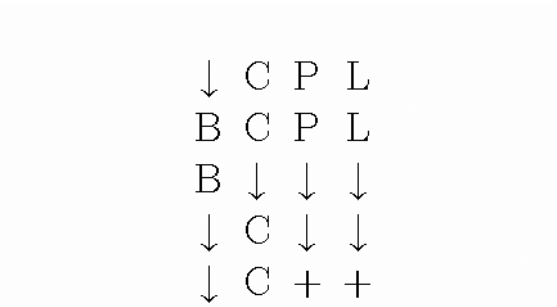


Routo Terada

Depto. de Ciência da Computação
Instituto de Matemática e Estatística
Universidade de São Paulo

Conteúdo

Tabela 1: Lista das 28 palavras-chave reservadas em *C*

auto	double	if	static
break	else	int	struct
case	entry	long	switch
char	extern	register	typedef
continue	float	return	union
default	for	sizeof	unsigned
do	goto	short	while

Tabela 2: Etapas de transformação pelo compilador *C*

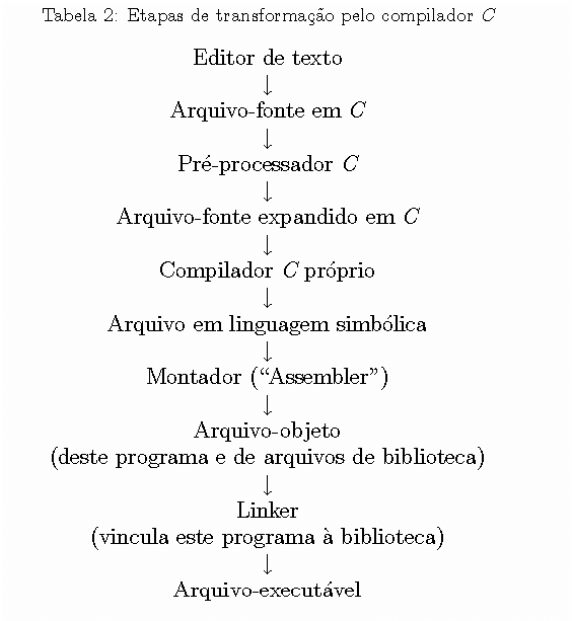


Tabela 3: Tabela de Identificadores

Identificador	Valor	Significado
SHRT_MAX	+32767	valor máximo de short
SHRT_MIN	−32767	valor mínimo de short
INT_MAX	+32767	valor máximo de int
INT_MIN	−32767	valor mínimo de int
LONG_MAX	+2147483647	valor máximo de long
LONG_K	−2147483647	valor mínimo de long
UINT_MAX	65535	valor máximo de unsigned int
ULONG_MAX	4294967295	valor máximo de unsigned long
USHRT_MAX	65535	valor máximo de unsigned short
FLT_MAX	$1E+37$	valor máximo de float
FLT_MIN	$1E-37$	valor mínimo de float

Tabela 4: Ordem de avaliação de operadores

Precedência	Associatividade
`() [] -> .`	$\longrightarrow$
`! ++ -- - (tipo) * & sizeof`	$\longleftarrow$
`* / %`	$\longrightarrow$
`+ -`	$\longrightarrow$
`< <= > >=`	$\longrightarrow$
`== !=`	$\longrightarrow$
`&&`	$\longrightarrow$
`||`	$\longrightarrow$
`?:`	$\longleftarrow$
`= += -= *= /= %=`	$\longleftarrow$

Em caso de dúvida *sempre* inclua parênteses explicitando a ordem desejada de avaliação.

Os operadores de comparação – `==` *igual*, `!=` *diferente*, `<` *menor*, `<=` *menor-ou-igual*, etc. – são avaliados com os valores 1 (verdadeiro) ou 0 (falso), conforme a igualdade ou desigualdade de seus operandos. Observe a precedência dos operadores lógicos – `!` *não*, `&&` *e*, `||` *ou* – e a dos operadores de comparação, que tornam desnecessário sobrecarregar de parênteses muitas expressões.

Em expressões condicionais qualquer valor não nulo é equivalente a verdadeiro.

Assim, `!v` vale 1 se e só se `v` vale 0. E `v&&w` vale 1 se e só se `v` e `w` não valem 0. E `v||w` vale 0 se e só se `v` e `w` valem 0.

Alguns exemplos de expressões condicionais: `(a<=10.2) && chovendo || (i>j+1)`, `!acabou || j==(M+51)`, `acabou && j!=(M+51)`.

