PCS 3216 – Sistemas de Programação

Primeira Prova (2021)

Considerações gerais

- Este documento contém o enunciado da primeira prova de PCS3216 – sistemas de programação, de 2021.
- Esta prova é um exercício-programa individual, e tem peso 1. É um projeto de pequeno porte, de implementação de um subconjunto de um sistema de programação.
- Esse trabalho deve ser elaborado, relatado e entregue em um prazo de 10 dias a partir da disponibilização deste enunciado.
- Ao final do prazo de elaboração desse software, um arquivo de dados de entrada será disponibilizado, para ser processado pelo programa a ser desenvolvido.
- As saídas resultantes da execução desse programa sobre esses dados deverão ser coletadas e devidamente analisadas, compondo parte do relatório final do trabalho.

Regras para esta prova

- Esta prova deverá ser desenvolvida **individualmente**.
- Sua meta será a obtenção, em um **prazo de 10 dias**, de um software completo, relatado, em correto funcionamento.
- A **disponibilização** deste enunciado foi feita no dia **13/05, quinta-feira.**
- No dia **22/05** será disponibilizado **até as 12:00h, no Moodle**, a especificação dos **dados de teste** a serem utilizados para a comprovação de funcionamento correto do programa.
- A **entrega** da documentação final desse software, contendo os resultados analisados e comentados dos testes, deverá ser feita **até as 23:59 do domingo, dia 23/05**.
- O material entregue deverá ser apresentado em formato eletrônico, como um **arquivo do tipo pdf**, **sem proteções e não comprimido**.

Como fazer a entrega desta prova

- **Entrega no prazo** estabelecido.
- Software **completo** funcionando **corretamente**.
- **Código** do programa, entradas e saídas dos testes
- **Relatório** técnico detalhado do software desenvolvido

- O relatório técnico bem feito e completo, incluindo:
  - **Descrição conceitual** do software desenvolvido
  - **Modelagem** do programa usando o conceito de simulação guiada por eventos
  - Projeto baseado em um **motor de eventos**.
  - Pseudocódigo/diagrama de blocos da **lógica do software**
  - Tabela de **eventos** e correspondentes **reações**
  - Finalidade dos **testes realizados**, resultados esperados, resultados obtidos, e **análise comentada dos resultados**

Personalização da prova

- Use seu número USP para individualizar as especificações da sua prova:
  - **N** = 1 + (**NUSP** MOD **5**)
  - Resolva esta prova, aplicando-a ao **N**-ésimo problema indicado no slide seguinte.

Problemas (detalhes nos slides seguintes)

1. **Loader+Dumper absolutos**, com entrada/saída em binário, hexadecimal, ASCII e mnemônico
2. **Editor de linhas básico** para arquivos de texto, incluindo operações: numerar, apagar, inserir, fim
3. **Monitor de console**, para operar programas em linguagem binária (endereçar, inserir, listar)
4. **Desmontador absoluto** básico: lista conteúdo da memória em código mnemônico e operando numérico
5. **Decomposição de um texto**, classificando em ordem alfabética e contando o número de ocorrências de cada componente extraído: palavras, números, sinais de pontuação, etc. Os componentes podem aparecer justapostos ou separados por espaço, tabulação, fim de linha

1. Loader + Dumper

Dispõe-se de uma notação numérica para o registro de conteúdos numéricos da memória em arquivos de texto (ASCII):

- ***Na primeira linha, indicar o endereço inicial (3 dígitos hexa)***
- ***Em cada uma das demais linhas, um grupo de 16 números de 2 dígitos hexa cada um, separados por espaço, e encerrado por um fim-de-linha***
- ***A última linha pode conter, se for o caso, um número de dados inferior a 16***
- ***O final do arquivo é marcado por uma linha em branco.***

Exemplo:

