificadas por um endereço.

1 célula = 1 byte = 8 bits
1 bit = 2 estados (0 e 1)
1 byte possui 2^8 estados = 256

1 Byte

00000000	0
00000001	1
00000010	2
00000011	3
....	...
11111111	255

O tipo int

- Declaração e inicialização
- Constantes
- Impressão com printf() e %d
- Outros tipos inteiros (short, long, unsigned, signed)
- Por que existe tipos múltiplos para inteiros?
- Imprimindo inteiros short, long e unsigned

O tipo char

- Declaração e inicialização
- Caracteres não imprimíveis (enter, backspace, beep, etc.)
- Tabela de seqüências de escape
- Imprimindo caracteres com printf() e %c

O tipo float

- Declaração e inicialização
- Representação de constantes
- Impressão usando printf(), %f e %e
- Erros de arredondamento

Sumário

- Tabela de tamanhos de tipos
- Programa typesize.c

Capítulo 4: Strings e Entrada/Saída Formatada

Sumário:

- Função strlen()
- Criando constantes com o pré-processador #define
- A palavra chave const
- Strings
- Usando scanf() e printf() com strings

Exemplo:

```
#include <stdio.h>
#include <string.h>        // protótipo para strlen()
#define DENSIDADE 62.4     // densidade humana em libras por pés cúbicos

int main()
{
   float peso, volume;
   int tamanho, letras;
   char nome[40];        // nome é um vetor de 40 chars

   printf("Olá! Qual o seu nome?\n");
   scanf("%s", nome);
   printf("%s, qual o seu peso em libras?\n", nome);
   scanf("%f", &peso);

   tamanho = sizeof (nome);
   letras = strlen(nome);
   volume = peso / DENSIDADE;
   printf("Bom, %s, seu volume é %f pés cúbicos.\n", nome, volume);
   printf("Seu nome tem %d letras,\n", letras);
   printf("e nós temos %d bytes ara guardá-lo.\n", tamanho);

   return 0;
}
```

Strings

C não possui um tipo específico para valores literais, eles são armazenados em vetores de caracteres.

Uma string termina com um caractere '\0'. Mostrar como é representada a string nome na memória do computador.

```
char ch;            // Aloca 1 byte

char nome[5];   //Aloca 5 bytes
```

'x' é diferente de "x".

A função Strlen()

Essa função retorna o comprimento de uma string em caracteres. O protótipo dessa função está no arquivo string.h, por isso requer um #include <string.h> para seu uso.

letras = strlen(nome);

tamanho = sizeof (nome);

Constantes e o Pré-processador C

Algumas vezes é preciso usar constantes em um programa.

circunferencia = 3.14159 * diametro;

Razões para usar constantes:

- Proporcionar maior clareza no programa:
  total = 0.015 * consumo;
  total = taxaDeServico * consumo;
- Alterar facilmente todas as ocorrências de um valor no programa:
  area =3.14159 * raio * raio;
  circunf = 2.0 * 3.14159 * raio;

Isso poderia ser feito da seguinte forma:

float PI = 3.14159;

Mas PI é uma variável e não uma constante. Uma forma melhor de declarar uma constante é usando a diretiva de pré-processamento #define ou a palavra-chave const.

#define PI 3.14159

Ou

const float PI = 3.14159;

A diretiva define pode definir constantes de qualquer tipo:

#define TAMANHOMAX 40

#define JS “Judson Santiago”

#define TAB ‘\t’

Explorando a função printf()

Sintaxe:

printf (string-de-controle, item1, item2, …)

Item1, item2, etc. são itens a serem impressos na tela. Eles podem ser constantes, variáveis ou expressões que são avaliadas antes de impressas. A string de controle é uma string que controla como os itens serão impressos. A string de controle deve conter um especificador de conversão para cada item a ser impresso.

%c	Caractere simples
%d	Valor decimal inteiro

%e	Número real em notação E
%f	Número real em notação decimal
%i	O mesmo que %d
%s	Vetor de caracteres
%%	Imprime o sinal de percentagem

Strings de controle contêm dois tipos de informação:

- Caracteres que são impressos na tela
- Especificadores de conversão

Exemplos:

```
printf(“A função pode imprimir apenas caracteres”);
printf(“%c%d”, ‘$’, 20);
printf(“O custo é de R$ %d“, 20);
printf(“O custo é de R$ %d“, custoTotal);
```

Modificações dos especificadores

digito	A largura mínima do espaço ocupado Exemplo: “%4d”
.digito	Número de dígitos após a vírgula Exemplo: “5.2f”
-	O item é alinhado à esquerda Exemplo: “%-20s”
+	Valores positivos são impressos com um sinal de + e os negativos com um sinal de – Exemplo: “%+6.2f”
espaço	Valores positivos são mostrados com um espaço no lugar do sinal positivo Exemplo: “% 6.2f”
0	Enche os espaços vazio com zero, no lugar de espaços vazios Exemplo: “%010d”, “%08.3f”

