UFS - Universidade Federal de Sergipe
CCET -Centro de Ciências Exatas e Tecnologia
DCOMP - Departamento de Computação

Disciplina:	COMP0223 – Arquitetura de Computadores I – T03.
Professor:	Brunelli
Período:	2018.1
Objeto:	Trabalho T02 – **Simpletron**.
Data de Entrega:	05/07/2018 (impresso e via SIGAA).

Escreva um programa em C que implemente o simulador de um computador simples (*Simpletron*).

O programa deve receber um arquivo de entrada (editado via *notepad*) contendo as instruções a serem executadas, produzir um arquivo de saída contendo o *dump* da execução do programa (no formato texto) e não deve interagir com o usuário durante a sua execução (invocado por linha de comando).

Observações:

- Adote o necessário.
- A descrição do *Simpletron* [1] encontra-se nas folhas seguintes (filtre as informações).
- Evite usar orientação a objetos.
- Evite interface gráfica.
- Evite usar Java, C++, C#.
- Não use as soluções da Internet.
- Deve ser entregue: trabalho escrito contendo a listagem (apêndice) e arquivos fonte e executável (submetido via SIGAA).
- Use a norma NBR 14724 (download via UFS/Bicen) para a escrita do trabalho e de maneira simplificada.
- Trabalho em grupo com dois (2) alunos.

oOo

deverá usar os mesmos parâmetros de `printArray`, `minimum` e `maximum`. Armazene os ponteiros para as quatro funções no array `processGrades` e use a escolha feita pelo usuário como o subscrito do array que chama cada função.

7.26 O que o programa a seguir faz?

```
1  /* ex07_26.c */
2  /* O que esse programa faz? */
3  #include <stdio.h>
4
5  int mystery3( const char *s1, const
   char *s2 ); /* protótipo */
6
7  int main( void )
8  {
9     char string1[ 80 ]; /* cria array
      char */
10    char string2[ 80 ]; /* cria array
      char */
11
12    printf( "Digite duas strings: " );
13    scanf( "%s%s", string1 , string2
      );
14    printf( "O resultado é %d\n", mys-
      tery3( string1, string2 ) );
15    return 0; /* indica conclusão bem-
      -sucedida */
16 } /* fim do main */
17
18 int mystery3( const char *s1, const
   char *s2 )
19 {
20    for ( ; *s1 != '\0' && *s2 != '\0';
      s1++, s2++ ) {
21
22       if ( *s1 != *s2 ) {
23          return 0;
24       } /* fim do if */
25    } /* fim do for */
26
27    return 1;
28 } /* fim da função mystery3 */
```

Seção especial de exercícios: criando seu próprio computador

A partir dos próximos problemas, faremos um desvio temporário do mundo da programação em linguagem de alto nível. 'Abriremos a tampa' de um computador e examinaremos sua estrutura interna. Apresentaremos a programação em linguagem de máquina e escreveremos vários programas nessa linguagem. Para tornar essa uma experiência especialmente valiosa, montaremos um computador (por meio da técnica de *simulação* baseada no software) por meio do qual você poderá executar seus programas em linguagem de máquina!

7.27 ***Programação em linguagem de máquina.*** Vamos criar um computador a que chamaremos de Simpletron. Como seu nome sugere, ele é uma máquina simples, porém, como veremos em breve, também é poderosa. O Simpletron roda programas escritos na única linguagem que ele entende diretamente — ou seja, a Simpletron Machine Language, ou SML, para abreviar.

O Simpletron contém um *acumulador* — um 'registrador especial' em que as informações são colocadas antes que o Simpletron as utilize em cálculos ou as examine de diversas maneiras. No Simpletron, toda a informação é tratada em termos de *palavras*. Uma palavra é um número decimal de quatro dígitos com sinal, como `+3364`, `-1293`, `+0007`, `-0001` e assim por diante. O Simpletron é equipado com uma memória de 100 palavras, e essas palavras são referenciadas por seus números de local 00, 01, ..., 99.

Antes de executar um programa SML, temos que *carregar*, ou colocar, o programa na memória. A primeira instrução (ou comando) de cada programa SML é sempre colocada no local 00.

Cada instrução escrita em SML ocupa uma palavra da memória do Simpletron (e, portanto, as instruções são números decimais de quatro dígitos com sinal). Consideramos que o sinal de uma instrução SML é sempre de adição, porém o sinal de uma palavra de dados pode ser tanto de adição quanto de subtração. Cada local na memória do Simpletron pode conter uma instrução, um valor de dados usado por um programa ou uma área da memória não utilizada (e, portanto, indefinida). Os dois primeiros dígitos de cada instrução SML consistem no *código de operação*, que especifica a operação a ser realizada. Os códigos de operação em SML estão resumidos na Figura 7.32.