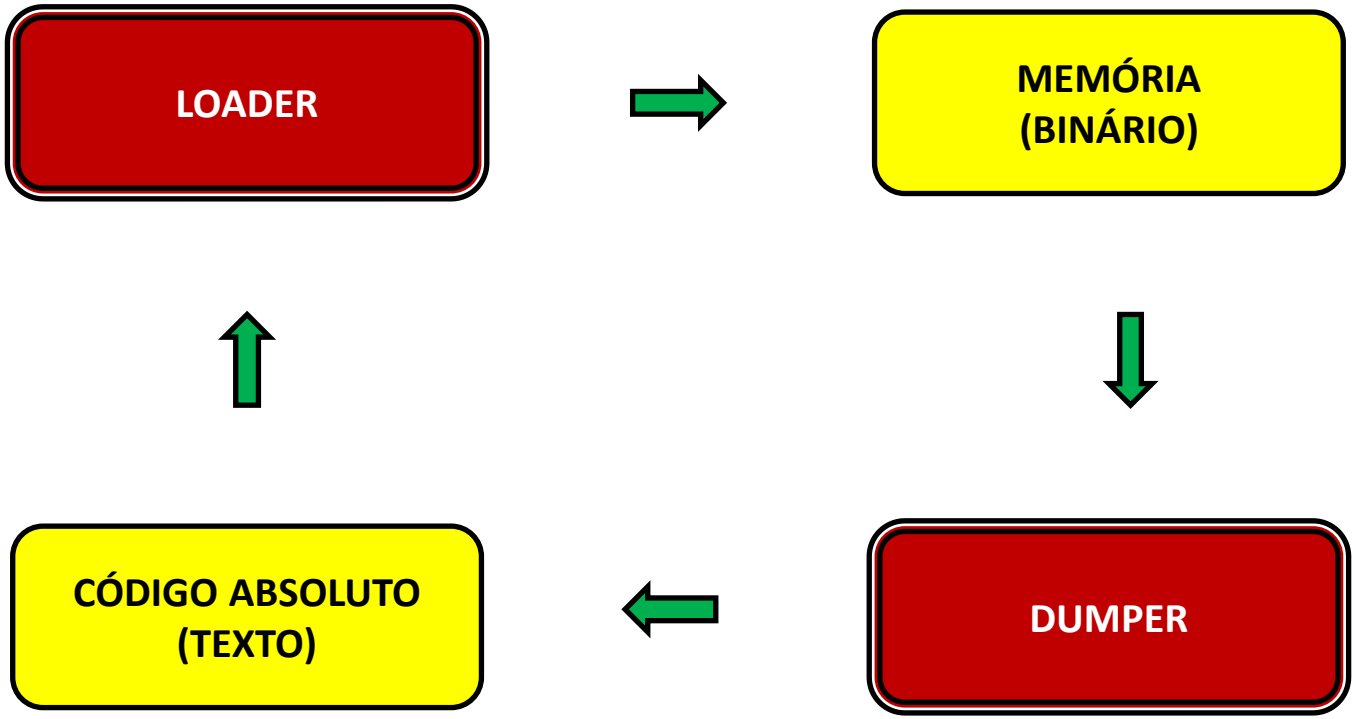
```
E80
DA 1C 9E 00 12 AF 34 9E 3F 7F AA 00 2D 11 FE CD
14 98 12 00 0C 0F F1 15 99 87 E6 5C 4B 23 90 07
12 47 39 48

```

```
(ENDEREÇO INICIAL)
(16 BYTES)
(16 BYTES)
(04 BYTES)
(ÚLTIMA LINHA VAZIA)
```

- O **Loader** deve tratar entradas em arquivo de texto em hexa, no formato acima, converter os dados para binário e armazenar como números binários em um vetor que representa uma memória com 4896 (=4K) endereços (ou seja, 000 a FFF em hexa).
- O **Dumper** deve fazer a operação inversa preparando, para o uso do Loader, um arquivo contendo a imagem da memória, no formato carregável descrito acima.

Complementares e compatíveis



2. Editor de Linhas

- Este programa opera em arquivos de texto. Permite:
  - **Observar conteúdo** (produzir listagem numerada)
  - **Inserir novas linhas** (após linha de número especificado)
  - **Remover uma sequência especificada de linhas**
- Essas ações são especificadas pelo usuário em um outro arquivo de texto, contendo um **script** cujas linhas sempre se iniciam com duas barras (//) seguidas de uma letra:
  //A(arquivo a editar), //L(listar), //I(Inserir), //R(remover), // (Final do script)
- Os comandos do script têm os formatos ilustrados a seguir (os numeros de linha devem estar especificados em ordem crescente dentro do script):

```
//A arquivo_de_entrada.txt
//L
//I número_da_linha depois da qual as linhas abaixo devem ser inseridas
Primeira linha a inserir
...
Última linha a inserir nesta sequência
//R número_da_primeira_linha número_da-última_linha desta sequência
//
```

Exemplo de um script de edição

```
//A modifica01.txt                    escolhe arquivo a editar
//L                                   lista o arquivo, numerado
//I 52                                insere após a linha 52
Primeira linha inserida após 52
Segunda linha inserida após 52
//I 76                                insere após a linha 76
Única linha inserida após 76
//I 94                                insere após a linha 94
Única linha inserida após 94
//R 95 110                            remove as linhas 95 a 110
//I 122                               insere após a linha 122
Primeira linha inserida após 122
Segunda linha inserida após 122
//R 123                               remove a linha 123
//                                    termina a edição
```

3. Monitor de Console

- Este programa é utilizado para operar, em máquinas reais ou virtuais, programas em linguagem binária.
- Os comandos do monitor são fornecidos pela console.
- Os comandos desse script permitem:
  - **Endereçar:** `@` endereço (3 dígitos hexa)
  - **Inserir:** bytes hexa, separados por espaços
  - **Listar:** `L` par de endereços (3 dígitos hexa cada)
  - **Executar:** `X` endereço (3 dígitos hexa)
  - **Finalizar:** `@@`

Uma sessão de uso do monitor

```
@F80

L E00 F80

21 4A FF 30 2E

@FF2

AF DD 2E 14

X 010


@@
```

endereça posição **F80** (hexa)

lista a região da memória no intervalo **E00** a **F80**

insere 5 dados a partir da posição **F80** (hexa)

endereça posição **FF2** (hexa)

insere 4 dados a partir da posição **FF2** (hexa)

executa o programa na memória a partir do endereço **010** (hexa)

finaliza a sessão

4. Desmontador básico

Dispõe-se de uma notação numérica para o usuário registrar conteúdos numéricos da memória:

- ***Na primeira linha, indicar o endereço inicial (3 dígitos hexa)***
- ***Nas demais, 16 números de 2 dígitos hexa cada um, separados por espaço e finalizados por um fim-de-linha***
- ***A última linha pode ter menos números, se for o caso***
- ***O final do arquivo é marcado por uma linha em branco.***

- Este programa desmontador tem por objetivo **converter** o conteúdo de um arquivo de **código absoluto**, no formato acima, ou então o conteúdo de um **intervalo especificado da memória**, em um arquivo de texto cujo conteúdo seja equivalente, porém denotado **em linguagem simbólica**.
- Nessa versão simbólica, os **códigos numéricos das instruções** de máquina devem ser **substituídos pelos mnemônicos** usados na linguagem do montador. Não se exige que os operandos (por exemplo, endereços referenciados) sejam denotados em forma simbólica, podendo ser mantidos na forma numérica (em hexa).

- O desmontador deve funcionar de forma similar à de um dumper quando aplicado ao conteúdo da memória (código binário absoluto), gerando porém linguagem simbólica em lugar de um arquivo numérico em notação hexadecimal.
- Quando aplicado a um arquivo de texto contendo código objeto absoluto em formato hexadecimal, deve operar de forma similar à de um loader, porém gerando instruções simbólicas em um arquivo de saída no lugar de depositar os códigos de máquina na memória do computador.

- Utilize neste projeto a linguagem simbólica do próximo slide para:
  - Determinar os mnemônicos das instruções
  - Usar a notação numérica sugerida
  - Utilizar as pseudo-instruções indicadas

Instruções da linguagem de montagem

```
; MNEMÔNICOS    CÓDIGO    INSTRUÇÃO DA MÁQUINA VIRTUAL
;                  OBS.1: y=número do banco de memória (hexadecimal,4 bits)
;                  OBS.2: x=dígito (hexadecimal, 4 bits)
;   JP           /0xxx     JUMP (UNCONDITIONAL) desvia para endereço /yxxx
;   JZ           /1xxx     JUMP IF ZERO idem, se acumulador contém zero
;   JN           /2xxx     JUMP IF NEGATIVE idem, se acumulador negativo
;   CN           /3x       * instruções de controle (/x = operação desejada)
;   +            /4xxx     ADD soma ac. + conteúdo do endereço /yxxx (8bits)
;   -            /5xxx     SUBTRACT idem, subtrai (operação em 8 bits)
;   *            /6xxx     MULTIPLY idem, multiplica (operação em 8 bits)
;   /            /7xxx     DIVIDE idem, divide (operação em 8 bits)
;   LD           /8xxx     LOAD FROM MEMORY carrega ac. com dado de /yxxx
;   MM           /9xxx     MOVE TO MEMORY move para /yxxx o conteúdo do ac.
;   SC           /Axxx     SUBROUTINE CALL guarda end.de retorno e desvia
;   OS           /Bx       * OPERATING SYSTEM CALL - 16 chamadas do sistema
;   IO           /Cx       * INPUT/OUTPUT – entrada, saída, interrupção
;                /D        * combinação disponível
;                /E        * combinação disponível
;                /F        * combinação disponível
;
; MNEMÔNICOS    OPERANDO  PSEUDO-INSTRUÇÃO DO MONTADOR
;   @     @     /yxxx     ORIGIN define end. inicial do código a seguir
;   #     #     /yxxx     END define final físico do prog. e end. de partida
;   $     $     /xxx      ARRAY define área de trabalho c/tamanho indicado
;   K     K     /xx       CONSTANT preenche byte corrente c/ constante
```

5. Decomposição de texto

- Neste problema, é dado um **arquivo de texto**, contendo material redigido em alguma língua natural (por exemplo, um artigo de jornal), e deseja-se **coletar dele todas as palavras** utilizadas, tabelando-as, ordenando-as alfabeticamente, e memorizando suas ocorrências.
- Usando uma **lista de palavras-chave** à sua escolha, separe em duas classes as palavras encontradas no texto, conforme se trate de uma palavra-chave ou uma palavra comum.

Texto para exercitar o programa

- ***O horizonte de eventos de um buraco negro é como se fosse o topo de uma cachoeira: imagine um peixe nadando num rio em direção ao topo da cachoeira; após certa distância, a água corre tão rapidamente que o peixe não consegue escapar apenas tentando nadar na direção oposta. O peixe, no caso, é o raio de luz se aproximando do buraco negro, sem conseguir escapar e "caindo" ali. Então, ao olhar para um buraco negro, saiba que existe luz e existe matéria ali, mas nós jamais poderemos ver isso, pois a luz não consegue escapar assim que "cai" de uma vez por todas.***

- A tabela de palavras coletadas deve ser uma coleção de linhas em ordem alfabética, tal como exemplificado no seguinte fragmento de tabela:

palavra	número de ocorrências
DE	3
EVENTOS	1
HORIZONTE	1
O	5
...	...