Exemplo:

```
#include <stdio.h>
#define PAGES 931

int main(void)
{
   printf("*%d*\n", PAGES);
   printf("*%2d*\n", PAGES);
   printf("*%10d*\n", PAGES);
   printf("*%-10d*\n", PAGES);
```

```
   return 0;
}
```

Saída:
```
*931*
*931*
*     931*
*931      *
```

Imprimindo strings longas

Existem 3 métodos:

- Usar mais de um printf()
- Usar a combinação contra-barra e enter
- Usar concatenação de constantes strings

Exemplo:

```
#include <stdio.h>
int main(void)
{
   printf("Here's one way to print a ");
   printf("long string.\n");
   printf("Here's another way to print a \
long string.\n");
   printf("Here's the newest way to print a "
          "long string.\n");     /* ANSI C */
   return 0;
}
```

Explorando a função scanf()

A entrada do teclado é texto porque as teclas do teclado geram caracteres texto: letras, dígitos e pontuação. Se você quer guardar a entrada como um valor numérico, seu programa deve converter o texto para um valor numérico. Esse é o trabalho do scanf().

A função scanf() converte a entrada texto em vários formatos: inteiro, real, caractere e string.

Da mesma forma que printf(), scanf() usa uma string de controle. A string de controle indica o tipo de dado de destino para a cadeia de caracteres lidos na entrada. A função printf() usa nomes de variáveis, constantes e expressões. A função scanf() usa endereços de variáveis.

Regras:

- Preceda o nome da variável com & para ler os valores dos tipos básicos (int, float, char)
- Não use & para ler strings em um vetor de caracteres.

Exemplo:

```
#include <stdio.h>
int main(void)
{
```

```
   int idade;                // variável
   float posses;             // variável
   char animal[30];          // string
   printf("Digite sua idade, valor das posses e animal favorito.\n");
   scanf("%d %f", &idade, &posses);        // use & aqui
   scanf("%s", animal);                    // não use & aqui
   printf("%d $%.2f %s\n", idade, posses, animal;
   return 0;
}
```

Saída:

```
Digite sua idade, valor das posses e animal favorito.
28
92360.88 papagaio
38 $92360.88 papagaio
```

A função scanf() usa os espaços em branco (nova linha, tabulações e espaços) para decidir como dividir a entrada em campos separados.

Capítulo 5: Operadores, Expressões e Instruções

Sumário:

- A estrutura de repetição WHILE
- Operadores da linguagem C
- Precedências dos operadores
- Funções com argumentos

A estrutura de repetição while

Sintaxe:

```
while (<condição>)
{
    <instruções>
}
```

```
enquanto (<condição>) faça
  <instruções>
fimenquanto
```

Semântica: enquanto a condição for verdadeira o bloco de instruções será executado. Para que a repetição não seja infinita a condição deve se tornar falsa em algum momento.

Exemplo de programa sem repetição:

```
/*converte graus Celsius para Fahrenheit */
#include <stdio.h>
#define ADJUSTE 1.8
#define ESCALA 32.0
int main(void)
{
   double grausF, grausC;
   grausC = 25.0;
   grausF = ESCALA *grausC + ADJUSTE;
   printf("Conversão de temperatura\n");
   printf("%10.1f %15.2f inches\n", grausC, grausF);
   return 0;
}
```

Exemplo com repetição:

```
/*converte graus Celsius para Fahrenheit */
#include <stdio.h>
#define ADJUSTE 1.8
#define ESCALA 32.0
int main(void)
{
   double grausF, grausC;
   printf("Conversão de temperatura\n");
   grausC = 15.0;
   while (grausC < 40.0)
   {
      grausF = ESCALA *grausC + ADJUSTE;
      printf("%10.1f %15.2f inches\n", grausC, grausF);
```

```
        grausC = grausC + 1.0;
    }
    return 0;
}
```

Operadores fundamentais

Operador de atribuição (=)

Sintaxe: <variável> = <expressão>

Semântica: o valor da expressão é atribuído à variável.

Exemplos:

carro = 2002;

Não pense que carro é igual a 2002 e sim que o valor 2002 é atribuído a variável carro.

i = i + 1; // Não faz sentido como igualdade matemática, mas é perfeitamente válido em C

2002 = carro; // Incorreto em C, mas válido na matemática.

Operador de adição (+)

Sintaxe: <op1> + <op2>

Semântica: o operador de adição soma os seus operandos, que podem ser constantes ou expressões mais complexas que resultam em valores aritméticos.