Código de operação	Significado
Operações de entrada/saída:	
`#define READ 10`	Lê uma palavra do terminal para um local específico na memória.
`#define WRITE 11`	Escreve uma palavra de um local específico na memória para o terminal.

Figura 7.32 ■ Códigos de operação na Simpletron Machine Language (SML). (Parte 1 de 2.)

Código de operação	Significado
Operações de carregamento/armazenamento:	
`#define LOAD 20`	Carrega uma palavra de um local específico na memória para o acumulador.
`#define STORE 21`	Armazena uma palavra do acumulador para um local específico na memória.
Operações aritméticas:	
`#define ADD 30`	Soma uma palavra de um local específico na memória à palavra no acumulador (deixa o resultado no acumulador).
`#define SUBTRACT 31`	Subtrai uma palavra de um local específico na memória da palavra no acumulador (deixa o resultado no acumulador).
`#define DIVIDE 32`	Divide uma palavra de um local específico na memória pela palavra no acumulador (deixa o resultado no acumulador).
`#define MULTIPLY 33`	Multiplica uma palavra de um local específico na memória pela palavra no acumulador (deixa o resultado no acumulador).
Operações de transferência de controle:	
`#define BRANCH 40`	Desvia para um local específico na memória.
`#define BRANCHNEG 41`	Desvia para um local específico na memória se o acumulador for negativo.
`#define BRANCHZERO 42`	Desvia para um local específico na memória se o acumulador for zero.
`#define HALT 43`	Para (*halt*), ou seja, o programa concluiu sua tarefa.

Figura 7.32 ■ Códigos de operação na Simpletron Machine Language (SML). (Parte 2 de 2.)

Os dois últimos dígitos de uma instrução SML consistem no *operando*, que é o endereço do local de memória que contém a palavra à qual a operação se aplica. Agora, consideremos diversos programas SML simples. O programa SML a seguir lê dois números do teclado, calcula e imprime sua soma.

Exemplo 1		
Local	Número	Instrução
00	+1007	*(Lê A)*
01	+1008	*(Lê B)*
02	+2007	*(Carrega A)*
03	+3008	*(Soma B)*
04	+2109	*(Armazena C)*
05	+1109	*(Escreve C)*
06	+4300	*(Halt)*
07	+0000	*(Variável A)*
08	+0000	*(Variável B)*
09	+0000	*(Resultado C)*

A instrução +1007 lê o primeiro número do teclado e o coloca no local 07 (que foi inicializado em zero). Depois, +1008 lê o próximo número para o local 08. A instrução *load*, +2007, coloca o primeiro número no acumulador, e a instrução *add*, +3008, soma o segundo número ao número no acumulador. *Todas as instruções aritméticas da SML deixam seus resultados no acumulador.* A instrução *store*, +2109, coloca o resultado de volta ao local de memória 09, do qual a instrução *write*, +1109, apanha o número e o imprime (como um número decimal de quatro dígitos com sinal). A instrução *halt*, +4300, encerra a execução.

O programa SML a seguir lê dois números do teclado, determina e imprime o valor maior. Observe o uso da instrução +4107 como uma transferência de controle condicional, da mesma forma que a estrutura `if` em C.

Exemplo 2		
Local	Número	Instrução
00	+1009	*(Lê A)*
01	+1010	*(Lê B)*
02	+2009	*(Carrega A)*
03	+3110	*(Subtrai B)*
04	+4107	*(Desvia, se negativo, para 07)*
05	+1109	*(Escreve A)*
06	+4300	*(Halt)*
07	+1110	*(Escreve B)*
08	+4300	*(Halt)*
09	+0000	*(Variável A)*
10	+0000	*(Variável B)*

Agora, escreva programas SML que realizem cada uma das tarefas a seguir:

a) Use um loop controlado por sentinela que leia 10 inteiros positivos, e calcule e imprima sua soma.

b) Use um loop controlado por contador que leia sete números, alguns positivos e alguns 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 deverão ser processados.

7.28 ***Um simulador de computador.*** A princípio, pode parecer incrível, mas ao resolver esse problema você estará construindo seu próprio computador. Não, você não terá

que soldar componentes. Na verdade, você usará a técnica poderosa da *simulação baseada no software* para criar um *modelo de software* do Simpletron. Você não ficará desapontado. Seu simulador do Simpletron transformará o computador que você está usando em um Simpletron, e você realmente poderá executar, testar e depurar os programas SML que foram escritos no Exercício 7.27.

