

PCS 3216 – Sistemas de programação

Prova 1



Nome: Jônatas de Souza Nascimento
Número USP: 10772400
Professor: João José

Índice

1.Introdução do Problema.....	2
2.Detalhamento do Programa.....	3
2.1.Descrição do Loader/Dumper.....	3
2.2.Formato do arquivo de texto.....	3
3.Modelagem do software.....	3
3.1.Tabela de eventos.....	3
3.2.Motor de eventos.....	4
3.3.Funções desenvolvidas no código.....	4
3.3.1.Funções auxiliares.....	4
3.3.1.1.Função <i>int_to_char</i>	5
3.3.1.2.Função <i>char_to_int</i>	5
3.3.1.3.Função <i>int_to_byte</i>	5
3.3.1.4.Função <i>byte_to_int</i>	5
3.3.1.5.Função <i>int_to_endereco</i>	5
3.3.1.6.Função <i>endereco_to_int</i>	5
3.3.2.Funções Principais.....	5
3.3.2.1.Função <i>loader</i>	5
3.3.2.2.Função <i>dumper</i>	6
3.3.2.3.Função <i>limpar_memoria</i>	7
3.3.2.4.Função <i>imprimir_memoria</i>	7
3.3.2.5.Função <i>main</i>	8
4.Testes e resultados.....	8
4.1.Teste do Loader.....	8
4.2.Teste do Dumper.....	9
4.3.Teste da Limpeza de Memória.....	10
5.Complemento da Prova.....	10
5.1.Convertir manualmente o código fornecido para o formato adotado na prova.....	10
5.2.Imprimir em hexadecimal o conteúdo inicial da memória.....	12
5.3.Submeter o código gerado no item 1 ao loader que você desenvolveu e imprimir outra vez o conteúdo da memória.....	12
5.4.Acionar o dumper para gerar um código objeto absoluto a partir do conteúdo da memória.....	13
5.5.Limpar a memória preenchendo-a com zeros, e imprimir o conteúdo (zerado) da memória.....	14
5.6.Alimentar o loader com o código gerado no item 4 e imprimir novamente o conteúdo da memória.....	15
5.7.Conclusão.....	15

1.Introdução do Problema

O problema desenvolvido nesse relatório, é a implementação de um Loader e de um Dumper que sejam capazes de tratar entradas de arquivo de texto em hexadecimal e armazenar em uma memória simulada em uma linguagem de programação.

Para a maior proximidade possível de linguagem de máquina, a linguagem escolhida foi C, permitindo assim que uma memória de 4096 endereços fosse simulada, e um Loader e um Dumper fosse programado utilizando-se apenas funções básicas de manipulação de arquivos em C.

2.Detalhamento do Programa

O programa desenvolvido é dividido em duas partes: o Loader e o Dumper, que formam um par em que cada um faz a operação inversa do outro.

2.1.Descrição do Loader/Dumper

O Loader tem a função de ler um arquivo de texto contendo um endereço e um conjunto de N bytes de conteúdo, a ser guardado na memória. Esse programa simula a memória, como um vetor contendo 4096 endereços (de 000 a FFF em hexadecimal), identifica o endereço inicial, lê cada um dos bytes a serem armazenados nessa memória simulada, e guarda cada byte, um a um, a partir do endereço inicial identificado no começo do programa.

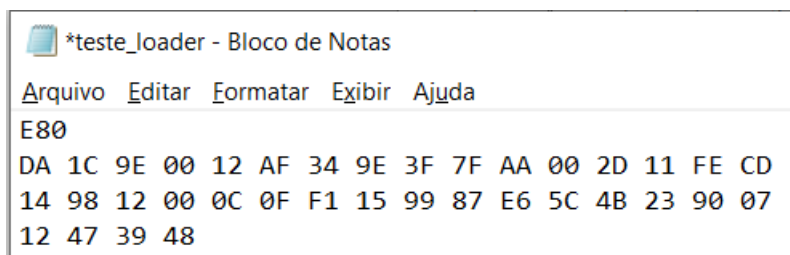
O Dumper, por outro lado, recebe o endereço inicial, o número N de bytes guardados nessa memória e a própria memória, e tem o papel de gerar um arquivo de texto contendo tanto o endereço inicial em ASCII, quanto cada um dos bytes armazenados na memória. Esse arquivo deve ter o mesmo formato do arquivo a ser lido pelo Loader.