A expressão a%b, a *módulo* b, vale o resto da divisão de a por b. Assim, (13%5==3) vale 1. A expressão condicional (condicao?a:b) tem valor a se a condicao é verdadeira, e valor b caso contrário, assim (a>b?a:b) vale $max(a, b)$.

Atribuições do tipo y+=x; ou y*=x; são apenas abreviações para as atribuições y=y+x; e y=y*x;. Os operadores de *incremento* e *decremento*, ++ e -- , aumentam ou diminuem de uma unidade o valor de uma variável inteira. Uma variável pós-fixada pelo operador de incremento, k++ , é primeiro avaliada e depois incrementada; uma variável pré-fixada é primeiro incrementada e depois avaliada. Assim, são equivalentes os comandos abaixo que estão na mesma linha.

```
y=++k;          { k+=1; y=k; }
y=k++;          { y=k; k+=1; }
```

O ponto-e-vírgula (;) ; é o *terminador* de comandos simples (e não um *separador* como em outras linguagens). O caractere nova-linha (\n), o espaço-em-branco e a marca-de-tabulação (\t), todos estes chamados *brancos*, são apenas separadores, de modo que:

Declaração de variáveis

As variáveis a serem usadas num programa precisam ser *declaradas*, em princípio para que o compilador lhes reserve o espaço necessário. Assim, as sentenças

```
int i, j, k;
float x[10], y, z;
```

declaram 3 variáveis de *tipo* inteiro, `i`, `j` e `k`, duas variáveis de *tipo* flutuante, `y` e `z`, e um *array* ou vetor com 10 posições de *tipo* flutuante. Um nome de variável pode ser qualquer seqüência de letras e dígitos, começando por uma letra. Letras são as maiúsculas, as minúsculas e a sublinha "_". Arrays em *C* sempre são indexados a partir de 0, de modo que com a declaração acima podemos nos referir a `x[0]`, `x[1]`, até `x[9]`, mas não a `x[10]`. Mais detalhes sobre arrays na Seção 9.

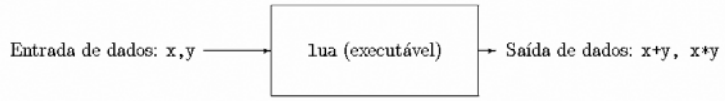
```
printf("y vale %f e i vale %d \n", y, i);
```

que, ao ser executado, provoca a saída do tipo `y vale 1.55 e i vale 1901` no *dispositivo padrão de saída*, em geral a tela do terminal. \n que ocorre na funo, entre aspas, é para provocar um *pulo* para uma nova linha. No lugar de y ou de i pode-se colocar uma expressão aritmética, como ilustrado a seguir.

Um programa em *C* é precedido por `main(){` e terminado por `}`. Por exemplo, o programa abaixo "lê" dois números flutuantes, e exibe a sua soma e o seu produto, precedidos de explicações do tipo " ... vale ...".

```
main(){
  float x, y;
  scanf("%f %f", &x, &y);
  printf("A soma vale %f, e o produto vale %f",
          x+y, x*y);
  }
```

Figura 1: Ilustração de entrada e saída de dados



Notas de Aula de C (Routo Terada) 10

Figura 2: `for`, `while`, e `do-while`

```
main(){                  *   main(){          *   main(){
  int i, j;              *     int i, j;      *     int i, j;
  i=0;                   *     i=0; j=1;      *     i=0; j=0;
  for(j=1; j<11; j=j+1) *     while(j<11){   *     do{
    i=i+j;               *       i=i+j;       *        j= j+1;
  printf("%d",i);        *       j=j+1;       *        i= i+j;
}                        *     }              *     }while(j<10);
                         *     printf{"%d",i);*     printf("%d",i);
                         *   }                *   }
                         *                    *
```