Ao executar seu simulador do Simpletron, ele deverá começar imprimindo:

```
*** Bem vindo ao Simpletron! ***
*** Favor digitar seu programa, uma instrução ***
*** (ou palavra de dados) por vez. Mostrarei  ***
*** o número do local e uma interrogação (?). ***
*** Você, então, deverá digitar a palavra para esse ***
*** local. Digite a sentinela -99999 para     ***
*** encerrar a entrada do seu programa.       ***
```

Sugestão:
Colocar no impessoal. ⇨

Simule a memória do Simpletron com um array de subscrito único, `memory`, que contenha 100 elementos. Agora, suponha que o simulador esteja sendo executado. Examinemos o diálogo enquanto digitamos o programa do Exemplo 2 do Exercício 7.27:

```
00 ? +1009
01 ? +1010
02 ? +2009
03 ? +3110
04 ? +4107
05 ? +1109
06 ? +4300
07 ? +1110
08 ? +4300
09 ? +0000
10 ? +0000
11 ? -99999
*** Carga do programa concluída    ***
*** Iniciando execução do programa ***
```

Agora, o programa SML foi colocado (ou carregado) no array `memory`. O Simpletron executa o programa SML. Ele começa com a instrução no local 00 e continua a sequência, a menos que seja direcionado para alguma outra parte do programa por uma transferência de controle.

Use a variável `accumulator` para representar o registrador do acumulador. Use a variável `instructionCounter` para registrar o local na memória que contém a instrução que está sendo executada. Use a variável `operationCode` para indicar a operação que está sendo executada no momento — ou seja, os dois dígitos da esquerda da palavra de instrução. Use a variável `operand` para indicar o local da memória em que a instrução está operando no momento. Assim, `operand` são os dois dígitos mais à direita da instrução que está sendo executada no momento. Não execute instruções diretamente da memória. Em vez disso, transfira a próxima instrução a ser executada da memória para uma variável chamada `Register`. Depois, 'recolha' os dois dígitos da esquerda e coloque-os na variável `operationCode`, e 'recolha' os dois dígitos da direita e coloque-os em `operand`.

Quando o Simpletron iniciar sua execução, os registradores especiais serão inicializados da seguinte forma:

`accumulator`	+0000
`instructionCounter`	00
`instructionRegister`	+0000
`operationCode`	00
`Operand`	00

Agora, vamos 'caminhar' pela execução da primeira instrução SML, +1009, no local de memória 00. Isso é chamado *ciclo de execução da instrução*.

O `instructionCounter` nos diz o local da próxima instrução a ser executada. *Buscamos* o conteúdo desse local a partir de `memory` usando a instrução C

```
instructionRegister = memory[ instructionCounter ];
```

O código da operação e o operando são extraídos do registrador de instrução por meio das instruções

```
operationCode = instructionRegister / 100;
operand = instructionRegister % 100;
```

Agora, o Simpletron precisa determinar se o código da operação é realmente uma *leitura* (ou uma *escrita*, uma *carga* e assim por diante). Um `switch` diferencia entre as doze operações da SML.

Na estrutura `switch`, o comportamento de diversas instruções SML é simulado da seguinte forma (deixamos as outras para o leitor):

ler: `scanf( "%d", &memory[ operand ] );`
carregar: `accumulator = memory[ operand ];`
somar: `accumulator += memory[ operand ];`
diversas instruções de desvio: Discutiremos em breve.
halt: Essa instrução imprime a mensagem

```
*** Execução do Simpletron encerrada ***
```

e depois imprime o nome e o conteúdo de cada registrador, bem como o conteúdo completo da memória. Essa listagem normalmente é chamada de *dump do computador*. Para ajudá-lo a programar sua função de dump, uma amostra do formato de dump aparece na Figura 7.33. Um dump, após a execução do programa Simpletron, mostraria os valores reais das instruções e os valores dos dados no momento em que a execução foi encerrada.

```
REGISTERS:
accumulator                +0000
instructionCounter            00
instructionRegister        +0000
operationCode                 00
operand                       00

MEMORY:
          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
```

Figura 7.33 ■ Exemplo do formato de dump do Simpletron.

Prosseguiremos com a execução da primeira instrução do nosso programa, a saber, o +1009 no local 00. Conforme indicamos, a estrutura `switch` simula isso ao executar a instrução C

```
scanf( "%d", &memory[ operand ] );
```

O ponto de interrogação (?) deve ser exibido na tela antes que o `scanf` seja executado, como modo de pedir a entrada do usuário. O Simpletron espera que o usuário digite um valor e depois pressione a tecla *Enter*. O valor é, então, lido no local 09.

Nesse ponto, a simulação da primeira instrução foi concluída. Tudo o que resta é preparar o Simpletron para a execução da próxima instrução. Como a instrução recém-executada não foi uma transferência de controle, precisamos simplesmente incrementar o registrador do contador de instrução da seguinte forma:

```
++instructionCounter;
```

Isso conclui a execução simulada da primeira instrução. O processo inteiro (ou seja, o ciclo de execução da instrução) começa novamente com a busca da próxima instrução a ser executada.

