Resolver: - os exercícios 1, 2, 5, 6, 7 da lista ED1\_Un2\_Lista Ponteiros em anexo; - os exercícios 1, 4, 6, 9, 11 da lista ED1\_Un3\_Lista\_Aloc em anexo. Submeter as soluções em um arquivo PDF, conforme instruções dadas no início das listas.

**LISTA DE EXERCÍCIOS**

**Ponteiros**

*OBS: Para os exercícios que pedem rastreio, execute-os no* ***Code Blocks,*** *utilize o Debugger com breakpoints e acompanhe os valores das variáveis na janela Watches. Faça impressão de tela da janela Watches e do console e inclua na resposta junto com o código fonte.*

1)

1 /\*

2 1. Considere o trecho de programa a seguir:

3

4 int i = 3, j = 5;

5 int \*p, \*q;

6 p = &i;

7 q = &j;

8

9 Determine o valor de cada expressÃ£o a seguir:

10 a) \*p;

11 b) \*q \* i;

12 c) p == &i;

13 d) q < &j;

14 e) (\*p > i) && !(j < \*q);

15 f) !(\*p == i) || (j < \*q);

16 g) \*p - \*q;

17 h) i / \*q;

18 i) \*p \* \*q;

19 j) (float)\*q / \*p;

20 k) 5.0 \* \*q / \*p - 2;

21 l) \*q / \*p \* 5.0 - 2;

22 m) \*q / (\*p â€“ 1.0);

23 n) 3 \* - \*p / \*q + 7;

24 o) ++\*q / \*p;

25 p) \*p / (float)++\*q;

26 q) \*\*&p + \*\*&q;

27 r) ++\*q / \*p \* ++j;

28 s) \*q / \*p \* (float)++i;

29 t) 2 \* \*q / --\*p \* ++i;

30 \*/

31

32 #include <stdio.h>

33 #include <stdlib.h>

34

35 **int** main()

36 {

37 **int** i = 3, j = 5;

38 **int** \*p, \*q;

39 p = &i;

40 q = &j;

41

42 printf("\n Saidas: \n");

43 //1 para TRUE

44 //o para FALSE

45

46 //letra a

47 printf("\n a: %d \n", \*p);

48

49 //letra b

50 printf("\n b: %d \n", \*q\*i);

51

52 //letra c

53 printf("\n c1: %d \n", (p == &i));

54

55 //letra d

56 printf("\n d1: %d \n", q < &j );

57

58 //letra e

59 printf("\n e1: %d\n", (\*p > i) && !(j < \*q) );

60

61 //letra f

62 printf("\n f1: %d \n", !(\*p == i) || (j < \*q));

63

64 //letra g

65 printf("\n g0: %d \n", \*p - \*q);

66

67 //letra h

68 printf("\n h1: %d\n", i / \*q);

69

70 //letra i

71 printf("\n i: %d\n", \*p \* \*q);

72

73 //letra j

74 printf("\n j: %d\n", (**float**)\*q / \*p);

75

76 //letra k

77 printf("\n k: %d\n", 5.0 \* \*q / \*p - 2);

78

79 //letra l

80 printf("\n l: %.2f \n", \*q / \*p \* 5.0 - 2);

81

82 //letra m

83 //printf("\n m: %d\n", \*q / (\*p â€“ 1.0));

84 printf("\n m: invalido \n");

85

86 //letra n

87 printf("\n n1: %d\n", 3 \* - \*p / \*q + 7);

88

89 //letra o

90 printf("\n o2: %d\n", ++\*q / \*p);

91

92 //letra p

93 printf("\n p1: %d\n", \*p / (**float**)++\*q);

94

95 //letra q

96 printf("\n q: %d \n", \*\*&p + \*\*&q);

97

98 //letra r

99 printf("\n r: %d \n", ++\*q / \*p \* ++j);

100

101 //letra s

102 printf("\n s: %d \n", \*q / \*p \* (**float**)++i);

103

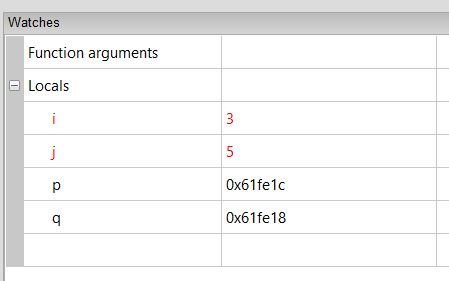
104 //letra t

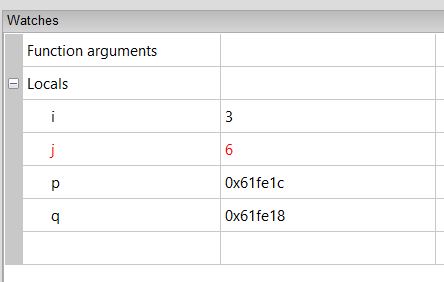
105 printf("\n t: %d \n", 2 \* \*q / --\*p \* ++i);

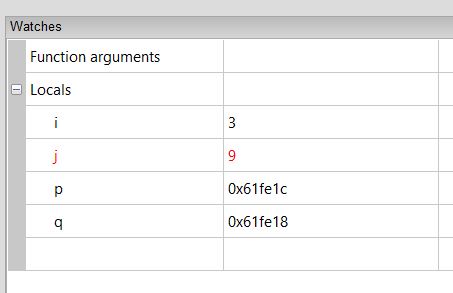
106

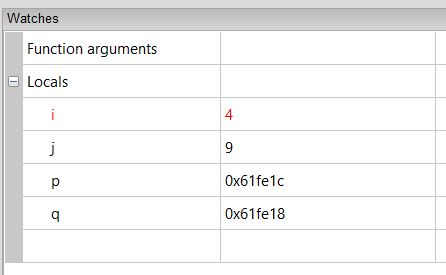
107 }

Watches:

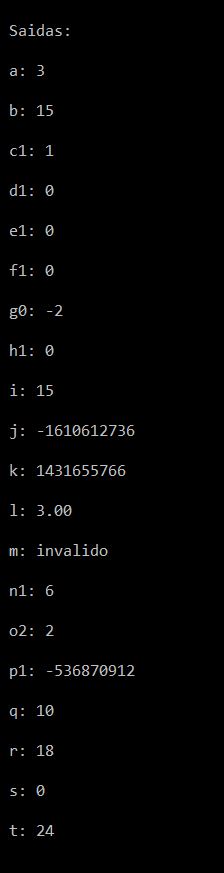








Console:



2)

1 /\*

2 2. Considere as declaraÃ§Ãµes/atribuiÃ§Ãµes a seguir:

3 int x = 10, \*px = &x, \*\*ppx = &px, \*ptemp, temp;

4 float y = 3.2, \*py = &y, \*\*ppy = &py, \*pftemp, ftemp;

5 Considere ainda que as variÃ¡veis ocupam os seguintes endereÃ§os de memÃ³ria:

6

7 x - 28FF1C

8 y - 28FF10

9 px - 28FF18

10 py - 28FF0C

11 ppx - 22FF14

12 ppy - 28FF08

13

14 Determine o valor de cada item a seguir:

15 a) x;

16 b) y;

17 c) px;

18 d) &y;