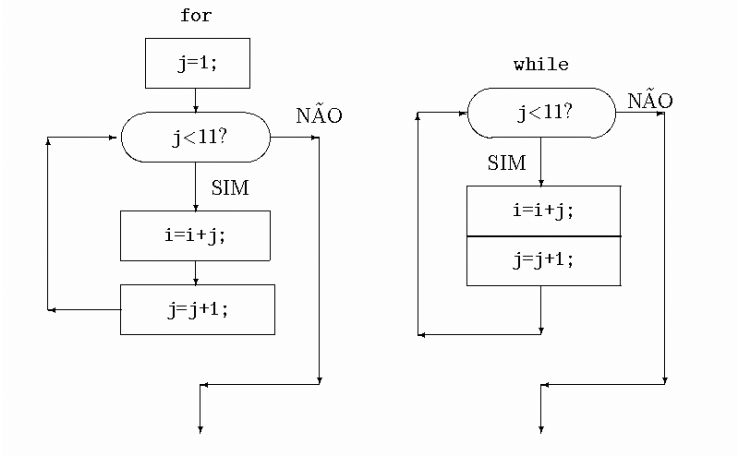
Os três programas na Figura 2 calculam $i = \sum_{j=1}^{10} j$. No primeiro programa o comando `for` significa que `j`, a *variável* do `for`, é *inicializada* com o valor 1 (`j=1;`). O *comando* ou *laço* do `for` (i.e., `i=i+j;`) será executado com argumento `j` enquanto a *condição* do `for`, `j<11;`, for verdadeira. Após a execução do laço, será realizada a *operação* do *for*, que atribuirá à variável `j` um novo valor, no caso `j+1`. O *laço* se repete até que a condição do `for` se verifique falsa, quando o laço é interrompido (o comando do `for` não é executado), e o programa prossegue executando o comando seguinte ao `for`, que é `printf(...`.

Esta é a forma em *C* de se dizer o equivalente a:

---

para `j=1`, faça `j=i+j;` incrementando `j` de 1 em 1 até `j` valer 10.

Figura 3: Ilustração de for e while

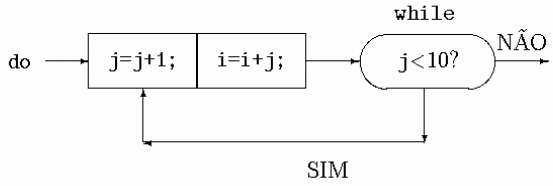



```
for(inicializacao; condicao; operacao){
  comando1; comando2; comando3;
}
comando-seguinte;
```

Figura 2: for, while, e do-while

```
main(){                    *   main(){           *   main(){
  int i, j;                *     int i, j;       *     int i, j;
  i=0;                     *     i=0; j=1;       *     i=0; j=0;
  for(j=1; j<11; j=j+1)    *     while(j<11){    *     do{
    i=i+j;                 *       i=i+j;        *       j= j+1;
  printf("%d",i);          *       j=j+1;        *       i= i+j;
}                          *     }               *     }while(j<10);
                           *     printf{"%d",i);*      printf("%d",i);
                           *   }                 *   }
                           *                     *
```

Figura 4: Ilustração de do-while

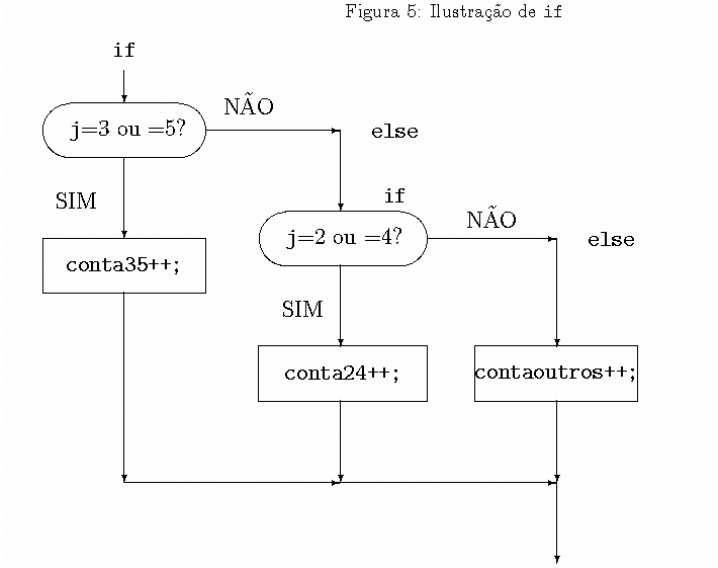


6 Comandos Condicionais `if` e `switch`

O fluxo de execução do programa pode ser controlado também com comandos como `if` e `switch`, exemplificados abaixo. O primeiro programa ilustra o uso dos comando `if` e resolve o seguinte: dos números de 1 a 100, conta quantos terminam em 3 ou 5, quantos terminam em 2 ou 4, e quantos são os outros.

```
main(){
  int i,j, conta35, conta24, contaoutros;
  for(i=1; i<101; i++){
    j=i-10*(i/10);
    if(j==3||j==5)
      conta35++;
    else
      if(j==2||j==4)
           conta24++;
      else contaoutros++;
    }
  printf("conta35 = %d, conta24 = %d, outros = %d",
          conta35, conta24, contaoutros);
  }
```