Agora, consideremos como as instruções de desvio — as transferências de controle — são simuladas. Tudo o que precisamos é ajustar o valor no contador de instrução de forma adequada. Portanto, a instrução de desvio incondicional (40) é simulada dentro do `switch` como

```
instructionCounter = operand;
```

A instrução condicional de 'desvio se acumulador é zero' é simulada como

```
if ( accumulator == 0 ) {
   instructionCounter = operand;
}
```

Nesse ponto, você deverá implementar seu simulador Simpletron e executar os programas SML que escreveu no Exercício 7.27. Você poderá embelezar a SML com outros recursos e oferecê-los em seu simulador.

Seu simulador deverá verificar os diversos tipos de erros. Durante a fase de carga do programa, por exemplo, cada número que o usuário digita na `memory` do Simpletron deverá estar no intervalo -9999 a +9999. Seu simulador deverá usar um loop `while` para testar se cada número inserido está nessa faixa e, se não estiver, deverá continuar pedindo ao usuário que informe o número novamente até que um número correto seja digitado.

Durante a fase de execução, seu simulador deverá verificar vários erros sérios, como tentativas de divisão por zero, tentativas de execução de códigos de operação invá-

lidos e estouros do acumulador (ou seja, operações aritméticas que resultam em valores maiores que +9999 ou menores que -9999). Esses erros sérios são chamados *erros fatais*. Quando um erro fatal é detectado, seu simulador deverá imprimir uma mensagem de erro como:

```
*** Tentativa de divisão por zero ***
*** Execução do Simpletron encerrada de forma anormal. ***
```

e deverá imprimir um dump completo do computador no formato que apresentamos anteriormente. Isso ajudará o usuário a localizar o erro no programa.

7.29 ***Modificações no simulador do Simpletron.*** No Exercício 7.28, você escreveu uma simulação de software de um computador que executa programas escritos na Simpletron Machine Language (SML). Nesse exercício, propomos várias modificações e melhorias no Simulador do Simpletron. Nos exercícios 12.26 e 12.27, propomos criar um compilador que converte programas escritos em uma linguagem de programação de alto nível (uma variação do BASIC) para a linguagem de máquina do Simpletron. Algumas das modificações e melhorias a seguir podem ser necessárias para executar os programas produzidos pelo compilador.

a) Estenda a memória do Simulador do Simpletron para que ele contenha 1000 locais de memória, permitindo que ele trate de programas maiores.

b) Permita que o simulador realize cálculos de módulo. Isso exige uma instrução adicional na linguagem de máquina do Simpletron.

c) Permita que o simulador realize cálculos de exponenciação. Isso exige uma instrução adicional na linguagem de máquina do Simpletron.

d) Modifique o simulador para que seja possível usar valores hexadecimais em vez de valores inteiros para representar as instruções na linguagem de máquina do Simpletron.

e) Modifique o simulador para permitir a saída de uma nova linha. Isso exige uma instrução adicional na linguagem de máquina do Simpletron.

f) Modifique o simulador para que ele possa processar valores de ponto flutuante, além de valores inteiros.

g) Modifique o simulador fazendo com que seja possível lidar com a input de string. [*Dica:* cada palavra do Simpletron pode ser dividida em dois grupos, cada um mantendo um inteiro de dois dígitos. Cada inteiro de dois dígitos representa o equivalente decimal ASCII de um caractere. Inclua uma instrução em linguagem de máquina que entrará com uma string, e armazenará essa string, a partir de um local específico da memória do Simpletron. A primeira metade da palavra nesse local será um contador do número de caracteres na string (ou seja, o comprimento da string). Cada meia palavra seguinte conterá um caractere ASCII expresso como dois dígitos decimais. A instrução em linguagem de máquina converterá cada caractere em seu equivalente ASCII e o atribuirá a uma meia palavra.]

h) Modifique o simulador para que seja possível lidar com a saída de strings armazenadas no formato da parte (g). [*Dica:* inclua uma instrução em linguagem de máquina que imprima uma string que começa em um local específico da memória do Simpletron. A primeira metade da palavra nesse local é o comprimento da string em caracteres. Cada meia palavra seguinte contém um caractere ASCII expresso como dois dígitos decimais. A instrução em linguagem de máquina verifica o comprimento e imprime a string, traduzindo cada número de dois dígitos em seu caractere equivalente.]

Referencia Bibliográfica

[1] DEITEL, P.; DEITEL, H. *C: Como Programar*. 6ª ed. Brasil: Pearson Education do Brasil, 2011.

oOo