Exemplos:

printf("%d", 4+20); // imprime o número 24 e não a expressão

total = 4+20;

total = salário + extras;

Operador de subtração (-)

Sintaxe: <op1> - <op2>

Semântica: faz a subtração entre os seus operandos.

Exemplos:

ganhos = 224.0 – 24.0; // atribui o valor 200.0 para a variável ganhos

Operadores de sinal (+/-)

Sintaxe: - <op>

Semântica: o operador de sinal "-" provoca uma mudança de sinal do seu operando.

Exemplos:

invertido = -12;

duzia = -invertido; // atribui o valor 12 a variável duzia

duzia = +12; // não faz nada, é o mesmo que dúzia = 12

Operador de multiplicação (*)

Sintaxe: <op1> * <op2>

Semântica: o operador de multiplicação multiplica seus operandos.

Exemplos:

centimetros = 2.54 * polegadas; // multiplica polegadas por 2.54 e atribui o resultado

```
/* quadrado.c – produz uma tabela dos primeiros 20 quadrados */
#include <stdio.h>
int main(void)
{
   int num = 1;
   while (num < 21)
   {
      printf("%4d %6d\n", num, num * num);
      num = num + 1;
   }
   return 0;
}
```

Operador de divisão (/)

Sintaxe: <op1> / <op2>

Semântica: o operador divide seus operandos. O resultado da divisão depende do tipo de dados dos operandos. Se algum operando for real, o resultado é real.

Exemplos:

quatro = 12.0 / 3.0; // O resultado é 4.0

A divisão funciona de forma diferentemente para tipos inteiros e reais:

```
/* divide.c -- divisions we have known */
#include <stdio.h>
int main(void)
{
    printf("divisão inteira:  5/4  é  %d \n", 5/4);
    printf("divisão inteira:  6/3  é %d \n", 6/3);
    printf("divisão inteira:  7/4  é %d \n", 7/4);
    printf("divisão real: 7.0/4.0 é %1.2f \n", 7.0/4.0);
    printf("divisão mista:    7.0/4 é %1.2f \n", 7.0/4);
    return 0;
}
```

divisão inteira 5/4 é 1
divisão inteira: 6/3 é 2
divisão inteira: 7/4 é 1
divisão real: 7.0/4.0 é 1.75
divisão mista: 7.0/4 é 1.75

Precedência de operadores

Considere o código:

total = 4 + 4 * 3 / 2;

Qual a ordem de cálculo (precedência dos operadores)?

Operadores	Associatividade
()	Esquerda para a direita
+ - (unário)	Direita para a esquerda
*** /**	Esquerda para a direita
+ - (binário)	Esquerda para a direita
=	Direita para a esquerda

Exemplo:

```
/* regras.c – teste de precedência */
#include <stdio.h>
int main(void)
{
   int max, total;
   max = total = -(2 + 5) * 6 + (4 + 3 * (2 + 3));
   printf("max = %d \n", max);
   return 0;
}
```

Saída: -23

Operador de incremento e decremento (++/--)

Sintaxe:

Prefixo: ++op --op
Posfixo: op++ op--
Semântica: incrementa ou decrementa o valor do operando em uma unidade. O modo prefixo incrementa e retorna valor. O modo posfixo retorna valor e incrementa.

Exemplo:

```
/* add_one.c -- incrementing: prefix and postfix */
#include <stdio.h>
int main(void)
{
   int ultra = 0, super = 0;
   while (super < 5)
   {
```

```
        super++;
        ++ultra;
        printf("super = %d, ultra = %d \n", super, ultra);
    }
    return 0;
}
```

Saída:

```
super = 1, ultra = 1
super = 2, ultra = 2
super = 3, ultra = 3
super = 4, ultra = 4
super = 5, ultra = 5
```

```
/* post_pre.c -- postfix vs prefix */
#include <stdio.h>
int main(void)
{
    int a = 1, b = 1;
    int aplus, plusb;
    aplus = a++;      /* postfix */
    plusb = ++b;      /* prefix  */
    printf("a   aplus   b   plusb \n");
    printf("%1d %5d %5d %5d\n", a, aplus, b, plusb);
    return 0;
}
```

Saída:

```
a   aplus   b   plusb
2    1      2    2
```

Expressões e Instruções

Uma expressão é uma combinação de operadores e operandos. A expressão mais simples é uma variável ou constante sem nenhum operador.