19 e) &px;

20 f) \*py;

21 g) \*ppx;

22 h) py;

23 i) &x;

24 j) pftemp = py++;

25 k) \*px--;

26 l) \*\*ppy;

27 m) &ppy;

28 n) \*&px;

29 o) temp = \*\*ppx++;

30 p) ++ppx;

31 q) &ppx;

32 r) --py;

33 s) --\*\*ppx;

34 t) ++\*py;

35 \*/

36

37 #include <stdio.h>

38 #include <stdlib.h>

39

40 **int** main()

41 {

42

43 //a terminar

44 //acertar os enderecos de memoria com os da questao

45 **int** x = 10, \*px = &x, \*\*ppx = &px, \*ptemp, temp;

46 **float** y = 3.2, \*py = &y, \*\*ppy = &py, \*pftemp, ftemp;

47

48 //a

49 printf("\n letra a: %d \n", x);

50

51 //b

52 printf("\n letra b: %.2f \n", y);

53

54 //c

55 /\*

56 De acordo com o enunciado:

57 \*/

58 **char** px2[20] = "28FF1C";

59 printf("\n letra c: %s \n", px2);

60

61 //d

62 /\*

63 De acordo com o enunciado:

64 \*/

65 **char** aux1[20] = "28FF10";

66 printf("\n letra d: %s \n", aux1);

67

68 //e

69 /\*

70 De acordo com o enunciado:

71 \*/

72 **char** aux2[20] = "28FF18";

73 printf("\n letra e: %s \n", aux2);

74

75 //f

76 printf("\n letra f: %.2f \n", \*py);

77

78 //g

79 printf("\n letra g: %d \n", \*\*ppx);

80

81

82 //h

83 printf("\n letra h: %d \n", py);

84

85 //i

86 /\*

87 De acordo com o enunciado:

88 \*/

89 **char** aux3[20] = "28FF1C";

90 printf("\n letra i: %d \n", aux3);

91

92 //j

93 pftemp = py++;

94 printf("\n letra j: %d \n", pftemp);

95

96 //k

97 printf("\n letra k: %d \n", \*px--);

98

99 //l

100 printf("\n letra l: %d \n", \*\*ppy);

101

102 //m

103 printf("\n letra m: %d \n", &ppy);

104

105 //n

106 printf("\n letra n: %d \n", \*&px);

107

108 //o

109 temp = \*\*ppx++;

110 printf("\n letra o: %d \n", temp);

111

112 //p

113 printf("\n letra p: %d \n", ++ppx);

114

115 //q

116 printf("\n letra q: %d \n", &ppx);

117

118 //r

119 printf("\n letra r: %d \n", --py);

120

121 //s

122 printf("\n letra s: %d \n", --\*\*ppx);

123

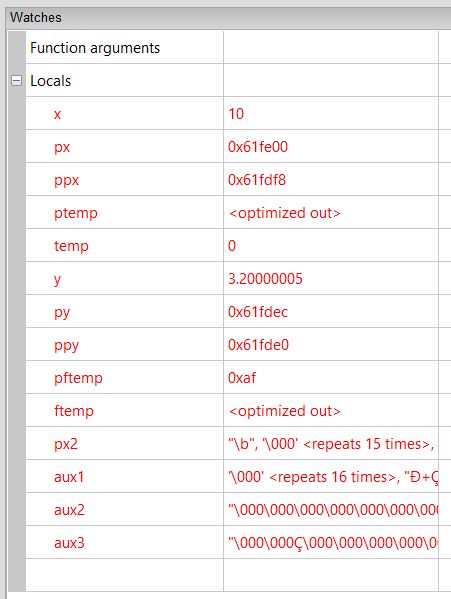
124 //t

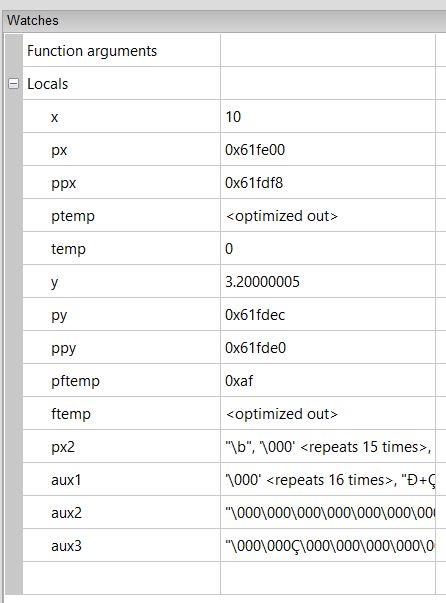
125 printf("\n letra t: %d \n", ++\*py);

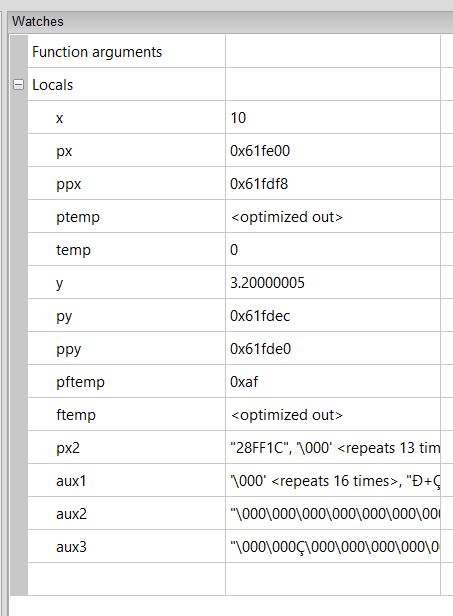
126

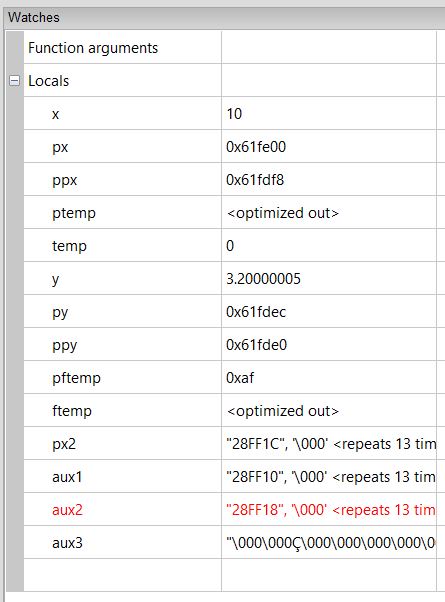
127 }

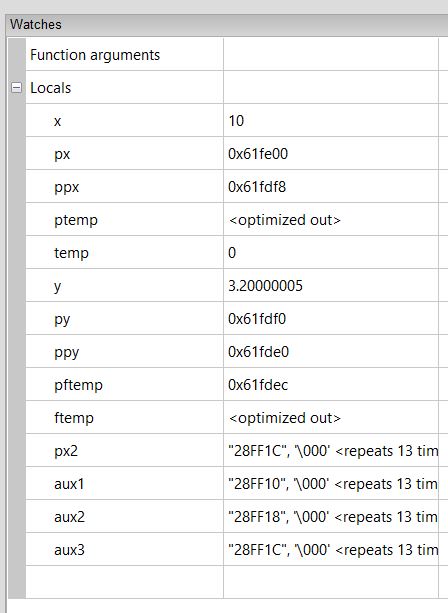
Watches:

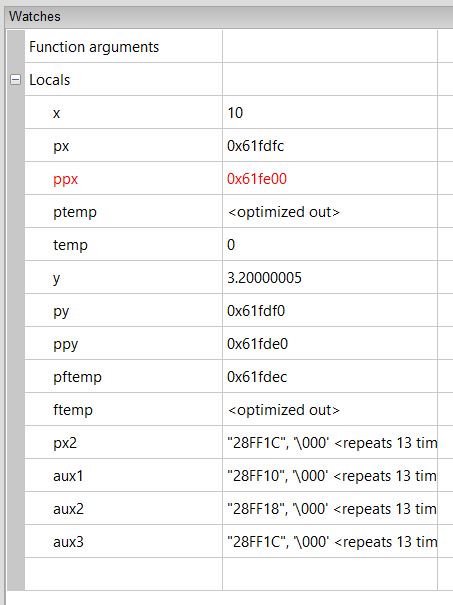


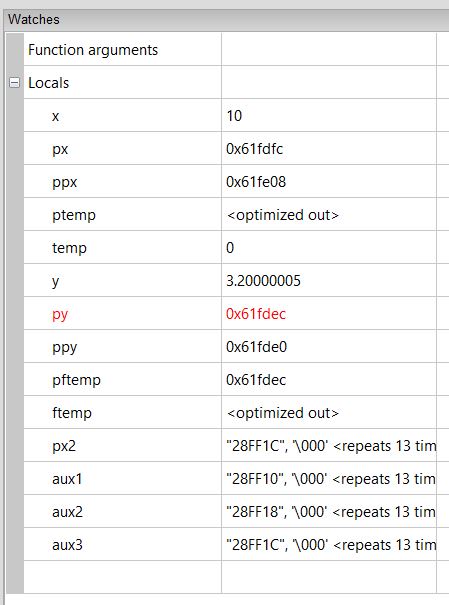




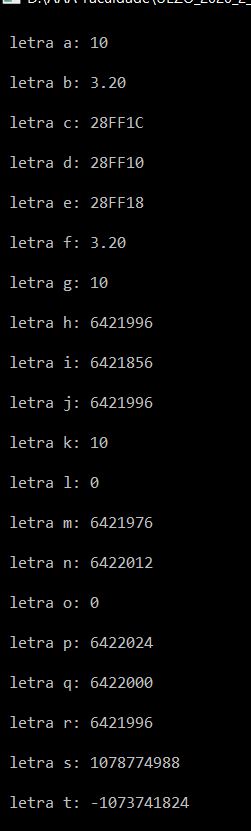








Console:



5)

1 /\*

2 5. Execute o programa abaixo no Code Blocks, faÃ§a o rastreio

3 (utilizando debugger e breakpoints) e identifique a saÃ­da.

4 Apresente impressÃµes de tela do rastreio e da saÃ­da.

5 \*/

6

7 #include <stdio.h>

8 #include <stdlib.h>

9

10 **int** main(**int** argc, **char** \*argv[])

11 {

12 **int** a,b,\*p1, \*p2;

13 a = 4;

14 b = 3;

15 p1 = &a;

16 p2 = p1;

17 \*p2 = \*p1 + 3;

18 b = b \* (\*p1);

19 (\*p2)++;

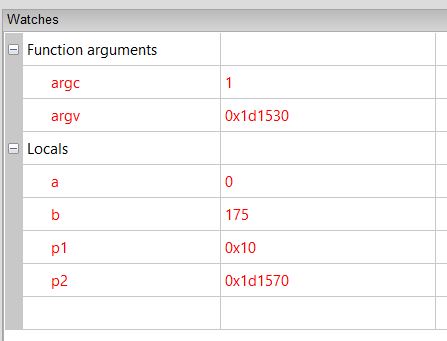
20 p1 = &b;

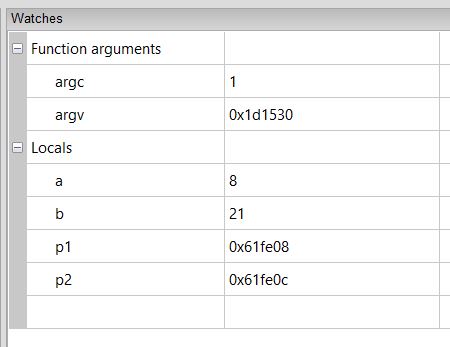
21 printf("%d %d\n", \*p1, \*p2);

22 printf("%d %d\n", a, b);

23 }

Watches:





Console:



6)

1 /\*

2 6. Considere o programa a seguir em que os valores

3 lidos para as variÃ¡veis x e y sÃ£o 3 e 4, respectivamente.

4 Execute o programa no Code Blocks, faÃ§a o rastreio

5 (utilizando debugger e breakpoints) e identifique a saÃ­da.

6 Apresente impressÃµes de tela do rastreio e da saÃ­da.

7 \*/

8

9 #include <stdio.h>

10 #include <stdlib.h>

11

12

13 **void** func(**int** \*px, **int** \*py)

14 {

15 px = py;

16 \*py = (\*py) \* (\*px);

17 \*px = \*px + 2;

18 }

19 **void** main(**void**)

20 {

21 **int** x, y;

22 scanf("%d",&x); //3

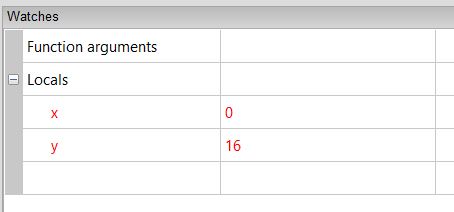
23 scanf("%d",&y); //4

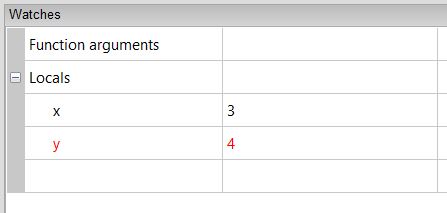
24 func(&x,&y);

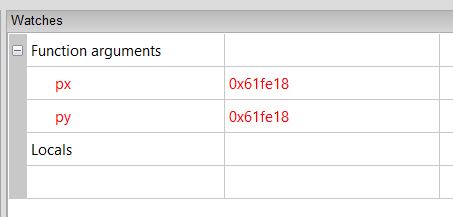
25 printf("x = %d, y = %d", x, y);

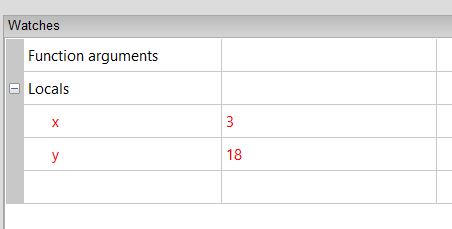
26 }

Watches:

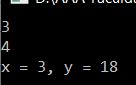








Console:



7)

1 /\*

2 7. Considere um cadastro de alunos matriculados em uma disciplina,

3 com as seguintes informaÃ§Ãµes para cada aluno:

4 â€¢ Nome do aluno: com atÃ© 80 caracteres

5 â€¢ NÃºmero de matrÃ­cula: representado por um valor inteiro

6 â€¢ Notas obtidas em trÃªs provas, P1, P2, P3: representadas por valores reais

7

8 (a) Defina uma estrutura em C, denominada aluno, que tenha os campos apropriados para

9 guardar as informaÃ§Ãµes de um aluno, conforme descrito acima.

