EXERCÍCIO PROGRAMA 1 – SISTEMAS DE PROGRAMAÇÃO

PROFESSOR: JOÃO JOSÉ NETO

NOME: LUCIANA DA COSTA MARQUES

NUSP: 8987826

1) Introdução

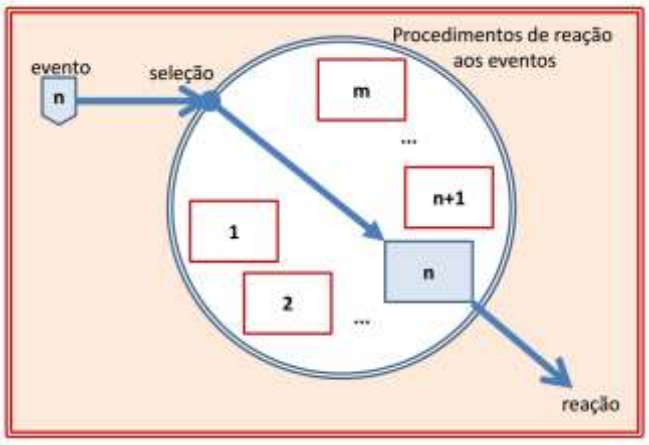
Este exercício programa visa reforçar e consolidar o aprendizado referente a Máquina de Von Neumann e Simulador de Eventos, sendo estes conceitos vistos em aula. Optou-se, na implementação deste exercício, por utilizar-se de uma estrutura de orientação a objetos, por esta ser uma ferramenta prática e de fácil programação e também porque pensou-se que fosse útil usar a categorização por classes. A linguagem utilizada para tal foi c++, devido à familiaridade com a ferramenta.

Esta documentação está dividida em duas partes: a primeira referente à implementação do Motor de Eventos, e a segunda à simulação da MVN usando este motor. É importante observar que não foi implementada uma estrutura de programação defensiva, uma vez que se optou por dar maior concentração ao uso dos conceitos estudados. Ou seja, supõe-se que o usuário siga corretamente os passos desta documentação, como programar eventos no local correto do programa fonte e escrever os arquivos simulados no formato correto.

2) Simulador de Eventos – caracterização

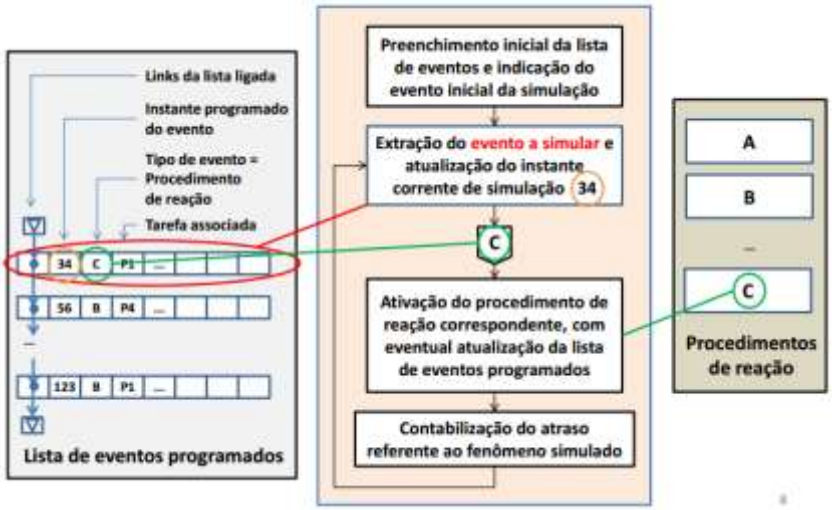
A implementação do Simulador de Eventos seguiu a ideia dada em aula de acordo com as Figuras 1 e 2.

Figura 1: Estrutura do procedimento de reação a um evento



Fonte: Slides de Aula do professor João José Neto

Figura 2: Estrutura geral do Motor de Eventos: Lista Ligada com eventos programados, extração de eventos da lista e procedimentos de reação ao evento extraído.



Fonte: Slides de aula do professor João José Neto

3) Descrição da implementação do Motor de Eventos

Conforme mencionado anteriormente, utilizou-se orientação a objetos para a implementação do Motor. Foi feita uma classe Evento, com os atributos da Figura 3. Este é seu arquivo header, e em seu arquivo source foram implementados o Construtor e Destrutor.

O Construtor da classe Evento exige como parâmetro a reação específica ao evento sendo construído, que no caso é representada por uma string, que será útil quando na rotina de reação aos eventos, descrita mais adiante.