4

-6

4+21

a*(b+c/d)/20

q = 5*2

x = ++q / 3

q > 3

Toda expressão tem um valor:

Expressão	Valor
-4 +6	2
c = 3 + 8	11
5 > 3	1
6 + (c = 3 + 8)	17

Instruções: são comandos para o computador. Existem instruções simples e compostas. Instruções simples terminam com um ponto e vírgula, como no exemplo:

Instrução de declaração:

```
int vacas;
```

Instrução de atribuição:

```
vacas = 12;
```

Instrução de chamada de função:

```
printf("%d\n", toes);
```

Instruções compostas, ou blocos, consistem de uma ou mais instruções entre chaves. Exemplo:

```
while (years < 100)
{
   wisdom = wisdom * 1.05;
   printf("%d %d\n", years, wisdom);
   years = years + 1;
}
```

Capítulo 6: Estruturas de controle (Repetição)

Sumário:

- while
- Operadores relacionais
- for
- do while

Revisitando o laço de repetição while

Este exemplo faz uso do valor de retorno do scanf() para finalizar a entrada de dados.

```
/* summing.c -- sums integers entered interactively */
#include <stdio.h>
int main(void)
{   long num;
    long sum = 0L;     /* initialize sum to zero   */
    int status;

    printf("Please enter an integer to be summed ");
    printf("(q to quit): ");
    status = scanf("%ld", &num);
    while (status == 1) /* == means "is equal to"   */
    {
      sum = sum + num;
      printf("Please enter next integer (q to quit): ");
      status = scanf("%ld", &num);
    }
    printf("Those integers sum to %ld.\n", sum);

   return 0;  }
```

Cuidados com a sintaxe

Endentação é uma ajuda visual para o programador e não significa nada para o computador.

- Atenção com a colocação das chaves, um código ruim pode gerar repetições infinitas:

```
#include <stdio.h>
int main(void)
 {
   int n = 0;
   while (n < 3)
       printf("n is %d\n", n);
       n++;
   printf("That's all this program does\n");

   return 0;
}
```

Saída:

```
n is 0
n is 0
```

n is 0
….

- Atenção com a colocação do ponto e vírgula

```
#include <stdio.h>
int main(void)
{
   int n = 0;
    while (n++ < 3);              /* line 7 */
       printf("n is %d\n", n);  /* line 8 */
    printf("That's all this program does.\n");

   return 0;  }
```

Saida:
n is 4
That's all this program does.

Operadores relacionais

Como os laços WHILE quase sempre utilizam expressões de teste que fazem comparações, é importante dar uma olhada mais de perto nas expressões relacionais.

Operador	Significado
<	é menor que
<=	é menor ou igual a
==	é igual a
>=	é maior ou igual a
>	é maior que
!=	é diferente de

Exemplos:

```
while (number < 6)
 {
    printf("Your number is too small.\n");
   scanf("%d", &number);
}
```

```
while (ch != '$')  {
     count++;
     scanf("%c", &ch);
}
```

```
while (scanf("%f", &num) == 1)
    sum = sum + num;
```

Repetição indefinida e repetição com contadores

Podemos utilizar laços WHILE para criar repetições em que não sabemos quantas vezes o laço será repetido ou utilizar contadores para criar laços com um número de repetições predeterminado.

```
// sweetie1.c -- a counting loop
#include <stdio.h>
 int main(void)
 {
   const int NUMBER = 22;
   int count = 1;                        // initialization
   while (count <= NUMBER)               // test
   {
      printf("Be my Valentine!\n");  // action
      count++;                           // update count
   }
    return 0;
 }
```

Laços com um número predeterminado de repetição podem ser criados mais facilmente com um laço FOR.

O laço de repetição FOR

O laço FOR coleta todas as ações (inicialização, teste e atualização) em um lugar.

Sintaxe:

```
for (initialize ; test ; update)
    statement
```

Semântica: O laço é repetido até o teste se tornar falso ou zero.

Exemplos:

```
for (n = 0;  n < 10 ; n++)
     printf(" %d %d\n", n, 2 * n + 1);
```

```
#include <stdio.h>
int main(void)
{
   const int NUMBER = 22;
   int count;
   for (count = 1; count <= NUMBER; count++)
      printf("Be my Valentine!\n");
   return 0;
}
```

```
/* for_cube.c -- using a for loop to make a table of cubes */
#include <stdio.h>
int main(void)
{
   int num;
   printf("    n   n cubed\n");
   for (num = 1; num <= 6; num++)
      printf("%5d %5d\n", num, num*num*num);
   return 0;
```

```
}
```

Saída:

```
n   n ao cubo
1     1
2     8
3    27
4    64
5   125
6   216
```

Incrementando geometricamente ao invés de aritméticamente.

```
/* for_geo.c */
#include <stdio.h>
int main(void)
{
   double debt;
   for (debt = 100.0; debt < 150.0; debt = debt * 1.1)
      printf("Your debt is now $%.2f.\n", debt);
   return 0;
}
```

Mais operadores de atribuição

Total += 20	É o mesmo que total = total + 20
Centavos -= 2	É o mesmo que centavos = centavos- 2
Coelhos *= 2	É o mesmo que coelhos = coelhos * 2
Tempo/= 2.73	É o mesmo que tempo = tempo / 2.73
Redução %=3	É o mesmo que redução = redução % 3

O laço Do while

Sintaxe:

```
do
  statement
while (expression);
```

Semântica: a instrução é repetida até a condição se tornar falsa ou zero.