10

11 (b) Escreva uma funÃ§Ã£o que receba como parÃ¢metro um ponteiro para

12 uma estrutura do tipo definido no item anterior e imprima na tela

13 do computador uma linha com o nome do aluno e outra linha com a

14 mÃ©dia obtida nas trÃªs provas.

15 Essa funÃ§Ã£o deve seguir o seguinte protÃ³tipo:

16 void imprime (struct aluno\* a);

17

18 (c) Escreva um programa para testar a funÃ§Ã£o.

19 \*/

20

21 #include <stdio.h>

22 #include <stdlib.h>

23 #include <string.h>

24

25 //a

26 **typedef struct** aluno{

27 **char** nome[80];

28 **int** matricula;

29 **float** nota\_p1, nota\_p2, nota\_p3;

30 }Aluno;

31 //b

32 **void** imprime (**struct** aluno\* a){

33 printf("\n Nome do aluno(a): %s \n", a->nome);

34 **float** media;

35 media = ((a->nota\_p1) + (a->nota\_p2) + (a->nota\_p3))/3;

36 printf("\n Media obtida nas 3 provas: %.2f \n", media);

37 }

38 //c

39 **int** main()

40 {

41 Aluno aluno1;

42 Aluno \*a = &aluno1;

43 a->matricula = 5656;

44 strcpy(a->nome, "JOAO DAS DORES");

45 a->nota\_p1 = 9;

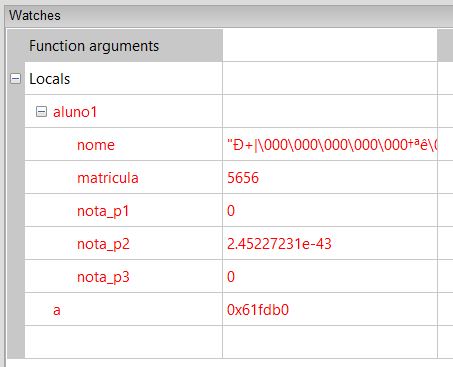
46 a->nota\_p2 = 3;

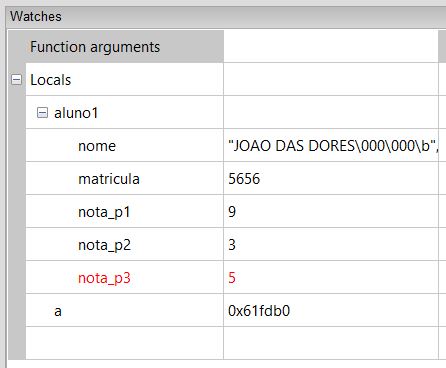
47 a->nota\_p3 = 5;

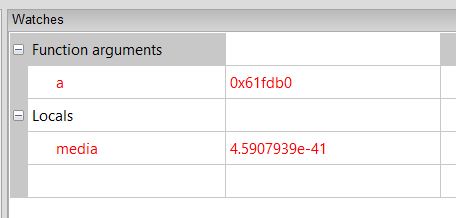
48 imprime(a);

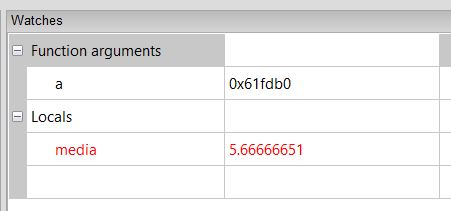
49 }

Watches:

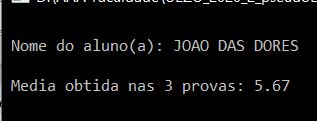








Console:



**LISTA DE EXERCÍCIOS**

**Alocação Dinâmica**

*OBS: Para os exercícios abaixo, crie um projeto “Console Application”, em C, no* ***Code Blocks*** *e, para execução, utilize o Debugger com breakpoints e acompanhe os valores das variáveis na janela Watches. Faça impressão de tela da janela Watches e do console e inclua na resposta junto com o código fonte.*

1)

1 #include <stdio.h>

2 #include <stdlib.h>

3 #include <locale.h>

4 /\*

5 1. Implemente uma função que receba dois vetores de inteiros v1 e v2,

6 mais um inteiro N com o tamanho dos vetores.

7 Sua função deve alocar e retornar um vetor de inteiros de tamanho N

8 onde o elemento na posição i de v3 é a soma dos elementos na posição i

9 de v1 e v2.

10

11 Sua função deve ter a seguinte declaração:

12

13 int \* soma\_vetores(int \*v1, int \*v2, int N);

14

15 Para testar seu programa, crie uma função main() que chame

16 sua função soma\_vetores e imprima os valores somados.

17

18 \*/

19 **int** \* soma\_vetores(**int** \*v1, **int** \*v2, **int** N){

20

21 **int** \*v3 = malloc(N \* **sizeof**(**int**));

22 **for**(**int** i=0;i<N;i++){

23 v3[i] = v2[i] + v1[i];

24 }

25 **return** v3;

26 }

27

28 **int** main()

29 {

30 setlocale(LC\_ALL,"");

31 **int** v1[5],v2[5],i2, N;

32 **int**\* v3;

33 **for**(i2=0;i2<=4;i2++){

34 v1[i2] = 3;

35 }

36 **for**(i2=0;i2<=4;i2++){

37 v2[i2] = 7;

38 }

39 N = **sizeof**(v1)/**sizeof**(v1[0]);

40 v3 = soma\_vetores(v1,v2,N);

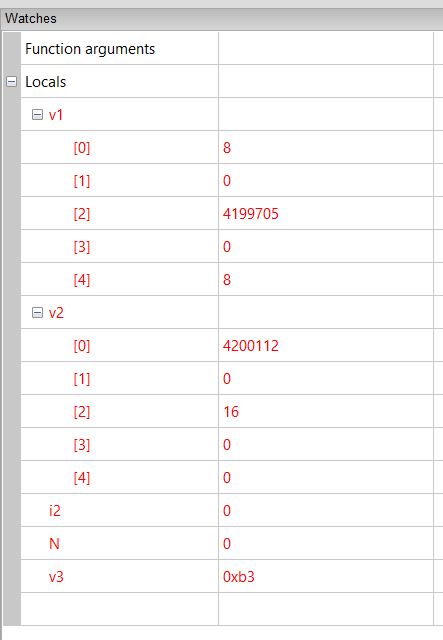
41 **for** (**int** i = 0; i < N; i++)