Figura 3: Atributos da classe “Evento”

```
class Evento{
public:
    Evento(string Reacao);
    ~Evento();
    string tipoDeReacao;
    int instanteProgramado;
    bool breakpoint = false;
    Evento* proximo;
};
```

- **<u>Breve descrição da Classe Evento:</u>** ela tem como parâmetros o instante em que ele foi programado para ocorrer, o tipo de Reação que ele gera e se há um breakpoint inserido pelo usuário. Como é uma classe que faz parte de uma lista ligada, inseriu-se também os atributos “próximo”, para detectar qual o próximo evento ligado a ele.

Para fazer a lista ligada de eventos, criou-se uma outra classe chamada ListaDeEventos. Na Figura 4, encontra-se seu arquivo header. Em seu arquivo source, foram implementados o seu Construtor e Destrutor.

Figura 4: Atributos da Classe “ListaDeEventos”

```
class ListaDeEventos{
public:
    ListaDeEventos();
    ~ListaDeEventos();
    Evento* inicio;
    Evento* fim;
};
```

- **Breve descrição da Classe ListaDeEventos:** o seu único parâmetro é o evento inicial, marcado no parâmetro “início”. Em seu construtor, este evento inicial deve ser enviado como parâmetro.

Para o procedimento da reação aos eventos, foram criadas duas funções específicas para cada tipo lista de evento criada, isto é, se é uma lista de eventos genérica ou se é uma lista de eventos relacionada à MVN. Este procedimento de reação é melhor descrito em cada seção específica a esses tipos, em tópicos a seguir.

4) SIMULADOR DE EVENTOS – LISTA DE EVENTOS GENÉRICA

O simulador permite que o usuário escolha entre simular uma lista de eventos nova, ou uma lista de eventos com eventos específicos da MVN.

No caso da lista de eventos genérica, o que o programa faz é carregar dois eventos iniciais na lista: o “INICIAR SIMULADOR” e o “FINALIZAR SIMULADOR”, sendo estes o início e o fim da lista ligada, respectivamente.

Como o funcionamento do simulador de eventos é melhor descrito com os eventos da máquina de von neumann, optou-se por nesta seção do simulador utilizar-se apenas três eventos simples: o de iniciar e finalizar, mencionados anteriormente, e o “OLA”, que simplesmente imprime saudações ao usuário na tela. A simplicidade dos eventos deve-se por ter optado apenas por demonstrar o funcionamento do motor de eventos.

Em termos de código, foram criados os três eventos:

```
//Eventos Simulador De Eventos
Evento* iniciar_simulador = new Evento("INICIAR SIMULADOR");
Evento* finalizar_simulador = new Evento("FINALIZAR SIMULADOR");
Evento* imprimir_ola = new Evento("OLA");
```

Para a reação a cada evento, foi criada uma função específica. No caso, a reação a cada evento é basicamente imprimir na tela uma mensagem específica.

```
void reacaoSimulador (ListaDeEventos* listaSimulador)
{
    Evento* atual = listaSimulador->inicio;
    int sair = 0;
    string codigo;
    while(sair == 0)
    {
        codigo = atual->tipoDeReacao;
        cout << "Evento atual: " << codigo << "\n";
        if (codigo == "INICIAR SIMULADOR")
        {
            cout << "\n\nIniciando Simulador\n\n";
            atual = atual->proximo;
        }
        if (codigo == "OLA")
        {
            cout << "Ola! Bem vindo ao Simulador de Eventos!\n";
            atual = atual->proximo;
        }
        if (codigo == "FINALIZAR SIMULADOR")
        {
            cout << "\n\nFinalizando Simulador\n\n";
            sair = 1;
        }
    }
}
```

Ao final do relatório, há a demonstração do funcionamento desta parte do simulador.

5) SIMULADOR DE EVENTOS – MÁQUINA DE VON NEUMANN

Para a simulação da MVN, optou-se por continuar utilizando a ferramenta de orientação a objetos. No caso, considera-se a MVN como uma classe específica em separado, com quatro registradores: o “program conter”, utilizado para acessar endereços de memória; o registrador de instrução, que armazena as instruções captadas da memória; e acumulador, para uso geral, e um registrador auxiliar para armazenar argumentos.