Exemplo:

```
do
   scanf("%d", &number);
while (number != 20);
```

Capítulo 7: Estruturas de Controle (condicionais e saltos)

Sumário:

- IF, else, switch, continue
- Break, case, default, goto
- Operadores: && || ?:
- Funções getchar(), putchar() e ctype.h

A instrução IF

O IF é chamado de instrução condicional ou instrução de seleção porque ele fornece um ponto em que o programa tem que escolher um caminho a seguir.

Sintaxe:

```
if (expressão)
    instrução
```

Semântica:

Se a expressão for verdadeira, a instrução é executada. Caso contrário ela é pulada. A instrução dentro de um IF pode ser uma instrução simples ou bloco de instruções delimitado por chaves.

Exemplo:

```
// diasfrios.c – acha o percentual de dias frios
#include <stdio.h>
int main(void)
{
   const int FRIO = 0;
   float temperatura;
   int dias_frios = 0;
   int todos_os_dias = 0;
   printf("Entre com a lista de temperatura dos últimos dias (s para sair).\n");
   while (scanf("%f", &temperatura) == 1)
   {
      todos_os_dias++;
      if (temperatura < FRIO)
         dias_frios++;
   }
   if (todos_os_dias != 0)
      printf("De um total de %d dias: %.1f%% foram dias frios.\n",
           todos_os_dias, 100.0 * (float) dias_frios / todos_os_dias);
   if (todos_os_dias == 0)
      printf("Nenhuma temperatura foi lida!\n");
   return 0;
}
```

A instrução IF ELSE

Sintaxe:

```
if (expressão)
    instrução1
else
    instrução2
```

Semântica: se a expressão é verdadeira a instrução1 é executada, senão a instrução2 é executada.

Exemplo:

```
if (todos_os_dias != 0)
   printf("De um total de %d dias: %.1f%% foram dias frios.\n",
        todos_os_dias, 100.0 * (float) dias_frios / todos_os_dias);
else
   printf("Nenhuma temperatura foi lida!\n");
```

As funções getchar() e putchar()

Já sabemos fazer entrada e saída de caracteres com scanf() e printf() com o especificador %c. Mas existe um par de funções especificamente projetadas para trabalhar com caracteres.

A função getchar() não recebe argumentos e retorna o próximo caractere da entrada.

Exemplo: ch = getchar(); // tem o mesmo efeito que scanf(“%c”, &ch);

A função putchar imprime o seu argumento.

Exemplo: putchar(ch); // tem o mesmo efeito que printf(“%c”, ch);

```
/* cifrar.c – altera a entrada, conservando os espaços */
#include <stdio.h>
#define ESPACO ' '          /* quote-espaço-quote */
int main(void)
{
   char ch;
   ch = getchar();          /* read a character        */
   while (ch != '\n')       /* while not end of line    */
   {
      if (ch == SPACE)      /* leave the space         */
         putchar(ch);     /* character unchanged      */
      else
         putchar(ch + 1);  /* change other characters  */

      ch = getchar();       /* get next character      */
   }
   putchar(ch);            /* print the newline       */

   return 0;
}
```

A família de funções do ctype.h

Se quiséssemos mudar o programa anterior para que ele não converta símbolos de pontuação, teríamos que criar muitos IFs. Uma alternativa é usar a função isalpha() definida no arquivo ctype.h.

```
if (isalpha(ch))  // se é uma letra
  putchar(ch+1);
else
  putchar(ch);
```

Funções de teste de caracteres

Nome	Verdadeira se o argumento é
isalnum()	Alfanumérico (alfabético ou numérico)
isalpha()	Alfabético
isblank()	Um caractere em branco (espaço, tab, ou nova linha)
iscntrl()	Um caractere de controle, como um Ctrl+B
isdigit()	Um dígito
islower()	Um caractere minúsculo
ispunct()	Um caractere de pontuação (qualquer coisa que não seja um espaço ou um alfanumérico)
isupper()	Um caractere maiúsculo

Conversão de caracteres

tolower(): Se o argumento é maiúsculo converte para minúsculo, senão retorna o caractere original.

toupper(): Se o argumento é minúsculo converte para maiúsculo, senão retorna o caractere original.