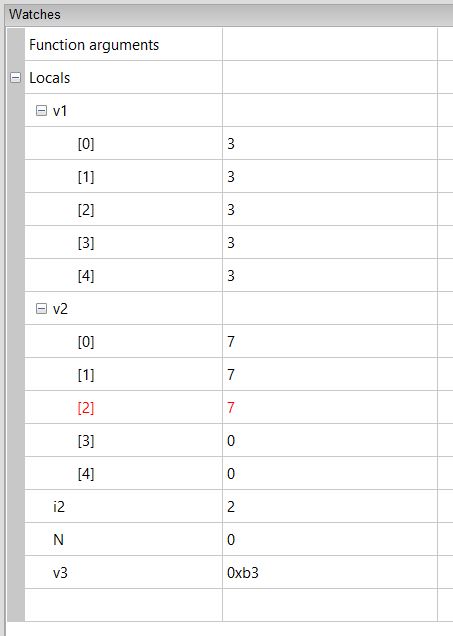
42 printf("Valor na posição %d: %d\n", i, v3[i]);

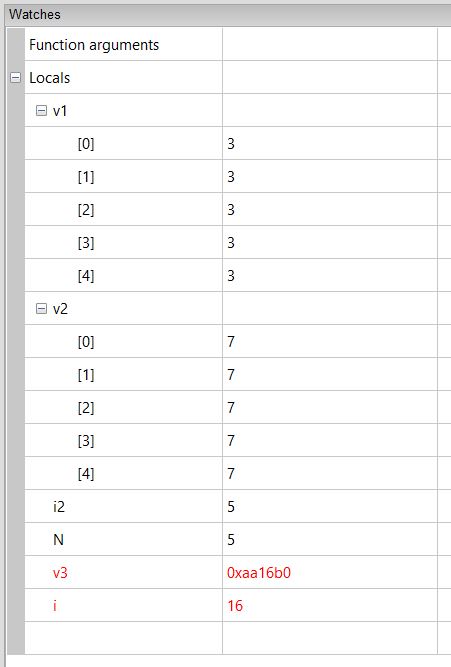
43 free(v3);

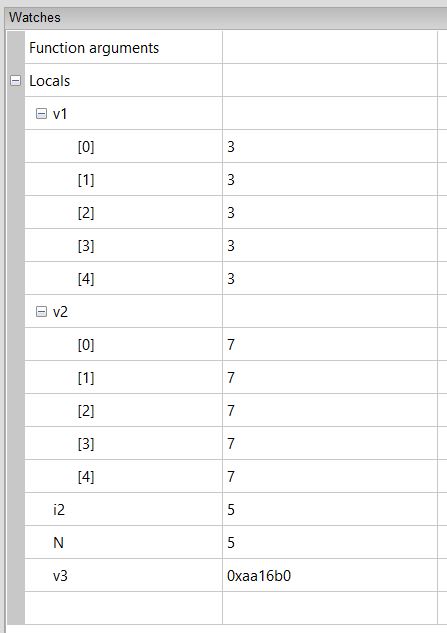
44 }

Watches:

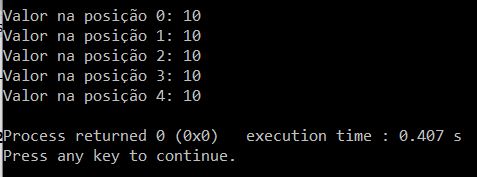








Console:



4)

1 #include <stdio.h>

2 #include <stdlib.h>

3 #include <locale.h>

4 /\*

5 4. Faça um programa para armazenar em memória

6 um vetor de dados contendo 1500 valores do tipo int,

7 usando a função de alocação dinâmica de memória calloc:

8 a. Faça um loop e verifique se o vetor contém realmente

9 os 1500 valores inicializados com zero

10 (conte os 1500 zeros do vetor).

11 b. Atribua para cada elemento do vetor o valor

12 do seu índice junto a este vetor.

13 c. Exibir na tela os endereços dos 10 primeiros

14 e dos 10 últimos elementos do vetor.

15 \*/

16 **int** main()

17 {

18 setlocale(LC\_ALL,"");

19 //a)

20 **int** \*vetor;

21 **int** cont, i;

22

23 vetor = (**int**\*)calloc(1500, **sizeof**(**int**));

24

25 printf("\n Conteúdo do vetor: \n");

26 **for**(i=0; i<=1499;i++){

27 printf("%d", vetor[i]);

28 }

29 printf("\n");

30 **for**(i=0; i<=1499;i++){

31 **if** (vetor[i] == 0){

32 cont += 1;

33 }

34 }

35 **if**(cont == 1500){

36 printf("\n Vetor contém 1500 posições em zero \n");

37 }**else**{

38 printf("\n Vetor não contém 1500 posições em zero \n");

39 }

40

41 //b)

42 **for**(i=0;i<=1500;i++){

43 vetor[i] = i;

44 }

45 **for**(i=0; i<=1499;i++){

46 printf("\n Posição: %d - Conteúdo: %d", i, vetor[i]);

47 }

48

49 //c)

50 printf("\n Os 10 primeiros endereços do vetor: \n");

51 **for**(i=0;i<=10;i++){

52 printf("\n Endereço %d : %p \n", i, (**void**\*)&vetor[i]);

53 }

54 printf("\n Os 10 últimos endereços do vetor: \n");

55 **for**(i=1490;i<=1500;i++){

56 printf("\n Endereço %d : %p \n", i, (**void**\*)&vetor[i]);

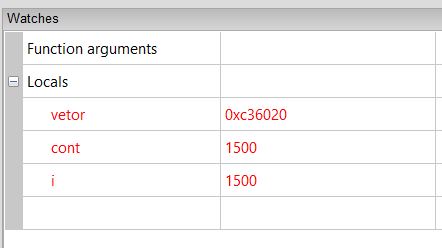
57 }

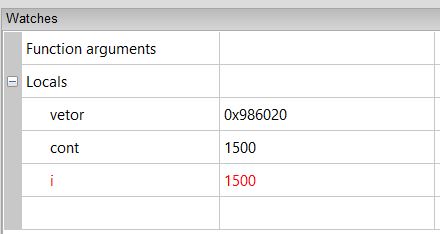
58 free(vetor);