A memória tem tamanho máximo de FFF (hexadecimal), e cada posição tem um byte (8 bits).

```
class MVN {
public:
    uint8_t memoria[0xFFF];
    uint16_t acc;
    uint16_t arg;
    uint16_t pc = 0;
    uint16_t instrucao;
    MVN();
    ~MVN();
};
```

Para a MVN, foram criados os seguintes eventos:

```
//Eventos MVN
Evento* carregar_programa = new Evento("CARREGAR PROGRAMA");
Evento* simular_programa = new Evento("SIMULAR PROGRAMA");
Evento* capturar_instrucao = new Evento("CAPTURAR INSTRUCAO");
Evento* decodificar = new Evento("DECODIFICAR");
Evento* executar = new Evento("EXECUTAR");
Evento* fim = new Evento("FINAL");
```

- CARREGAR PROGRAMA: evento cuja reação é ler um arquivo texto e carregar na memória da MVN um programa a partir do valor armazenado no registrador PC;
- SIMULAR PROGRAMA: evento para gerar as reações na lista de eventos criada para a simulação de um programa na memória da MVN;
- CAPTURAR INSTRUÇÃO: ação de capturar uma instrução na memória, atualizando o PC e adicionando ao fim da lista o evento DECODIFICAR;
- DECODIFICAR: evento que trata do OPCODE da instrução e determina o tipo de instrução a ser executada;
- EXECUTAR: execução da instrução conforme seu OPCODE, e adiciona ao final da lista um novo evento CAPTURAR INSTRUÇÃO;
- FINAL: evento que marca o fim da simulação do programa da MVN (e por isso o fim da lista ligada de eventos), criado meramente para auxiliar a programação.

Para a reação da lista de eventos da MVN, foi criada também uma função específica, cujo corpo está aqui omitido por julgar-se suficiente descrever que ela trata o evento atual da lista de eventos e realiza a reação específica a cada um deles conforme mencionado anteriormente na caracterização dos eventos.

As instruções da MVN têm suas ações de acordo com o OPCODE, definido pelos primeiros quatro bits. Quando a instrução é decodificada no evento DECODIFICAR, analisa-se esses quatro bits e a instrução recebe um mneumônico:

- (JP) JUMP: 0XXX
  Atualiza o PC conforme o operando em XXX.
- (JZ) JUMP IF ZERO: 1XXX
  Atualiza o PC para o valor do operando em XXX caso o valor armazenado no acumulador seja nulo.
- (JN) JUMP IF NEGATIVE: 2XXX
  Atualiza o valor de PC para o do operando em XXX caso o valor armazenado no acumulador seja negativo (ou seja, seu bit mais significativo seja 1)
- (LV) LOAD VALUE: 3XXX
  Carrega no acumulador o valor do operando XXX e atualiza o PC para PC + 2.
- (+) ADD: 4XXX
  Soma o conteúdo no acumulador com o armazenado em XXX e atualiza PC para PC + 2.
- (-) SUBTRACT: 5XXX

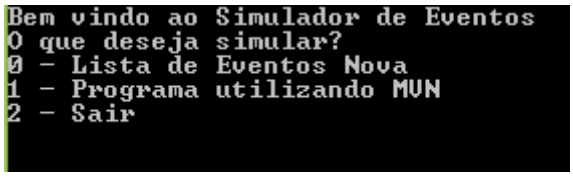
Subtrai o conteúdo no acumulador pelo armazenado em XXX e atualiza PC para PC + 2.

- (*) MULTIPLY: 6XXX
  Multiplica o conteúdo no acumulador pelo armazenado em XXX e atualiza PC para PC + 2.
- (/) DIVIDE: 7XXX
  Divide o valor no acumulador pelo armazenado em XXX e atualiza PC para PC + 2.
- (LD) LOAD: 8XXX
  Coloca no acumulador o valor armazenado no endereço XXX da memória.
- (MM) MEMORY LOAD: 9XXX
  Coloca no endereço XXX da memória o valor do acumulador.
- (SC) CALL SUBROUTINE: AXXX
  Armazena nos endereços XXX e XXX+1 o valor de PC+2. A seguir faz com que PC receba XXX+2.
- (RS) RETURN FROM SUBROUTINE: BXXX
  Faz com que PC receba os valores armazenados em XXX e XXX+1.
- (HM) HALT MACHINE: CXXX
  Existem várias funções para Halt Machine, mas neste caso optou-se por parar a execução das instruções e finalizar a simulação do programa.
- (GD) GET DATA: DXXX
  É feita a leitura de um valor em hexadecimal do teclado, e armazena-o no acumulador. Atualiza PC para PC+2.
- (PD) PUT DATA: EXXX
  O valor lido no acumulador é impresso no terminal/console (utiliza-se um “cout”, equivalente a um “printf” em linguagem c).
- (OS) SYSTEM CALL: FXXX
  Atualiza-se PC para PC+2.