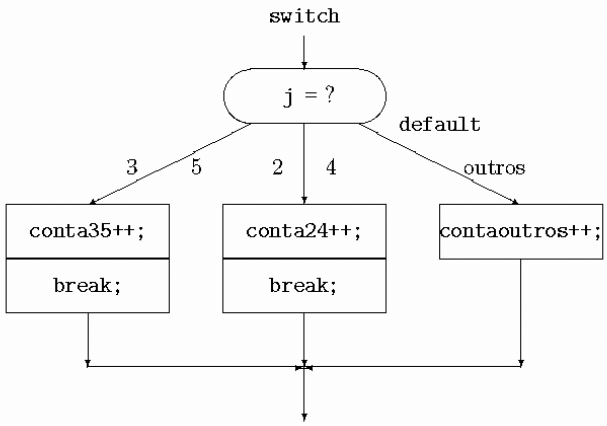
Figura 5: Ilustração de if



O comando `if(condicao)` suprime a execução do comando seguinte caso a sua condição seja falsa. O comando *if-else*, `if(condicao) com1; else com2;` executa o comando `com1` se a condição do `if` for verdadeira, e executa o comando `com2` caso contrário.

O programa a seguir resolve o *mesmo* problema anterior (resolvido com o comando `if`), agora com o uso do comando `switch`.

```
main(){
  int i,j, conta35, conta24, contaoutros;
  for(i=1; i<101; i++){
    j=i-10*(i/10);
    switch(j){
      case 3: case 5: conta35++;
                      break;
      case 2: case 4: conta24++;
                      break;
      default: contaoutros++;
    }
  printf("conta35 = %d, conta24 = %d, outros = %d",
         conta35, conta24, contaoutros);
  }
```




```
switch(expressao){
  case expressao-const1: comandos-1; break;
  case expressao-const2: comandos-2; break;
  ...
  default: comandos;
  }
```

Notas de Aula de C (Routo Terada) 17

Figura 7: Ilustração de continue e break

```
main(){                    *   main(){              * main(){
  int i, j;                *     int i, j;          *   int i,j;
  i=0;                     *     i=0; j=1;          *   i=0;
  for(j=1; j<11; j=j+1){   *     while(j<11){       *   for(j=1;j<11;j=j+2)
    if(j%2==0) continue;   *       if(j==6) break;  *     i=i+j;
    else i=i+j;            *       else {           *   printf("%d",i);
    }                      *         i=i+j;         * }
  printf("%d",i);          *         j=j+1;
}                          *         }
                           *     }
                           *     printf{"%d",i);
                           *   }
```

7 Comandos continue e break

Se dentro de um laço ocorrer um comando continue, a execução do resto do laço é suprimida, e volta para o *início* do laço. Por outro lado, se ocorrer um comando break, a execução de todo o laço (mais interno) é interrompida. Nos exemplos a seguir ilustramos o uso destes comandos.

O programa à esquerda na Figura 7 calcula $i = \sum_{j=1}^{10} j$, mas só para j ímpar, pois quando j é par, o comando continue é executado. O programa à direita na Figura 7 faz o mesmo cálculo, só que com j=j+2; no for. No programa ao meio, o laço do while é executado completamente apenas para j igual a $1, 2, \ldots, 5$ pois o comando break é executado para o valor de j igual a 6.

8 Pré-Processamento

```
#define N 10
#define QUAD(x) ( x ) * ( x )
#define ABS(x) ((x>=0)?(x):-(x))
#define MAX(a,b) ((a>b)?(a):(b))
```

```
main(){
   ...
  while(x<=N){
     ...
    x=QUAD(MAX(x+1,y) ;
     ...
   }
}
```

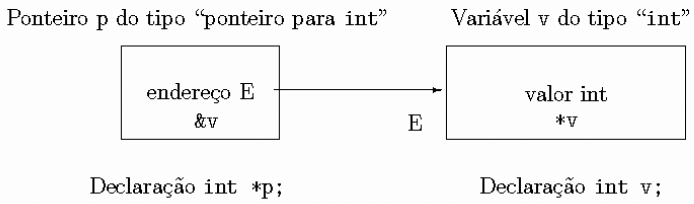
```
**********************************************************

main(){
   ....
  while(x<=10){
     ...
    x=( ((x+1)>(y)?(x+1):(y)) ) * ( ((x+1)>(y)?(x+1):(y)) ) ;
     ...
   }
}
```

Comentários

Comentários em *C* têm seu começo e seu fim marcados pelos dígrafos /* e */. Em *C* comentários NÃO podem ser aninhados.