59

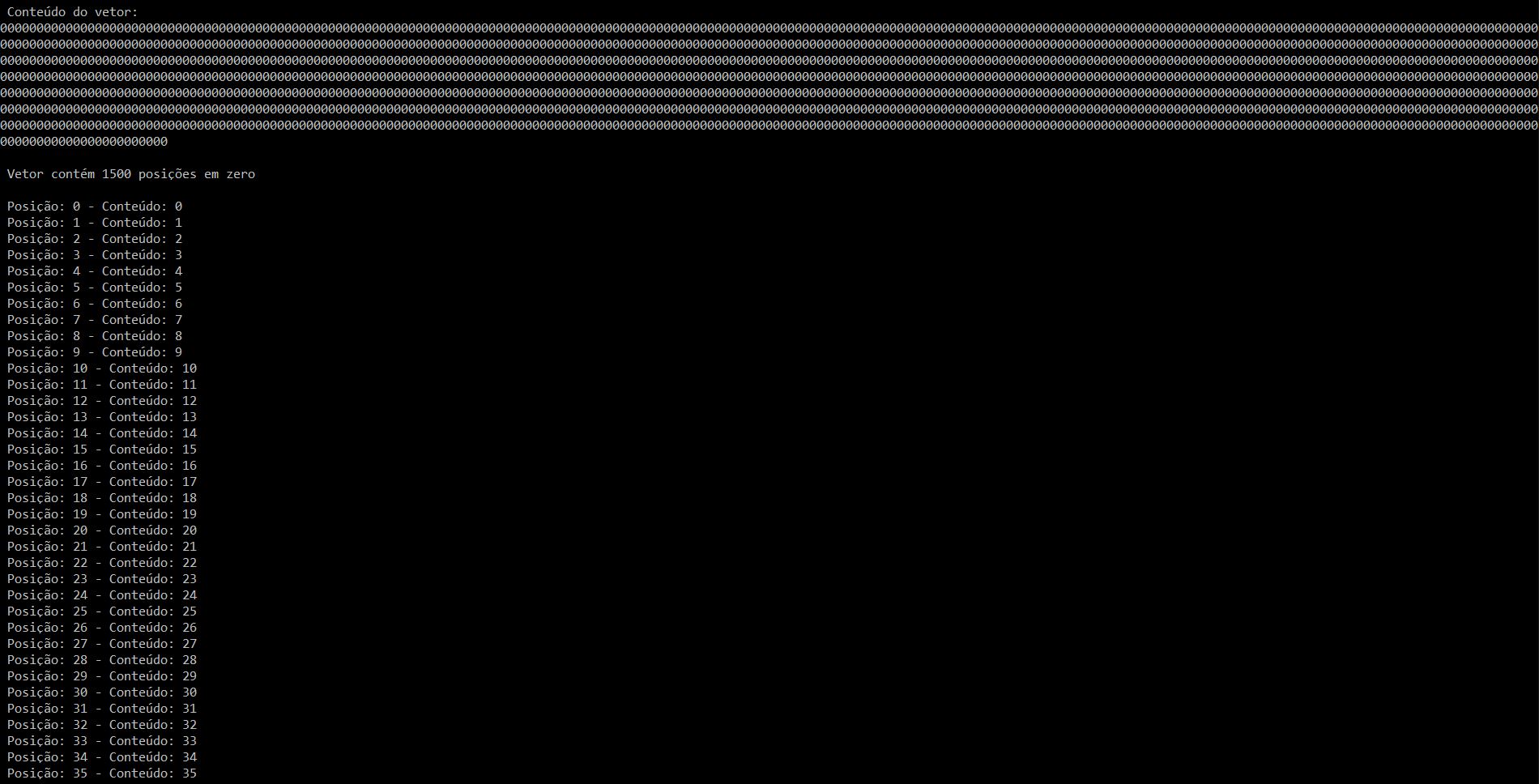
60 }

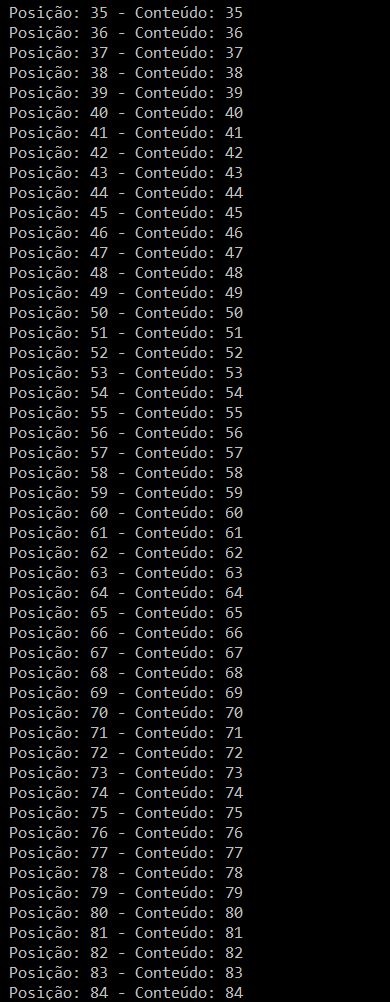
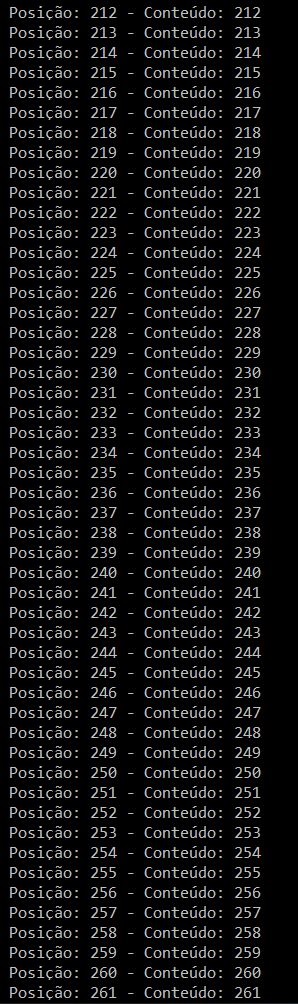
Watches:

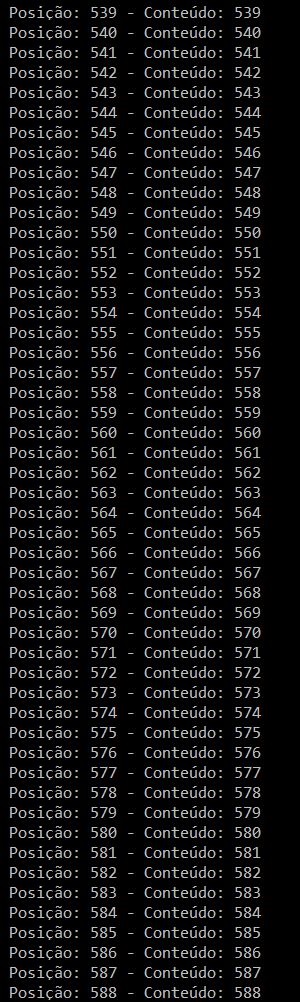
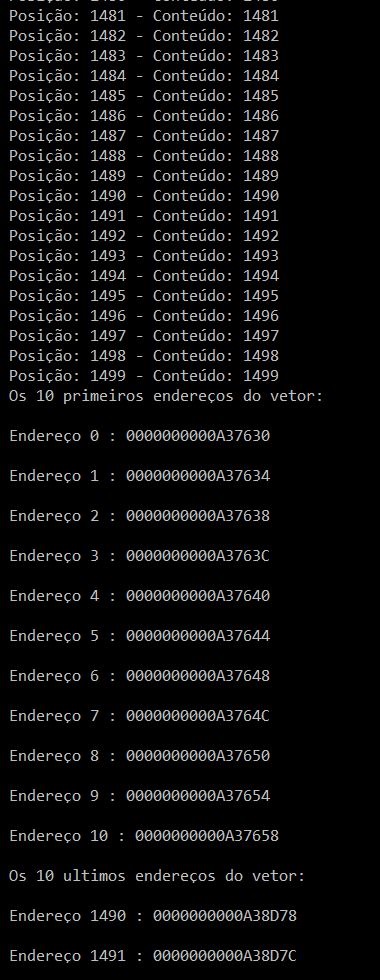