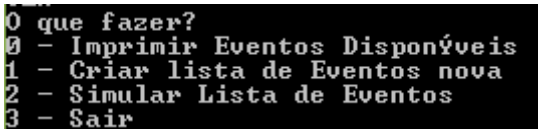
6) FUNCIONAMENTO GERAL DO PROGRAMA

Optou-se aqui por fazer a demonstração do programa apenas da parte da lista de eventos geral, por otimização de tempo. É possível verificar o funcionamento do programa para a máquina de von neumann nos exercícios programa menores (entregues no próximo relatório) e também no projeto do Montador/Ligador.

Ao iniciar o programa, o usuário é direcionado para um menu de início. Deve-se fazer a escolha dentre três opções. Para o caso da lista de Eventos Nova, pressionar “0”. As outras opções são auto-explicativas.

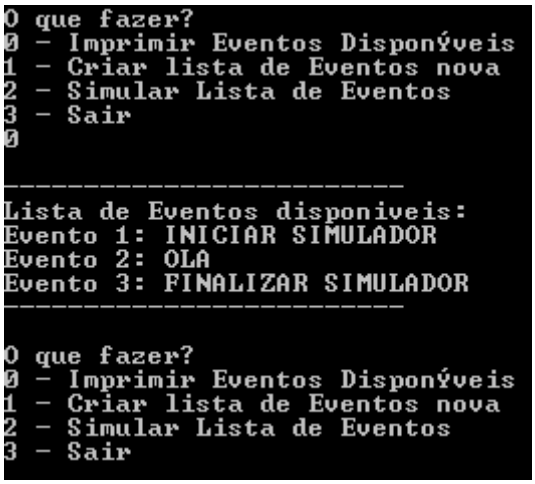


Feita a opção, o programa é redirecionado para o próximo menu referente a uma lista de eventos nova. É possível escolher: 0- consultar os eventos disponíveis; 1-criar uma lista de eventos nova; 2- Simular a lista de eventos; 3- Sair e voltar ao menu principal.



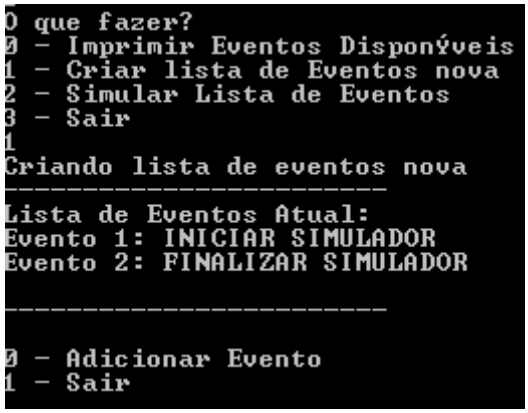
- Imprimir Eventos Disponíveis

  Esta opção permite o usuário verificar os eventos disponíveis. No caso, conforme mencionado anteriormente, são apenas três: iniciar o simulador, imprimir uma mensagem de "olá" e finalizar o simulador. Feita a impressão, o usuário é redirecionado para o menu anterior.

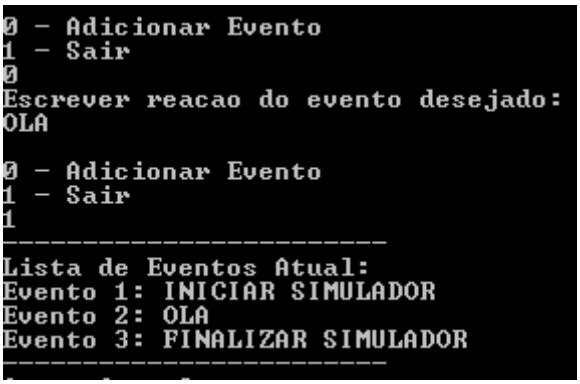


- Criar lista de eventos nova

  Esta opção permite criar-se uma nova lista de eventos. Por definição, a lista inicia com "INICIAR SIMULADOR" e "FINALIZAR SIMULADOR", no começo e no fim da lista respectivamente. O usuário pode ir adicionando novos eventos entre ambos.



  Depois que finalizar a inserção de eventos, a lista criada é impressa e pode ser simulada.



- Simular Lista de eventos

  A simulação da lista de eventos propriamente dita. Para uma lista de eventos que só tenha os eventos de início e fim, por exemplo, ela só imprime que a lista está iniciando e que a lista está finalizando. A cada evento da lista, é impressa também a sua reação, para que o usuário entenda o que ocorre a cada iteração.