Figura 8: Ilustração de ponteiro após execução de p= &v;



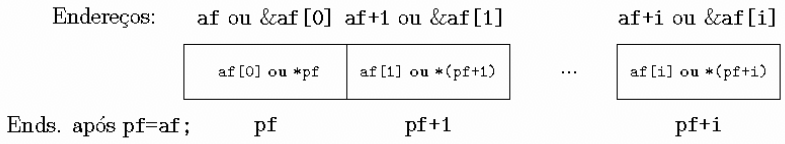
Para os exemplos dos próximos parágrafos considere as declarações:

```
int i, j, *pi, ai[10]; float x, y, *pf, *qf, af[10];
```

onde declaramos `i` uma variável inteira, `pi` um ponteiro para inteiros, `ai` um array de inteiros de 10 elementos, `x` um variável de ponto flutuante, e `pf` um apontador para variáveis em ponto flutuante, e `af` um array de 10 variáveis de ponto flutuante.

Nunca esqueça de inicializar qualquer ponteiro. Só a sua declaração não implica em inicialização!

Figura 9: Ilustração de array e ponteiros



Strings são arrays de caracteres cujo término é marcado pelo caractere nulo, \0. Uma forma alternativa de inicializar strings é trocar a lista de códigos de caracteres pela seqüência dos caracteres entre aspas duplas, sem incluir explicitamente o terminador \0, como na quarta declaração abaixo, que é equivalente à terceira.

```
int a[5] = {1, 2, 3, 4, 5 };
int a[] = {1, 2, 3, 4, 5, };
char nome[] = {'D', 'i', 'a', '-', 'D', '\0' };
char nome[] = "Dia-D";
int ident2[][] = {{1,0},{0,1}};
int ident2[2][2] = {1,0,0,1};
```

```
main(){
  char nome[80];
  int i, nletras_a;
  printf("Digitar um nome e teclar ENTER (max de 80 letras)");
  scanf("%s", nome);
  i=0; nletras_a=0;
  while(nome[i]!='\0'){
    if(nome[i] == 'a') nletras_a++;
    i++;
    }
  printf("%d letras 'a' ocorrem no nome", nletras_a);
  }
```

Funções. Parâmetros e Argumentos

```
float potencia( float x, int n);  /* declaracao da funcao */

main(){
 int i; float y, z;
 i=3; y=2.0;
 /* nesta linha tem-se uma invocacao da funcao potencia */
 z = potencia(y,i);
 printf("%f", z);
}  /* programa principal */

float potencia( float x, int n ){
 int j; float w;
 if(n<0)
   return 0.0;
 else
   w = 1.0;
   for(i=1; i<=n; i++)
     w = w*x;
   return w;
} /* definicao da funcao */
```

```
nome-da-funcao(lista-de-parametros){corpo-da-funcao}.
```

```
void trocanao( float x, float y);
void trocasim( float *px, float *py);
main(){
  float x=1.0, y=2.0;
  printf("%f %f",x,y);     /* 1.0 2.0 */
  trocanao(x,y);           /* 2.0 1.0 */
  printf("%f %f",x,y);     /* 1.0 2.0 */
  trocasim(&x, &y);        /* 2.0 1.0 */
  printf("%f %f",x,y);     /* 2.0 1.0 */
}

void trocanao(float x, float y){
  float aux;
  aux=x; x=y; y=aux;
  printf("%f %f",x,y);
}

void trocasim(float *px, float *py)
  float aux;
  aux=*px; *px=*py; *py=aux;
  printf("%f %f",*px,*py);
}
```

```
// Funções com matriz
#include <stdio.h>
int func1(int A[20][20], int j){
int k;
j=j+1;
printf("dentro da func1 A[0][0]= %d, j= %d \n", A[0][0], j);
return j;
}
int func2(int A[20][20], int *pj){
int k;
*pj=*pj+1;
printf("dentro da func2 A[0][0]= %d, pj= %d, *pj= %d\n", A[0][0],
pj, *pj);
return *pj;
}
main(){
int i,j,k;
int A[20][20];
A[0][0]=1996;
j=0;
printf("func1 return= %d \n", func1(A, j));
printf("apos func1 j= %d\n", j);
printf("func2 return= %d \n", func2(A, &j));
printf("apos func2 j= %d\n", j);
printf(" func2 return= %d \n", func2(&A[0], &j));
printf(" apos func2 j= %d\n", j);
}
```