2.2.Formato do arquivo de texto

O arquivo de texto a ser lido pelo Loader, e a ser gerado pelo Dumper deve manter um padrão, esse padrão deve seguir as seguintes instruções:

- Na primeira linha deve-se conter o endereço inicial em hexadecimal (3 dígitos).
- Em cada uma das demais linhas, deve-se conter um conjunto de 16 números de 2 dígitos em hexadecimal, separados por um espaço, e encerrados por um fim de linha.
- A última linha pode conter um número inferior a 16.
- O documento deve se encerrar com uma linha vazia.

Um exemplo do formato está a seguir:



```
*teste_loader - Bloco de Notas
Arquivo  Editar  Formatar  Exibir  Ajuda
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
```

3.Modelagem do software

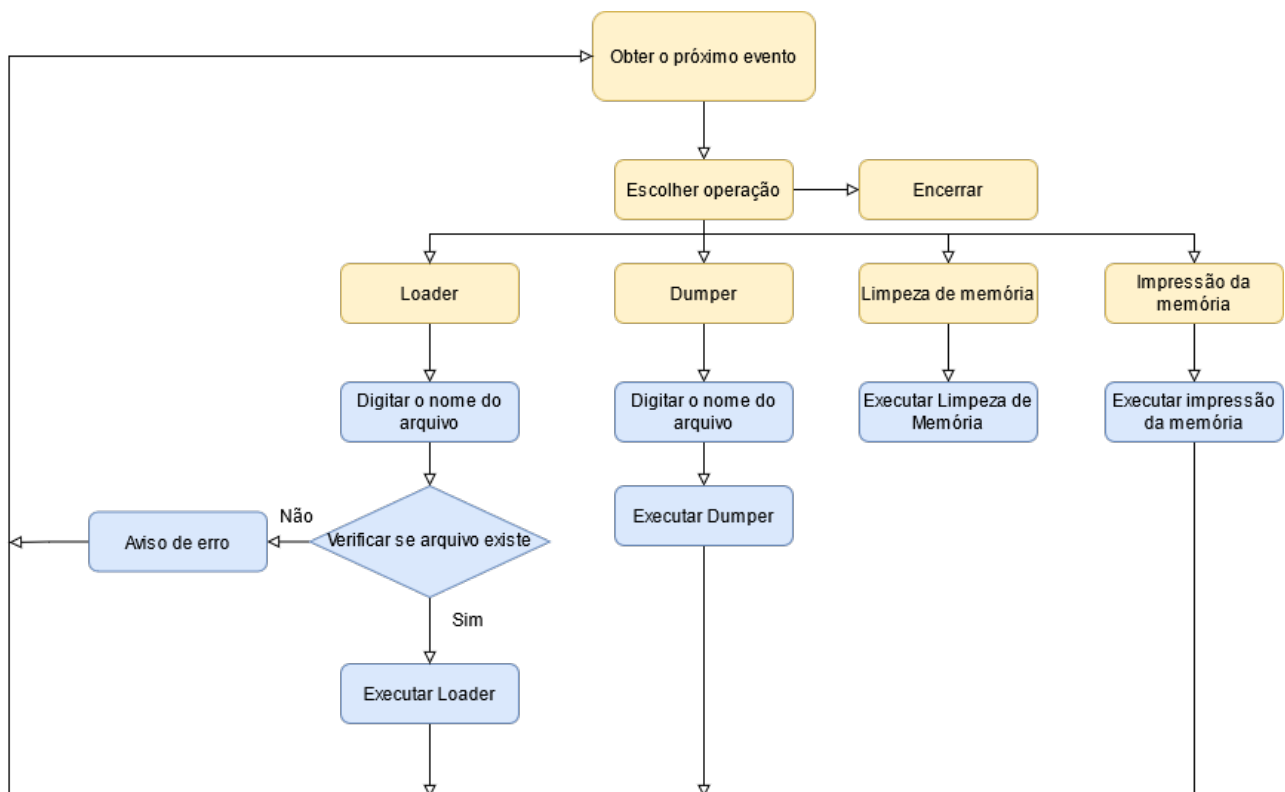
3.1.Tabela de eventos

O programa foi desenvolvido com o método de modelagem baseado em um motor de eventos, para facilitar sua concepção. Portanto o primeiro passo foi desenvolver uma tabela de eventos possíveis e suas reações.

