03	+3110	(Subtrair B)
04	+4107	(Desvio negativo para 07)
05	+1109	(Escrever A)
06	+4300	(Terminar)
07	+1110	(Escrever B)
08	+4300	(Terminar)
09	+0000	(Variavel A)
10	+0000	(Variavel B)

Esse programa em LMS lê dois números do teclado, determina o maior valor e o imprime. Observe o uso da instrução **+4107** como uma transferência condicional de controle, muito parecida com a instrução **if da** linguagem C. Agora escreva programas LMS para realizar as seguintes tarefas.

a) Use um loop controlado por um valor sentinela para ler 10 números positivos e calcular e imprimir sua soma.

b) Use um loop controlado por contador para ler sete números, alguns positivos e outros negativos, e calcule e imprima sua média.

c) Leia uma série de números e determine e imprima o maior deles. O primeiro número lido indica quantos números devem ser processados.

7.19 *(Um Simulador de Computador)* **À** primeira vista pode parecer chocante, mas neste problema você vai construir seu próprio computador. Não, você não estará unindo componentes. Em vez disso, você usará a poderosa técnica de *simulação baseada em software* para criar um *modelo de software* do Simpletron. Você não ficará desapontado. Seu simulador do Simpletron transformará seu computador em um Simpletron e você poderá realmente executar, testar e depurar os programas em LMS escritos no Exercício 7.18. Quando seu simulador Simpletron for executado, ele deve começar imprimindo:

***** Bem vindo ao Simpletron! *****
***** Por favor digite uma instrução (ou palavra *****
***** de dados) de seu programa por vez. Digitarei o *****
***** numero da posição e um ponto de interrogação *****
***** (?). Digite então a palavra para aquela posição *****
***** Digite o valor sentinela -99999 para encerrar a *****
***** digitação de seu programa. *****

Simule a memória do Simpletron com um array unidimensional **memória** com 100 elementos. Agora admita que o simulador está sendo executado, e vamos examinar o diálogo quando entrarmos com o programa do Exemplo 2 do Exercício 7.18:

00 ? +1009
01 ? +1010
02 ? +2009
03 ? +3110
04 ? +4107
05 ? +1109
06 ? +4300
07 ? +1110
08 ? +4300
09 ? +0000
10 ? +0000
11 ? -99999
***** Carregamento do programa concluido *** *** Inicio da execução do programa *****

O programa em LMS foi colocado (ou carregado) no array **memória.** Agora o Simpletron executa seu programa em LMS. A execução começa com a instrução no local **00** e, como o C, continua seqüencialmente, a menos que seja dirigido para alguma outra parte do programa por meio de uma transferência de controle

Use a variável **acumulador** para representar o registro acumulador. Use a variável **contadorInstrucao** para controlar o local da memória que contém a instrução que está sendo realizada. Use a variavel **codigoOperacao** para indicar a operação que está sendo realizada atualmente, i.e., os dois dígitos da esquerda, na palavra de instrução. Use a variável **operando** para indicar o local na memória no qual a instrução atual é aplicada. Dessa forma, **operando** compreende os dois dígitos da direita da instrução que está sendo realizada atualmente. Não execute instruções diretamente da memória. Em vez disso, transfira a próxima instrução a ser realizada da memória para a variável chamada **registrolnstrucao.** A seguir, "apanhe" os dois dígitos da esquerda e os coloque em **codigoOperacao** e "apanhe" os dois dígitos da direita e coloqueos em **operando.**

Quando o Simpletron iniciar a execução, os registros especiais são inicializados como se segue:

acumulador +0000
contadorlnstrucao 00
registrolnstrucao +0000
codigoOperacao 00
operando 00

Agora vamos "percorrer" a execução da primeira instrução LMS, +100 9 no local 00 da memória. Isso é chamado *ciclo de execução de instruções.*

O contadorlnstrucao nos informa o local da próxima instrução a ser realizada. Vamos *buscar* o conteúdo de tal local da memória usando a instrução C

registrolnstrucao = memória[contadorlnstrucao];

O código da operação e o operando são extraídos do registro de instruções pelas instruções

codigoOperacao = registrolnstrucao / 100; operando = registrolnstrucao **%** 100;

Agora o Simpletron deve determinar que o código da operação é realmente *ler* (e não *escrever, carregar* etc). Uma estrutura switch reconhece a diferença entre as doze operações da LMS.

Na estrutura switch, o comportamento das várias instruções LMS é simulado da maneira que se segue (deixamos as outras a cargo do leitor):

ler. scanf("%d", &memoria[operando]);
carregar. acumulador = memória[operando];
adicionar. acumulador += memória[operando];

Várias instruções de desvios: Serão vistas brevemente. *terminar:* Esta instrução imprime a mensagem

*** Execução do Simpletron concluida ***

e então imprime o nome e o conteúdo de cada registro assim como o conteúdo completo da memória. Tal saída é chamada freqüentemente *despejo (descarga* ou *dump) do computador* (não, um despejo de computador não é jogar fora um computador velho). Para ajudá-lo a programar a função de despejo, o formato de um exemplo de despejo é mostrado na Fig. 7.32. Observe que um despejo depois da execução de um programa Simpletron mostraria os valores reais das instruções e os valores dos dados no momento em que a execução terminasse.

Vamos continuar com a execução da primeira instrução de nosso programa, ou seja, +1009 no local 00. Como indicamos, a instrução switch simula isso realizando a instrução em C

scanf(***"%d",*** &memoria[operando]);

Deve ser mostrado um ponto de interrogação (?) na tela antes de scanf ser executado, para pedir a digitação do usuário. O Simpletron espera que o usuário digite um valor e então pressione a tecla *Return.* O valor é então lido e colocado no local 09.