Saida resulta em:

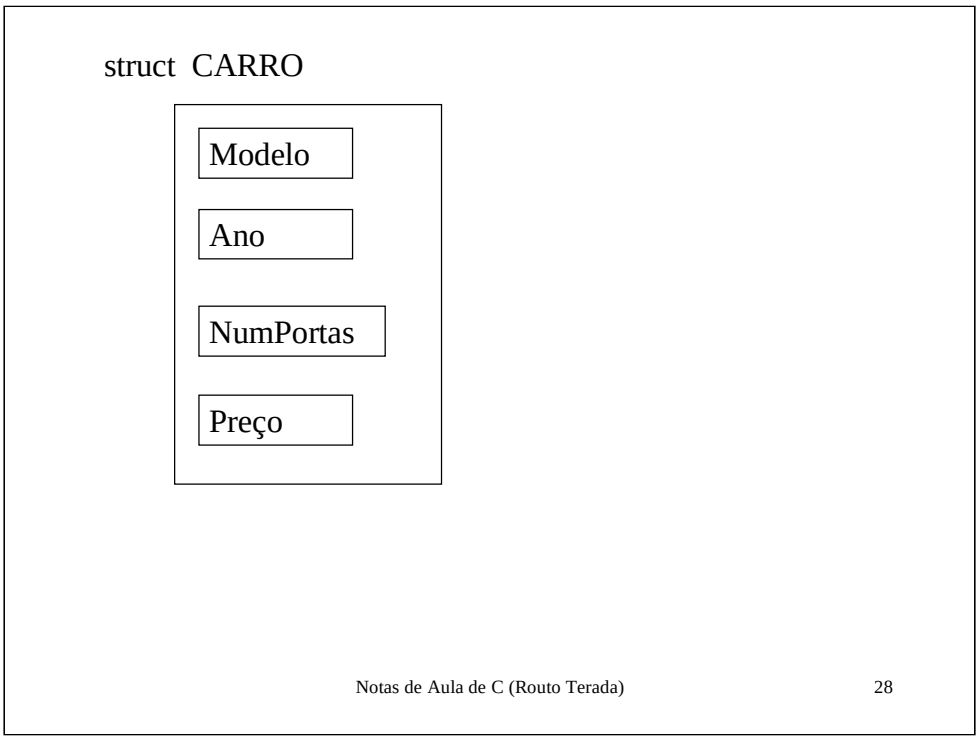
```
dentro da func1 A[0][0]= 1996, j= 1
func1 return= 1
apos func1 j= 0
dentro da func2 A[0][0]= 1996, pj= -268437496, *pj= 1
func2 return= 1
apos func2 j= 1
dentro da func2 A[0][0]= 1996, pj= -268437496, *pj= 2
 func2 return= 2
 apos func2 j= 2
```

Exemplos de Programa em C

```
/*
 *  BUSCASEQ.C--Exemplo de Busca Sequencial
 *  Problema: dada uma sequencia de N numeros reais (do tipo float)
 *  e um numero real X, verificar se X ocorre na sequencia
 */
#include <stdio.h>
#define Nmax 101           /* Numero maximo de elementos em R[] */
void main()
{
   int N,   /* numero de elementos em R[] */
       i;
   float R[Nmax],  /* vetor de N elementos */
         X;        /* elemento a ser procurado em R[] */
   /*
    *  Leitura dos parametros
    */
   printf("\nDigitar o numero de elementos de R[] -> ");
   scanf("%d", &N);
   printf("\nN = %d\n", N);
   if(N>Nmax-1){
     printf("\nNumero maximo de elementos foi excedido\n");
     exit(0);
     } /*end if */
```

```
   printf("Digitar os elementos de R[] -> ");
   for(i=0; i<N; i=i+1)
     scanf("%f", &R[i]);
   printf("\n");
   for(i=0; i<N; i=i+1)
     printf("R[%d] = %f", i, R[i]);
   printf("\nDigitar o elemento X a ser procurado em R[] -> ");
   scanf("%f", &X);
   printf("\nX = %f", X);
   /*
    * Busca Sequencial de X em R[], em tempo proporcional a N
    */
   R[N] = X;   /* valor X como ''sentinela'' apos ultimo
                  elemento valido de R[] */
   i=0;
   while(R[i] != X)
     i= i+1;
   /*
    *  Dar resposta final
    */
   if( i != N )
     printf("\n--- X = %f ocorre em R[]\n", X);
   else
       printf("\n --- X = %f nao ocorre em R[]\n", X);
} /* end main */
```