Exemplo:

```
ch = tolower(ch);  // convert ch to lowercase
```

Operadores lógicos

Suponha que exp1 e exp2 são duas expressões então:

exp1 && exp2 é verdadeiro apenas se ambos exp1 e exp2 são verdadeiros.

exp1 || exp2 é verdadeiro se um ou outro é verdadeiro.

!exp1 é verdadeiro se exp1 é falso, e é falso se exp1 é verdadeiro.

Exemplos:

5 > 2 && 4 > 7 é falso porque apenas uma expressão é verdadeira.

5 > 2 || 4 > 7 é verdadeiro porque uma das expressões é verdadeira.

!(4 > 7) é verdadeiro porque 4 não é maior que 7.

```
// Usando o operador lógico AND
#include <stdio.h>
#define PONTO '.'
int main(void)
{
   int ch;
   int contchar = 0;
   while ((ch = getchar()) != PONTO)
   {
      if (ch != '"' && ch != '\'')
         contchar++;
   }
   printf("Existem %d caracteres que não são aspas.\n", contchar);
   return 0;
}
```

O operador condicional ?:

Sintaxe: exp1 ? exp2 : exp3

Semântica: Se a expressão 1 é verdadeira, toda a expressão tem valor o valor da expressão 2. Se a expressão 1 é falsa, toda a expressão resulta no valor da expressão 3.

Exemplo: max = (a > b) ? a : b;

Continue e break

A instrução **continue** faz com que o resto do laço seja pulado. A instrução break faz com que o programa saia do laço corrente e prossiga a execução das instruções fora do laço.

Escolhas múltiplas: switch e break

A expressão de teste do switch deve ter valor inteiro (o que inclui o tipo char). As opções de escolha devem ser de tipo inteiro também, constantes ou expressões contendo apenas constantes inteiras. Variáveis não podem ser usadas num case.

```
switch (expressão inteira)
{
   case constante1:
         instrução   ←optional
   case constante2:
         instrução   ←optional
   default :           ←optional
         instrução   ←optional
}
```

Exemplo:

```
/* Uso da instrução switch */
#include <stdio.h>
#include <ctype.h>
int main(void)
{
   char ch;
   printf("Dê-me uma letra do alfabeto e “);
   printf("eu vou te dar um nome de animal “);
   printf(“começando com essa letra.\n");
   printf("Digite # para finalizar o programa.\n");

   while ((ch = getchar()) != '#')
   {
      if('\n' == ch)
         continue;
      if (islower(ch))
         switch (ch)
         {
            case 'a' :
                printf("anta\n");
                break;
            case 'b' :
                printf("bode\n");
                break;
            case 'c' :
                printf("cavalo\n");
                break;
            case 'd' :
                printf("dromedário\n");
                break;
            case 'e' :
                printf("elefante\n");
                break;
            case 'f' :
                printf("foca\n");
                break;
            default :
                printf("Não sei!\n");
         }
      else
         printf("Eu reconheço apenas minúsculas.\n");

      while (getchar() != '\n')
          continue;     /* pula o resto da entrada até o \n */

      printf("Entre com outra letra ou um #.\n");
   }
   printf("Bye!\n");
   return 0;
}
```

Capítulo 8: Funções

Uma função é um trecho de código projetado para realizar uma tarefa específica, como imprimir texto na tela, ler texto do teclado, calcular o número de caracteres numa string, etc. Uma função em C faz o mesmo papel que sub-rotinas, procedimentos e funções fazem em outras linguagens.

Algumas funções produzem uma ação (printf) outras funções acham um valor para ser usado no programa (strlen). Uma função em C pode produzir ações e retornar valores (scanf).

Por que usar e criar funções? Para poupar trabalho:

- Evitando reescrever código para tarefas repetitivas. Você cria uma função que resolve um problema uma vez e re-usa ela quantas vezes forem necessário.
- Permitindo que o mesmo código seja reusado em vários programas.
- Tornando o seu código mais modular e mais fácil de manter, atualizar e corrigir.

Funções simples

Exemplo: Suponha que você quer fazer um programa para ler uma lista de números, ordenar estes números, achar a média deles e imprimir um gráfico de barras. Você poderia usar um programa assim:

```
#include <stdio.h>
#define TAM 50
int main(void)
{
  float lista[TAM];

  lerLista(lista, TAM);
  ordenar(list, TAM);
  acharMedia(list, TAM);
  graficoBarra(list, TAM);
  return 0;
}
```

Uma função é normalmente vista como uma caixa preta, definida em função da informação que entra e do valor ou ação que ela produz (sua saída).