REGISTROS:
acumulador +0000
contadorlnstrucao 00
registrolnstrucao +0000
codigoOperacao 00
operando 00

MEMÓRIA:

	0	1	2	3	4	5	6	7	8	9
0	+0000	+0000	+0000	+0000	+0000	+0000	+0000	+0000	+0000	+0000
10	+0000	+0000	+0000	+0000	+0000	+0000	+0000	+0000	+0000	+0000
20	+0000	+0000	+0000	+0000	+0000	+0000	+0000	+0000	+0000	+0000
30	+0000	+0000	+0000	+0000	+0000	+0000	+0000	+0000	+0000	+0000
40	+0000	+0000	+0000	+0000	+0000	+0000	+0000	+0000	+0000	+0000
50	+0000	+0000	+0000	+0000	+0000	+0000	+0000	+0000	+0000	+0000
60	+0000	+0000	+0000	+0000	+0000	+0000	+0000	+0000	+0000	+0000
70	+0000	+0000	+0000	+0000	+0000	+0000	+0000	+0000	+0000	+0000
80	+0000	+0000	+0000	+0000	+0000	+0000	+0000	+0000	+0000	+0000
90	+0000	+0000	+0000	+0000	+0000	+0000	+0000	+0000	+0000	+0000

Fig. 7.32 Um exemplo de despejo.

Nesse momento termina a simulação da primeira instrução. Tudo que resta é preparar o Simpletron para executar a próxima instrução. Como a instrução que acabou de ser realizada não foi uma transferencia de controle, precisamos apenas incrementar o registro do contador de instruções como se segue:

++contadorInstrução;

Isso completa a execução simulada da primeira instrução. O processo inteiro (i.e., o ciclo de execução instruções) começa de novo com a busca da próxima instrução a ser executada.

Agora vamos examinar como as instruções de desvios — ou transferências de controle — são simuladas Tudo que precisamos fazer é ajustar apropriadamente o valor no contador de instruções. Assim, a instrução de desvio incondicional (4 0) é simulada dentro da estrutura switch da seguinte forma

contadorlnstrucao = operando;

A instrução condicional "desvio se o acumulador for zero" é simulada por

if(acumulador == 0)
contadorlnstrucao = operando;

Neste ponto você deve implementar seu simulador Simpletron e executar cada um dos programas LMS escritos no Exercício 7.18. Você deve aprimorar a LMS com recursos adicionais e fornecê-los ao seu simulador.

Seu simulador deve verificar vários tipos de erros. Durante a fase de carregamento do programa, por exemplo, cada número que o usuário digitar na memória do Simpletron deve estar no intervalo -9999 a + 9999. Seu simulador deve usar um loop while para examinar se cada número fornecido está nesse intervalo e, se não estiver, continuar a pedir ao usuário que forneça novamente o número até que seja fornecido um número correto.

Durante a fase de execução, seu simulador deve verificar vários erros sérios, como uma tentativa de divisão por zero, tentativa de executar códigos de operações inválidas, overflow do acumulador (i.e., operações aritméticas resultando em valores maiores do que +9999 ou menores do que -9999) e coisas assim. Tais

erros sérios são chamados *erros fatais.* Ao ser detectado um erro fatal, seu simulador deve imprimir uma mensagem de erro como:

*** Tentativa de dividir por zero ***
*** Interrupção anormal da execução do Simpletron ***
e deve um imprimir um despejo completo de computador no formato analisado anteriormente. Isso ajudará o usuário a localizar o erro no programa.

7.20 Modifique o programa de embaralhamento e distribuição de cartas da Fig. 7.24 de modo que as operações embaralhamento e distribuição sejam realizadas pela mesma função (embaralharEDistribuir). A função deve conter uma estrutura de loops aninhados similar à da função embaralhar da Fig. 7.24.

7.21 O que faz o seguinte programa?

```
#include <stdio.h>
void mystery1(char *,  const char *);
main() {
char stringl[80],  string2[80]; printf("Digite duas strings: "); scanf("%s%s",  stringl,  string2);
mystery1(stringl, string2);
printf ("'%s\n", string1); return 0;
}
void mysteryl(char *s1,  const char *s2) {
while  (*s1  != '\0') + + s1;
for  (  ;   *s1 = *s2;  s1++, s2++) ;  /* instrução vazia */
```

7.22 O que faz o seguinte programa?

```
#include <stdio.h>
int mystery2(const char *);
main()
char string[80]; ,
printf("Digite uma string: "); scanf("%s", string); printf ("9&d\n", mystery2 (string) ) ;
return 0;
int mystery2(const char *s)
int x = 0;
for  (  ;  *s  !=  '\0'; s++) + +x;
return x;
```

7.23 Encontre o erro em cada um dos seguintes segmentos de programas. Se o erro puder ser corrigido, explique como.

a) **int *number;**
printf("%d\n", *number);
b) **float *realPtr; long *integerPtr; integerPtr = realPtr;**
c) **int * x, y; x = y;**
d) **char s[] = "isso e um array de caracteres"; int count;**
for (; *s != ' \0'; s + +) printf("%c ", *s);
e) **short *numPtr, result; void *genericPtr = numPtr; result = *genericPtr + 7;**
f) **float x = 19.34; float xPtr = &x; printf (**"5sf**\n", xPtr) ;**
g) **char *s;**
printf("%s\n", s) ;

7.24 *(Classificação Rápida)* Nos exemplos e exercícios do Capítulo 6, analisamos as técnicas de classificação de bolhas, classificação de depósitos e classificação de seleção. Apresentamos agora uma técnica de classificação recursiva chamada *Classificação Rápida (Quicksort)*. O algoritmo básico para um array unidimensional de valores é o seguinte:

1) *Etapa de Partição:* Tome o primeiro elemento do array não-ordenado e determine seu local