Evento	Reação
Escolha do Loader	Pede o nome do arquivo
Escolha do Dumper	Pede o nome do arquivo
Arquivo Loader existe	Executa Loader
Arquivo Loader não existe	Envia mensagem de erro
Nome do arquivo do Dumper selecionado	Executa Dumper
Escolha da limpeza de memória	A memória é zerada
Escolha do Encerramento do programa	Encerra Programa

3.2.Motor de eventos

De posse da tabela de eventos, foi possível elaborar um motor de eventos, para guiar o desenvolvimento do software.



3.3.Funções desenvolvidas no código

Para o projeto do software, algumas funções simples foram implementadas em C, e todas elas serão detalhadas a seguir, com atenção especial ao foco do problema, o Loader e o Dumper.

3.3.1.Funções auxiliares

3.3.1.1.Função *int_to_char*

Essa função tem como objetivo receber um inteiro e retornar um char correspondente a esse inteiro, para facilitar a transformação de decimal para hexadecimal. Sua entrada é um inteiro em decimal de 0 a 15, e sua saída é o caractere correspondente em hexadecimal, de '0' a 'F'.

3.3.1.2. Função *char_to_int*

Essa função tem como objetivo receber um char correspondente a um número de '0' a 'F' em hexadecimal, e retornar o inteiro correspondente de 0 a 15. É a função inversa a *int_to_char*.

3.3.1.3. Função *int_to_byte*

A função *int_to_byte* recebe um número inteiro correspondente a um número hexadecimal de "00" a "FF" e retorna um vetor de char de tamanho 2 contendo o número hexadecimal correspondente.

3.3.1.4. Função *byte_to_int*

A função *byte_to_int* recebe um vetor de char de tamanho 2 contendo um número hexadecimal de "00" a "FF" e retorna o número inteiro na base decimal correspondente. É a função inversa a *int_to_byte*.

3.3.1.5. Função *int_to_endereco*

A função *int_to_endereco* recebe um número inteiro correspondente a um número hexadecimal de "000" a "FFF" e retorna um vetor de char de tamanho 3 contendo o número hexadecimal correspondente.

3.3.1.6. Função *endereco_to_int*

A função *endereco_to_int* recebe um vetor de char de tamanho 3 contendo um número hexadecimal de "000" a "FFF" e retorna o número inteiro na base decimal correspondente. É a função inversa a *int_to_endereco*.