Notas de Aula de C (Routo Terada) 27


struct CARRO
Modelo
Ano
NumPortas
Preço
Notas de Aula de C (Routo Terada)
28

```
// programa de carros em struct
//
#include <stdio.h>
void main(){
 int j;
 struct CARRO {       // CARRO e' o nome do tipo de estrutura
   char *Modelo;
   int Ano;
   int Km;  // quilometragem atual
   char *Fabricante; // nome do fabricante
   char *Cor;
   int NumPortas; // numero de portas
   int GasOuAlc;  // 1==gasolina, 2==alcool
   int Preco;  // preco atual de mercado, em reais
 };   // note o ; aqui

struct CARRO meucarro, carronovo, carrodopai;
          // 3 vars do tipo CARRO
 meucarro.Modelo= "Astra";
 meucarro.Ano= 2000;
 meucarro.Km= 31;
 meucarro.Fabricante= "GM";
 meucarro.Cor= "verde";
 meucarro.NumPortas= 2;
 meucarro.GasOuAlc= 1;
 meucarro.Preco= 32000;
 printf("modelo %s\n", meucarro.Modelo);

}
```

```
// programa de estrutura simples (Routo Terada)
//
#include <stdio.h>
void main(){
 int j;

 struct PESSOA{       // PESSOA e' o nome do tipo de estrutura
   char Nome[80];
   int Altura;  // em centimetros
   int Peso;    // em quilos
 };   // note o ; aqui

 struct PESSOA marcio, alcides, maria;  // 3 vars do tipo PESSOA
 struct PESSOA amigos[20];  // grupo de 20 pessoas

 marcio.Altura=170;
 printf("Altura: %d\n", marcio.Altura);

 marcio.Nome[0]='M';
 marcio.Nome[1]='a';
 marcio.Nome[2]='r';
 marcio.Nome[3]='c';
 marcio.Nome[4]='i';
 marcio.Nome[5]='o';
 marcio.Nome[6]='\0';
 printf("Nome: %s\n", marcio.Nome);
 sprintf(marcio.Nome, "Marcio Silva"); // exemplo de sprintf
 printf("Nome: %s\n", marcio.Nome);
 amigos[2].Peso= 72;
 printf("Peso de um amigo: %d\n", amigos[2].Peso);
}
```

```
Saída resulta em:

Altura: 170
Nome: Marcio
Nome: Marcio Silva
Peso de um amigo: 72
```

```
// programa de carros em struct (Routo Terada)
//
#include <stdio.h>
void main(){
 int j;
 struct CARRO {       // CARRO e' o nome do tipo de estrutura
   char *NomeDono;
   struct {
     char *RuaeNum; // Nome da rua e numero
     char *Bairro;
     char *CEP;
     char *Cidade;
     char *Telefone;
     } Endereco;     // nome de um componente da estrutura
   char *Modelo;
   int Ano;
   int Km;  // quilometragem atual
   char *Fabricante; // nome do fabricante
   char *Cor;
   int NumPortas; // numero de portas
   int GasOuAlc;  // 1==gasolina, 2==alcool
   int Preco;  // preco atual de mercado, em reais
   char *Chapa;
 };   // note o ; aqui
struct CARRO meucarro, carronovo, carrodopai;
                    // 3 vars do tipo CARRO
 meucarro.Endereco.RuaeNum= "Rua Padre Anchieta, 1011";
 meucarro.Modelo= "Astra";
 meucarro.Ano= 2000;
 meucarro.Km= 31;
 meucarro.Fabricante= "GM";
```