O que você precisa saber sobre funções:

- Como defini-las apropriadamente
- Como invocá-las (fazer chamadas a funções)
- Como realizar a comunicação entre as funções

Exemplos:

```
/* lethead1.c */
#include <stdio.h>
#define NAME "GIGATHINK, INC."
#define ADDRESS "101 Megabuck Plaza"
#define PLACE "Megapolis, CA 94904"
#define WIDTH 40

void starbar(void);  /* prototype the function */


int main(void)
{
    starbar();
    printf("%s\n", NAME);
    printf("%s\n", ADDRESS);
    printf("%s\n", PLACE);
    starbar();       /* use the function       */
    return 0;
}


void starbar(void)   /* define the function    */
{
    int count;
    for (count = 1; count <= WIDTH; count++)
        putchar('*');
    putchar('\n');
}
```

Funções com Argumentos

Argumentos são variáveis locais às funções, eles não são visíveis fora da função.

```
/* lethead2.c */
#include <stdio.h>
#include <string.h>            /* for strlen() */

#define NAME "GIGATHINK, INC."
#define ADDRESS "101 Megabuck Plaza"
#define PLACE "Megapolis, CA 94904"
#define WIDTH 40
#define SPACE ' '

void show_n_char(char ch, int num);

int main(void)
{
    int spaces;
    show_n_char('*', WIDTH);   /* using constants as arguments */
    putchar('\n');
    show_n_char(SPACE, 12);    /* using constants as arguments */
    printf("%s\n", NAME);
    spaces = (WIDTH - strlen(ADDRESS)) / 2;
                               /* Let the program calculate    */
                               /* how many spaces to skip      */
    show_n_char(SPACE, spaces);/* use a variable as argument   */
    printf("%s\n", ADDRESS);

    show_n_char(SPACE, (WIDTH - strlen(PLACE)) / 2);
                               /* an expression as argument    */
    printf("%s\n", PLACE);
    show_n_char('*', WIDTH);
    putchar('\n');
    return 0;
}



/* show_n_char() definition */
void show_n_char(char ch, int num)
{
    int count;

    for (count = 1; count <= num; count++)
        putchar(ch);

}
```

Funções que retornam valores

```
/* lesser.c -- finds the lesser of two evils */
#include <stdio.h>

int imin(int, int);


int main(void)
{
    int evil1, evil2;

    printf("Enter a pair of integers (q to quit):\n");
    while (scanf("%d %d", &evil1, &evil2) == 2)
    {
        printf("The lesser of %d and %d is %d.\n",
            evil1, evil2, imin(evil1,evil2));
        printf("Enter a pair of integers (q to quit):\n");
    }
    printf("Bye.\n");
    return 0;
}


int imin(int n,int m)
{
    int min;

    if (n < m)
        min = n;
    else
        min = m;

    return min;
}


/* minimum value function, second version */
imin(int n,int m)
{
     return (n < m) ? n : m;
}


/* minimum value function, third version */
imin(int n,int m)
{
    if (n < m)
        return n;
    else
        return m;
}
```

Recursões

```
// factor.c -- uses loops and recursion to calculate factorials

#include <stdio.h>

long fact(int n);
long rfact(int n);

int main(void)
{
    int num;

    printf("This program calculates factorials.\n");
    printf("Enter a value in the range 0-12 (q to quit):\n");
    while (scanf("%d", &num) == 1)
    {
        if (num < 0)
            printf("No negative numbers, please.\n");
        else if (num > 12)
            printf("Keep input under 13.\n");
        else
        {
            printf("loop: %d factorial = %ld\n", num, fact(num));
            printf("recursion: %d factorial = %ld\n", num, rfact(num));
        }
        printf("Enter a value in the range 0-12 (q to quit):\n");
    }
    printf("Bye.\n");
    return 0;
}

long fact(int n)     // loop-based function
{
    long ans;
    for (ans = 1; n > 1; n--)
        ans *= n;
    return ans;
}

long rfact(int n)    // recursive version
{
    long ans;
    if (n > 0)
        ans= n * rfact(n-1);
    else
        ans = 1;
    return ans;
}
```

Separando o programa em vários arquivos

- gcc file1.c fil2.c
- Criação arquivo de cabeçalho (.h)
- Inclusão do cabeçalho em ambos os arquivos (file1.c e file2.c)

Capítulo 9: Vetores e matrizes

Vetores são estruturas que armazenam uma série de elementos do mesmo tipo. Vetores podem ter os mesmos tipos que as variáveis comuns.

Declaração

A declaração de um vetor diz ao compilador quantos elementos o vetor contém e qual o tipo desses elementos.