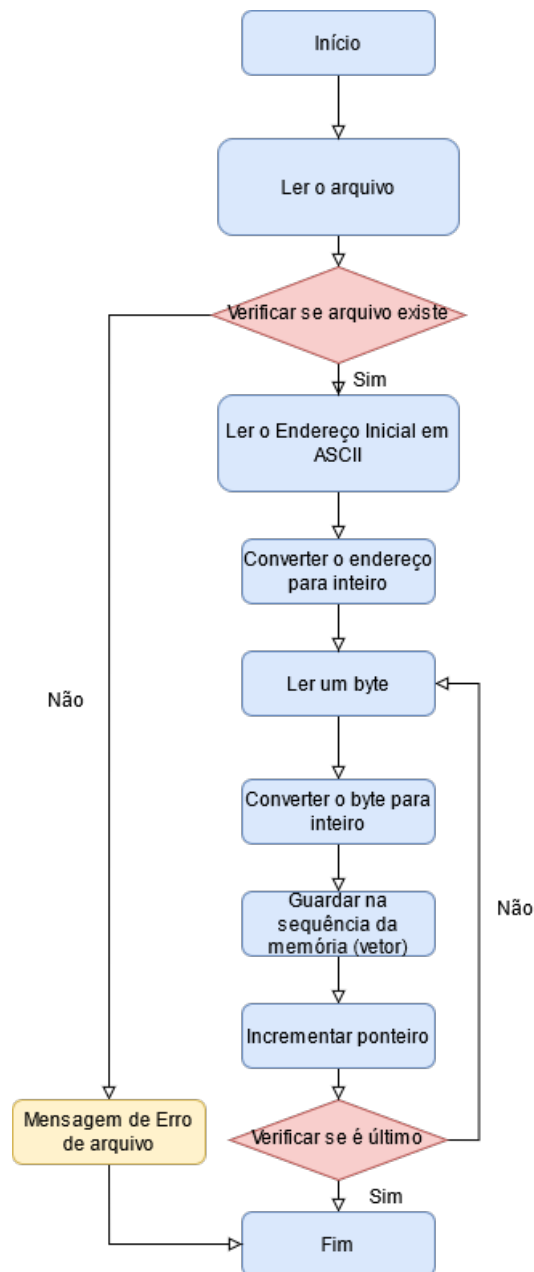
3.3.2. Funções Principais

As funções principais do software são as que vão operar no problema propriamente dito, ou seja, o Loader, o Dumper, as funções que imprimem e limpam a memória, e a função *main*.

3.3.2.1. Função *loader*

O objetivo do Loader é ler um arquivo no formato descrito na seção 2.2, identificar o endereço descrito e os valores a serem guardados, e armazená-los na memória simulada.

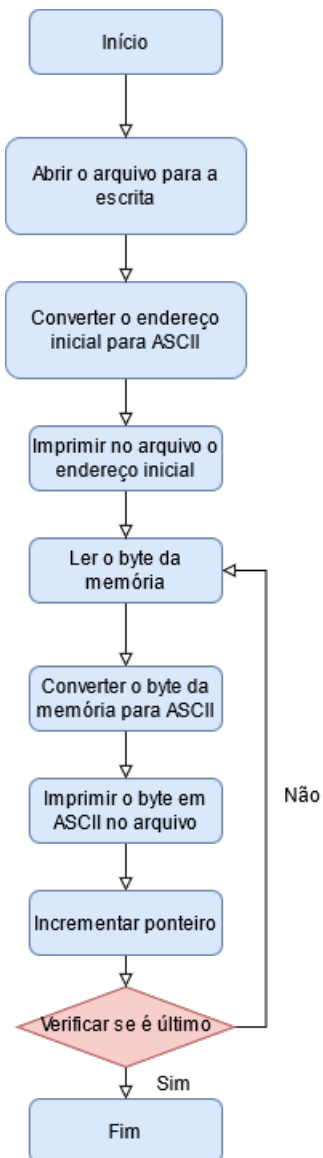
Podemos descrever o Loader como diagrama de blocos da seguinte maneira:



3.3.2.2. Função *dumper*

A função *dumper* recebe o endereço inicial da memória onde os valores estão guardados, e o número de elementos armazenados, e ela deve gerar um arquivo no mesmo formato da seção 2.2.

Podemos descrever a função *dumper* como diagrama de blocos da seguinte maneira:



3.3.2.3.Função *limpar_memoria*

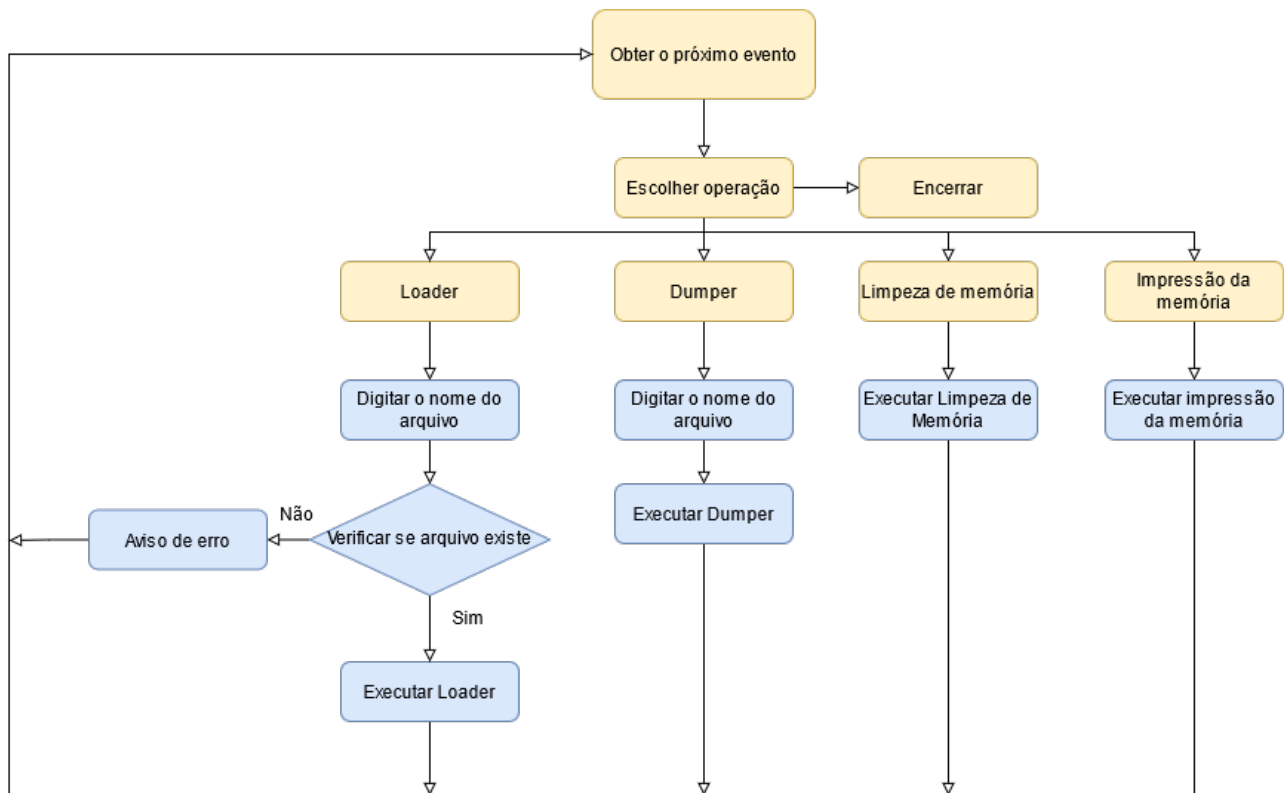
A função `limpar_memoria` recebe também o endereço inicial e o número N de valores armazenados na memória, e ao ser executada tem a função de levar todos esses valores da memória a zero.

3.3.2.4.Função `imprimir_memoria`

A função `imprimir_memoria` imprime no próprio programa a memória completa, no mesmo formato do arquivo de texto, porém iniciando do endereço "000".

3.3.2.5.Função `main`

A função *main* é o corpo de execução do programa completo, onde serão selecionadas as opções de loader/dumper ou de limpeza e exclusão da memória. A função *main* foi elaborada baseando-se no motor de eventos modelado inicialmente, e um diagrama de blocos para ela é descrito a seguir:



4. Testes e resultados

Com o programa implementado, os testes são feitos de modo a exaurir as possibilidades do motor de eventos.

4.1. Teste do Loader

Como primeiro passo de teste, um arquivo *teste_loader.txt* foi gerado com a seguinte entrada:

```

*teste_loader - Bloco de Notas
Arquivo  Editar  Formatar  Exibir  Ajuda
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
  
```

Então se executa o programa, se seleciona a opção do loader e se digita o nome de arquivo:


```
E:\Jônatas\Documents\USP\7 semestre\Sistemas de Programação\P1\loader_du
Bem vindo, NAO utilize o dumper antes do loader

-----

Insira 1 caso queira utilizar o loader
Insira 2 caso queira utilizar o dumper
Insira 3 caso queira limpar a memoria
Insira 4 caso queira imprimir a memoria
Insira 5 caso queira sair
1
Digite o nome do arquivo do loader (inclusive a extensao):
teste_loader.txt
0 Loader funcionou com sucesso!
```

4.2. Teste do Dumper

Para a verificar se o loader funcionou corretamente, vamos selecionar a opção do dumper, e escolher um nome do arquivo para o dumper:

```
teste_loader.txt
0 Loader funcionou com sucesso!

-----

Insira 1 caso queira utilizar o loader
Insira 2 caso queira utilizar o dumper
Insira 3 caso queira limpar a memoria
Insira 4 caso queira imprimir a memoria
Insira 5 caso queira sair
2
Digite o nome do arquivo do dumper (inclusive a extensao):
teste_dumper.txt
0 Dumper funcionou com sucesso!
```

Agora para verificar se tudo ocorreu sem problemas, devemos abrir o arquivo gerado pelo programa:

```
teste_dumper - Bloco de Notas
Arquivo Editar Formatar Exibir Ajuda
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
```

Tudo ocorreu como o esperado, e o arquivo está com o mesmo conteúdo carregado inicialmente, e no exato mesmo formato.