31

```
struct CARRO meucarro, carronovo, carrodopai;
                    // 3 vars do tipo CARRO
 meucarro.Endereco.RuaeNum= "Rua Padre Anchieta, 1011";
 meucarro.Modelo= "Astra";
 meucarro.Ano= 2000;
 meucarro.Km= 31;
 meucarro.Fabricante= "GM";
 meucarro.Cor= "verde";
 meucarro.NumPortas= 2;
 meucarro.GasOuAlc= 1;
 meucarro.Preco= 32000;
 printf("Nome do meu modelo: %s\n", meucarro.Modelo);
          // mostra Astra
 printf("Nome da minha rua e numero: %s \n",
               meucarro.Endereco.RuaeNum);
}
```

```
#include <stdio.h>
void main(){
 int j;
 typedef struct { // typedef aqui
   char *NomeDono;
   char *Modelo;
   int Ano;
   int Km;  // quilometragem atual
   char *Fabricante; // nome do fabricante
   char *Cor;
   int NumPortas; // numero de portas
   int GasOuAlc;  // 1==gasolina, 2==alcool
   int Preco;  // preco atual de mercado, em reais
   char *Chapa;
 } CARRO;          // aqui vai nome do tipo que e' CARRO

 CARRO meucarro, carronovo, carrodopai;  // 3 vars do tipo CARRO
 // note que acima a palavra struct nao ocorre antes de CARRO

 meucarro.Modelo= "Astra";
 meucarro.Ano= 2000;
 meucarro.Km= 31;
 meucarro.Fabricante= "GM";
 meucarro.Cor= "verde";
 meucarro.NumPortas= 2;
 meucarro.GasOuAlc= 1;
 meucarro.Preco= 32000;
 printf("Nome do meu modelo: %s\n", meucarro.Modelo);
              // mostra Astra

}                                                                    33
```

```
// programa com
// struct de STACK (Routo Terada)
#include <stdio.h>
#define Nmax 50
// a seguir typedef global
typedef struct car{ // typedef aqui com nome car
   char *Modelo;
   int Preco;  // preco atual de mercado, em reais
 } CARRO;  // nome do tipo

typedef struct s{
  CARRO carro[Nmax];
  int topo;
 } STACK;        // nome do tipo

void ImprTopo(STACK *frota){ //
 // imprime os valores de um objeto carro no topo do stack
 printf("  Dentro de ImprTopo, Modelo do carro: %s \n",
     ((*frota).carro[(*frota).topo]).Modelo);
 printf("  Idem, Preco: %d \n",
((*frota).carro[(*frota).topo]).Preco);
} // fim ImprTopo

void ImprCarro(CARRO *pcar){ //
 // imprime os valores de um carro apontado por pcar
 printf("  Dentro de ImprCarro, Modelo do carro: %s \n",
               pcar->Modelo);
 printf("  Idem, Preco: %d \n", pcar->Preco);
} // ImprCarro
```

34

```
int PushDown(STACK (*frota), char *modelo, int preco){
  // empilha modelo e preco de um carro no topo do stack
  if(((*frota).topo)<Nmax){
    (*frota).topo=(*frota).topo+1;
    (*frota).carro[(*frota).topo].Modelo=modelo;
    (*frota).carro[(*frota).topo].Preco=preco;
    return 1; // retorna 1 se esta' tudo OK
  }
  else return -1; // retorna -1 se deu errado
 } // fim PushDown

CARRO * PopUp(STACK (*frota)){
  // desempilha um carro do topo do stack,
  // retorna apontador para este carro CARRO *pcar;
  if((*frota).topo==(-1)) {return NULL;}
          // retorna NULL se nao havia carro
  else{
    pcar = &((*frota).carro[(*frota).topo]);
    (*frota).topo= (*frota).topo-1;
    return pcar;
  }
 } // fim PopUp

int Vazio(STACK (*frota)){
  //
  if((*frota).topo==(-1)) {return -1;}
          // retorna -1 se pilha vazia
  else return 1;
 } // fim Vazio
```

35

```
void main(){
CARRO *pcar;
STACK frota;
int retorno;
frota.topo=-1;
  retorno= PushDown(&frota, "Astra A", 666); // empilha um carro
    if(retorno!=(-1)) ImprTopo(&frota);
  retorno= PushDown(&frota, "Astra B", 777);
                                  // empilha outro carro
    if(retorno!=(-1))ImprTopo(&frota);
  pcar = PopUp(&frota); // desempilha do topo
    if(pcar!=NULL) ImprCarro(pcar);
  retorno= PushDown(&frota, "Astra C", 888);
                               // empilha outro carro
    if(retorno!=(-1))ImprTopo(&frota);

  if( Vazio(&frota) == 1) printf("Pilha nao esta' vazia\n");
  else printf("\nPilha esta' vazia\n");

  //  Dentro de ImprTopo, Modelo do carro: Astra A
  //  Idem, Preco: 666
  //  Dentro de ImprTopo, Modelo do carro: Astra B
  //  Idem, Preco: 777
  //  Dentro de ImprCarro, Modelo do carro: Astra B
  //  Idem, Preco: 777
  //  Dentro de ImprTopo, Modelo do carro: Astra C
  //  Idem, Preco: 888
  //Pilha nao esta' vazia

} // fim main
```

36