Exemplo:

```
int main(void)
{
   float candy[365];      /* array of 365 floats */
   char code[12];         /* array of 12 chars   */
   int states[50];        /* array of 50 ints    */
   ...
}
```

Os colchetes indicam que as variáveis são vetores e o número dentro dos colchetes indica a quantidade de elementos do vetor.

Atribuição de valores

O acesso aos elementos do vetor é feito através de um índice que começa em zero. Então `candy[0]` é o primeiro elemento do vetor `candy` que pode ir até `candy[364]` porque ele tem 365 elementos.

Exemplo:

```
int main(void)
{
    int states[50];
    for (int counter = 0; counter < 50; counter++)
        states[counter] = 2 * counter;
}
```

Inicialização

Da mesma forma que variáveis comuns, vetores podem ser inicializados na sua declaração.

Exemplo:

```
/* day_mon1.c -- prints the days for each month */
#include <stdio.h>
#define MONTHS 12

int main(void)
{
    int days[MONTHS] = {31,28,31,30,31,30,31,31,30,31,30,31};
    int index;

    for (index = 0; index < MONTHS; index++)
        printf("Month %d has %2d days.\n", index +1,
               days[index]);
    return 0;
}
```

Podemos definir o tamanho do vetor automaticamente na inicialização.

Exemplo:

```
/* day_mon2.c -- letting the compiler count elements */
#include <stdio.h>
int main(void)
{
    const int days[] = {31,28,31,30,31,30,31,31,30,31};
    int index;

    for (index = 0; index < sizeof days / sizeof days[0]; index++)
        printf("Month %2d has %d days.\n", index +1,
               days[index]);

    return 0;
}
```

Um vetor só pode ser inicializado na sua declaração. Fora da declaração a única forma de dar valores a um vetor é através da atribuição individual de valores aos seus elementos.

```
/* nonvalid array assignment */
#define SIZE 5
int main(void)
{
    int oxen[SIZE] = {5,3,2,8};      /* ok here     */
    int yaks[SIZE];

    yaks = oxen;                     /* not allowed  */
    yaks[SIZE] = oxen[SIZE];         /* invalid      */
    yaks[SIZE] = {5,3,2,8};          /* doesn't work */
}
```

Matrizes

Vetores multidimensionais são chamados de matrizes.

Suponha que se deseja guardar a quantidade de chuva de cada mês durante 5 anos. Podemos ter um vetor de 12 elementos para cada ano. Ou ter uma matriz de ordem 5x12.

Exemplo:

```
float rain[5][12];  // array of 5 arrays of 12 floats
```

Inicialização

A inicialização usa listas de elementos separadas por vírgula.

```
float weekdayrain[2][5] =
{
    {4.3, 4.3, 4.3, 3.0, 2.0},
    {8.5, 8.2, 1.2, 1.6, 2.4}
};
```

Exemplo:

```
/* rain.c  -- finds yearly totals, yearly average, and monthly
                average for several years of rainfall data */

#include <stdio.h>
#define MONTHS 12    // number of months in a year
#define YEARS   5    // number of years of data

int main(void)
{
 // initializing rainfall data for 2000 - 2004
    const float rain[YEARS][MONTHS] =
    {
        {4.3,4.3,4.3,3.0,2.0,1.2,0.2,0.2,0.4,2.4,3.5,6.6},
        {8.5,8.2,1.2,1.6,2.4,0.0,5.2,0.9,0.3,0.9,1.4,7.3},
        {9.1,8.5,6.7,4.3,2.1,0.8,0.2,0.2,1.1,2.3,6.1,8.4},
        {7.2,9.9,8.4,3.3,1.2,0.8,0.4,0.0,0.6,1.7,4.3,6.2},
        {7.6,5.6,3.8,2.8,3.8,0.2,0.0,0.0,0.0,1.3,2.6,5.2}
    };

    int year, month;
    float subtot, total;

    printf(" YEAR    RAINFALL  (inches)\n");
    for (year = 0, total = 0; year < YEARS; year++)
    {
        for (month = 0, subtot = 0; month < MONTHS; month++)
            subtot += rain[year][month];

        printf("%5d %15.1f\n", 2000 + year, subtot);
        total += subtot; // total for all years
     }

    printf("\nThe yearly average is %.1f inches.\n\n", total/YEARS);
    printf("MONTHLY AVERAGES:\n\n");
    printf(" Jan  Feb  Mar  Apr  May  Jun  Jul  Aug  Sep  Oct Nov  Dec\n");

    for (month = 0; month < MONTHS; month++)
    {
        for (year = 0, subtot =0; year < YEARS; year++)
            subtot += rain[year][month];

        printf("%4.1f ", subtot/YEARS);
    }
    printf("\n");

    return 0;
}
```