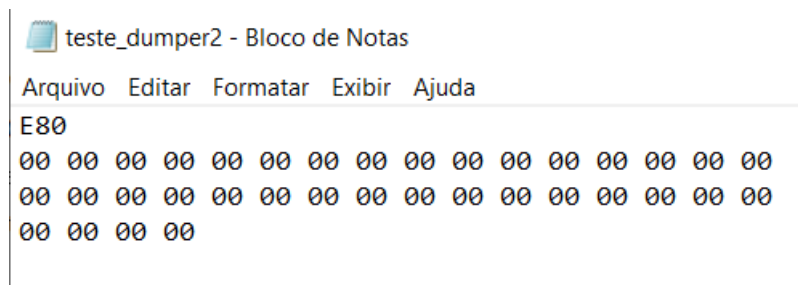
4.3. Teste da Limpeza de Memória

Para verificar agora se a opção de limpeza de memória funciona corretamente, vamos seguir executando o programa e selecionar essa opção:

```
Insira 1 caso queira utilizar o loader
Insira 2 caso queira utilizar o dumper
Insira 3 caso queira limpar a memoria
Insira 4 caso queira imprimir a memoria
Insira 5 caso queira sair
3
Memoria limpa com sucesso
```

Agora vamos utilizar o dumper novamente, com um novo arquivo *teste_dumper2.txt*, e verificar a saída e o arquivo de texto:

```
Insira 1 caso queira utilizar o loader
Insira 2 caso queira utilizar o dumper
Insira 3 caso queira limpar a memoria
Insira 4 caso queira imprimir a memoria
Insira 5 caso queira sair
2
Digite o nome do arquivo do dumper (inclusive a extensao):
teste_dumper2.txt
0 Dumper funcionou com sucesso!
```



Como esperado, o dumper gera o arquivo com a memória zerada, indicando que tudo ocorreu corretamente.

5. Complemento da Prova

5.1. Converter manualmente o código fornecido para o formato adotado na prova

Foi fornecido um código objeto de teste em outro formato, que foi transformado para o formato padrão da seção 2.1.

Código objeto:

05 00 2E 00 18 B5	05 00 1C 10 30 9F	06 00 30 C0 30 01 D9
05 00 10 80 32 39	05 00 20 40 32 69	04 00 33 02 C7
05 00 26 40 33 CD	04 00 34 00 C8	05 00 14 90 34 23
05 00 24 80 34 23	05 00 28 90 34 0F	04 00 35 04 C3
05 00 1A 50 35 5C	04 00 36 00 C6	05 00 16 90 36 1F
07 00 2A 40 36 90 36 93	04 00 37 00 C5	05 00 12 90 37 22
05 00 18 80 37 2C	05 00 1E 80 37 26	05 00 22 90 37 12

Nesse formato, o terceiro byte de cada bloco se refere ao endereço, e os próximos bytes tirando o último se referem aos valores que devem ser incluídos na memória.

Fazendo uma análise um a um, podemos identificar quais valores devem ser guardados em qual posição da memória

10: 80 32 | 12: 90 37 | 14: 90 34 | 16: 90 36 | 18: 80 37 | 1A: 50 35 | 1C: 10 30 | 1E: 80 37 | 20: 40 32
 22: 90 37 | 24: 80 34 | 26: 40 33 | 28: 90 34 | 2A: 40 36 90 36 | 2E: 00 18 | 30: C0 30 01 | 33: 02
 34: 00 | 35: 04 | 36: 00 | 37: 00

Com base nisso o formato de texto já explicado anteriormente pôde ser montado:

```

codigo_teste - Bloco de Notas
Arquivo  Editar  Formatar  Exibir  Ajuda
010
80 32 90 37 90 34 90 36 80 37 50 35 10 30 80 37
40 32 90 37 80 34 40 33 90 34 40 36 90 36 00 18
C0 30 01 02 00 04 00 00

```

Agora já está no formato padrão do dumper/loader.

E:\Jônatas\Documents\USP\7 semestre\Sistemas de Programação\P1\load

```
-----
Insira 1 caso queira utilizar o loader
Insira 2 caso queira utilizar o dumper
Insira 3 caso queira limpar a memoria
Insira 4 caso queira imprimir a memoria
Insira 5 caso queira sair
1
Digite o nome do arquivo do loader (inclusive a extensao):
codigo_teste.txt
0 Loader funcionou com sucesso!
```

E:\Jônatas\Documents\USP\7 semestre\Sistemas de Programaç

```
-----
Insira 1 caso queira utilizar o loader
Insira 2 caso queira utilizar o dumper
Insira 3 caso queira limpar a memoria
Insira 4 caso queira imprimir a memoria
Insira 5 caso queira sair
4
000
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
80 32 90 37 90 34 90 36 80 37 50 35 10 30 80 37
40 32 90 37 80 34 40 33 90 34 40 36 90 36 00 18
C0 30 01 02 00 04 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Podemos ver por conta da impressão da memória, que valores que anteriormente estavam zerados, agora estão com os exatos valores que foram incluídos no *codigo_teste.txt*.

5.4.Acionar o dumper para gerar um código objeto absoluto a partir do conteúdo da memória

Agora podemos acionar o dumper para gerar um arquivo *teste_dumper3.txt* que deve ter a mesma forma que o código objeto gerado inicialmente.

```
-----
Insira 1 caso queira utilizar o loader
Insira 2 caso queira utilizar o dumper
Insira 3 caso queira limpar a memoria
Insira 4 caso queira imprimir a memoria
Insira 5 caso queira sair
2
Digite o nome do arquivo do dumper (inclusive a extensao):
teste_dumper3.txt
0 Dumper funcionou com sucesso!
```

teste_dumper3 - Bloco de Notas

Arquivo Editar Formatar Exibir Ajuda

010

```
80 32 90 37 90 34 90 36 80 37 50 35 10 30 80 37
40 32 90 37 80 34 40 33 90 34 40 36 90 36 00 18
C0 30 01 02 00 04 00 00
```

Podemos confirmar que o arquivo foi gerado com êxito.

5.5.Limpar a memória preenchendo-a com zeros, e imprimir o conteúdo (zerado) da memória

Podemos utilizar a função limpar memória para colocar todos os valores em zero, e depois imprimir a memória como fizemos nos itens anteriores.

E:\Jônatas\Documents\USP\7 semestre\Sistemas de Programa

```
Insira 1 caso queira utilizar o loader
Insira 2 caso queira utilizar o dumper
Insira 3 caso queira limpar a memoria
Insira 4 caso queira imprimir a memoria
Insira 5 caso queira sair
3
Memoria limpa com sucesso

-----

Insira 1 caso queira utilizar o loader
Insira 2 caso queira utilizar o dumper
Insira 3 caso queira limpar a memoria
Insira 4 caso queira imprimir a memoria
Insira 5 caso queira sair
4
000
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Novamente como o esperado, toda a memória foi levada a zero.

5.6. Alimentar o loader com o código gerado no item 4 e imprimir novamente o conteúdo da memória.

Seguindo na mesma execução do programa, vamos utilizar o loader com o arquivo *teste_dumper3.txt* e imprimir a memória novamente:

```
E:\Jônatas\Documents\USP\7 semestre\Sistemas de Programação

Insira 1 caso queira utilizar o loader
Insira 2 caso queira utilizar o dumper
Insira 3 caso queira limpar a memoria
Insira 4 caso queira imprimir a memoria
Insira 5 caso queira sair
1
Digite o nome do arquivo do loader (inclusive a ext
teste_dumper3.txt
0 loader funcionou com sucesso!

-----

Insira 1 caso queira utilizar o loader
Insira 2 caso queira utilizar o dumper
Insira 3 caso queira limpar a memoria
Insira 4 caso queira imprimir a memoria
Insira 5 caso queira sair
4
000
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
80 32 90 37 90 34 90 36 80 37 50 35 10 30 80 37
40 32 90 37 80 34 40 33 90 34 40 36 90 36 00 18
C0 30 01 02 00 04 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Novamente o resultado foi como o esperado, e a partir da posição “10” (em hexadecimal), a memória está com os valores indicados no arquivo fornecido.

5.7. Conclusão

Dados os resultados ótimos obtidos nos mais diversos testes documentados, é possível dizer que o projeto inicial da implementação de um loader e um dumper em linguagem C foi concluído com sucesso. Foram utilizadas as metodologias aprendidas ao longo da disciplina de PCS-3216 para o desenvolvimento de um software que cumpre todos os requisitos propostos, e está em pleno funcionamento.