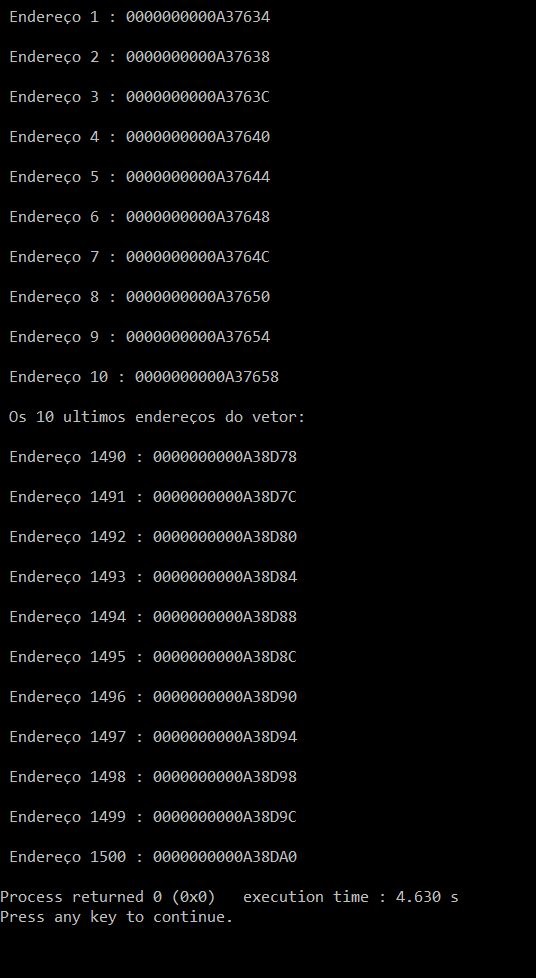


Console:





6)

1 #include <stdio.h>

2 #include <stdlib.h>

3 #include <locale.h>

4

5 /\*

6 6. FaÃ§a um programa que simule â€™virtualmenteâ€™

7 a memÃ³ria de um computador: o usuÃ¡rio comeÃ§a

8 especificando o tamanho da memÃ³ria

9 (define quantos bytes tem a memÃ³ria),

10 e depois ele irÃ¡ ter 2 opÃ§Ãµes: inserir um dado

11 em um determinado endereÃ§o, ou consultar o dado contido

12 em um determinado endereÃ§o. A memÃ³ria deve iniciar com todos

13 os dados zerados.

14 \*/

15

16 **int** main()

17 {

18 setlocale(LC\_ALL,"");

19

20 **int** tam\_memoria, endereco, opcao, i;

21 **int** controle = 0;

22

23 //criando a memoria

24 printf("Defina o tamanho da memÃ³ria: \n");

25 scanf("%d", &tam\_memoria);

26

27 **int** \*vetor\_memoria;

28 vetor\_memoria = (**int**\*)calloc(tam\_memoria, **sizeof**(**int**));

29

30 //imprimindo o estado atual da memoria

31 **for**(i=0;i<tam\_memoria;i++){

32 printf("\n %d \n", vetor\_memoria[i]);

33 }

34

35 **if**(vetor\_memoria == NULL){

36 printf("\n Erro de alocacao de memoria \n");

37 exit(1);

38 }

39 **while**(controle == 0){

40 printf("\n Lista de opÃ§Ãµes: \n");

41 printf("\n 1 - Inserir um dado em um determinado endereÃ§o. \n");

42 printf("\n 2 - Consultar o dado contido em um determinado endereÃ§o. \n");

43 printf("\n 3 - Sair \n");

44 printf("Escolha uma opÃ§Ã£o: \n");

45 scanf("%d", &opcao);

46 **if** (opcao == 1){

47 printf("\n Entre com o endereco onde quer inserir o dado : ");

48 scanf("%d", &endereco);

49 **for**(i=0;i<tam\_memoria;i++){

50 **if**(i == endereco){

51 printf("\n Entre com o valor inteiro para o Ã­ndice %d: ", i);

52 scanf("%d", &vetor\_memoria[i]);

53 }

54 }

55 //lista o conteudo existentes na memoria:

56 **for**(i=0;i<1;i++){

57 printf("\n MEMORIA: ");

58 **for**(i=0;i<tam\_memoria;i++){

59 printf("\n ConteÃºdo atual do Ã­ndice %d: %d \n", i, vetor\_memoria[i]);

60 }

61 }

62 }**else if** (opcao == 2){

63 printf("Digite o endereco do dado a ser consultado na sua memoria: \n");

64 scanf("%d", &endereco);

65 **for**(i=0;i<tam\_memoria;i++){

66 **if**(i == endereco){

67 printf(" \n Dado contido no endereco %d : %d \n", i, vetor\_memoria[i]);

68 }

69 }

70 }**else if**(opcao == 3){

71 printf("\n VocÃª saiu \n");

72 controle = 1;

73 }**else**{

74 printf("\n OpÃ§Ã£o invÃ¡lida. \n");

75 printf("\n VocÃª saiu \n");

76 controle = 1;

77 }

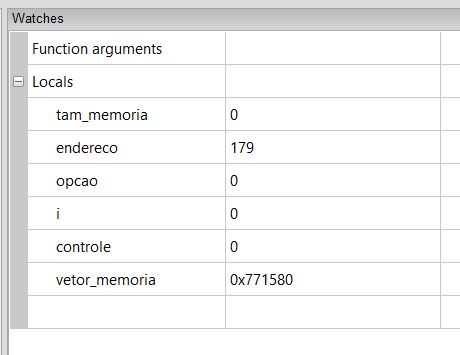
78 }

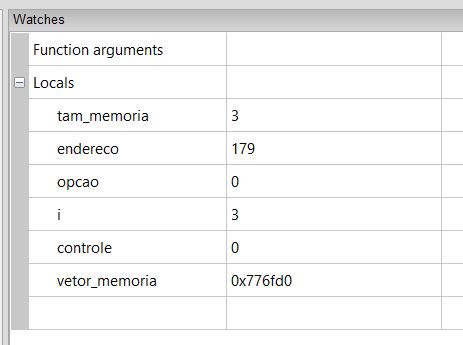
79

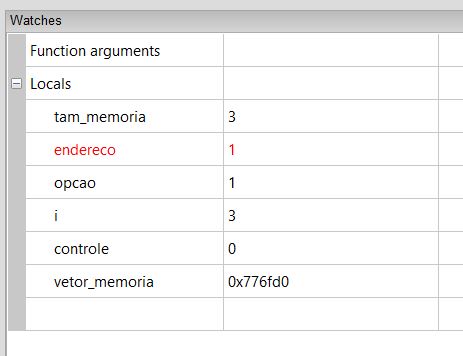
80 free(vetor\_memoria);

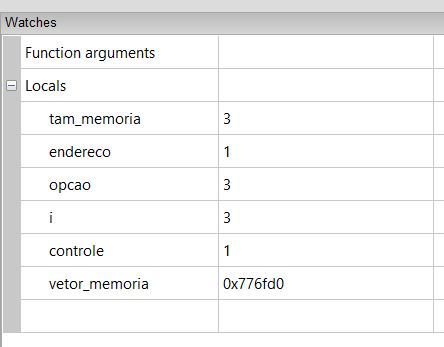
81 }

Watches:

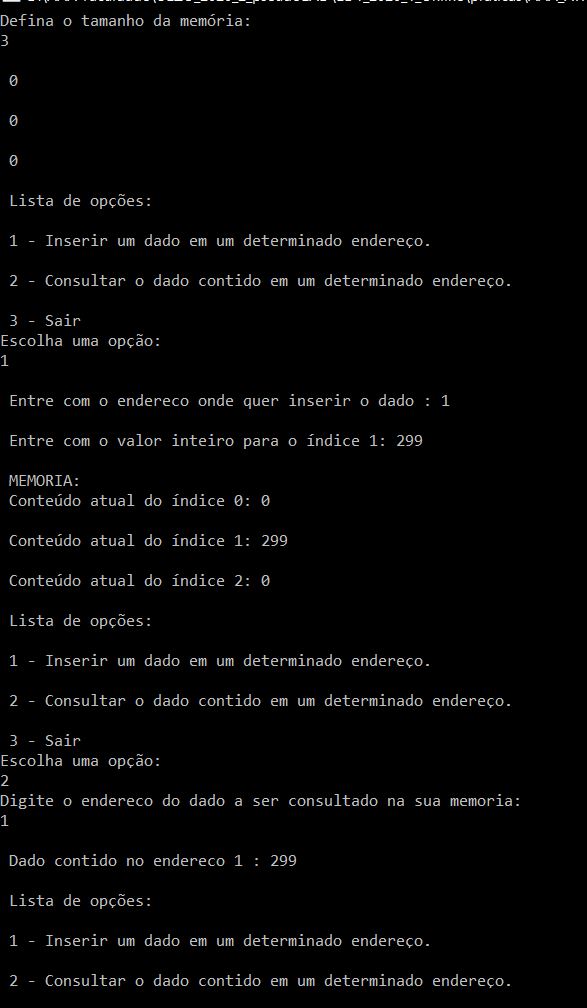


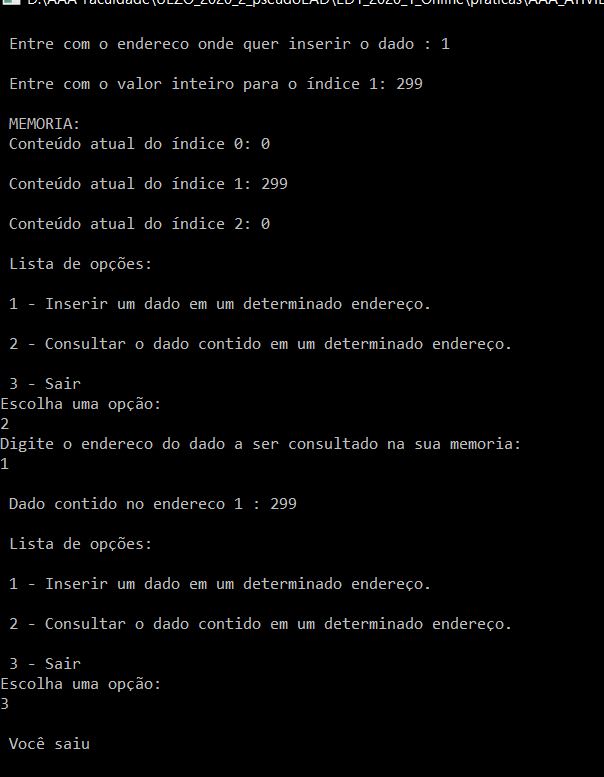






Console:





9)

1 /\*

2 9. Considere um cadastro de produtos de um estoque,

3 com as seguintes informaÃ§Ãµes para cada produto:

4 â€¢ CÃ³digo de identificaÃ§Ã£o do produto: representado por um valor inteiro

5 â€¢ Nome do produto: com ate 50 caracteres

6 â€¢ Quantidade disponÃ­vel no estoque: representado por um nÃºmero inteiro

7 â€¢ PreÃ§o de venda: representado por um valor real

8

9 a. Defina uma estrutura, denominada produto, que tenha

10 os campos apropriados para guardar as informaÃ§Ãµes de um produto

11

12 b. Crie um conjunto de n produtos (n Ã© um valor fornecido pelo usuÃ¡rio)

13 e peÃ§a ao usuÃ¡rio para entrar com as informaÃ§Ãµes de cada produto

14

15 c. Encontre o produto com o maior preÃ§o de venda e imprima na tela

16

17 d. Encontre o produto com a maior quantidade disponÃ­vel no estoque

18 e imprima na tela.

19 \*/

20

21 #include <stdio.h>

22 #include <stdlib.h>

23 #include <string.h>

24

25 //letra a

26 **typedef struct** produto{

27 **int** codigo\_id\_produto, qtd\_disp\_estoque;

28 **char** nome[50];

29 **float** preco\_de\_venda;

30 }Produto;

31

32 **int** main()

33 {

34 **int** n, i, maiorQTDestoque = 0;

35 **float** maiorPreco=0.0;

36

37 //letra b

38 printf("Digite a quantidade de produtos a serem cadastrados: \n");

39 scanf("%d", &n);

40

41 Produto \*p = malloc(n\***sizeof**(Produto));

42

43 **for**(i=0;i<n;i++){

44 printf("\n Entre com o Codigo do Produto: ");

45 scanf("%d", &p[i].codigo\_id\_produto);

46 printf("\n Entre com a quantidade em Estoque do Produto: ");

47 scanf("%d", &p[i].qtd\_disp\_estoque);

48 printf("\n Entre com o nome do produto: ");

49 scanf("%s", p[i].nome);

50 printf("\n Preco: ");

51 scanf("%f", &p[i].preco\_de\_venda);

52 }

53

54 //letra c

55 //define qual produto com o maior preco

56 **for** (i=0;i<n;i++) {

57 **if**(p[i].preco\_de\_venda > maiorPreco) {

58 maiorPreco = p[i].preco\_de\_venda;

59 }

60 }

61 //produto com maior preco

62 **for** (i=0;i<n;i++) {

63 **if**(p[i].preco\_de\_venda == maiorPreco) {

64 printf("\n == Produto com maior preco == \n");

65 printf("Codigo: %d \n", p[i].codigo\_id\_produto);

66 printf("Nome: %s \n", p[i].nome);

67 printf("Preco: %.2f \n", p[i].preco\_de\_venda);

68 printf("Quantidade disponivel em estoque: %d \n", p[i].qtd\_disp\_estoque);

69 }

70 }

71

72 //letra d

73 //define qual produto com o maior quantidade disponivel em estoque

74 **for** (i=0;i<n;i++) {

75 **if**(p[i].qtd\_disp\_estoque > maiorPreco) {

76 maiorQTDestoque = p[i].qtd\_disp\_estoque;

77 }

78 }

79 //produto com maior preco

80 **for** (i=0;i<n;i++) {

81 **if**(p[i].qtd\_disp\_estoque == maiorQTDestoque) {

82 printf("\n == Produto com maior quantidade disponivel em estoque == \n");

83 printf("Codigo: %d \n", p[i].codigo\_id\_produto);

84 printf("Nome: %s \n", p[i].nome);

85 printf("Preco: %.2f \n", p[i].preco\_de\_venda);

86 printf("Quantidade disponivel em estoque: %d \n", p[i].qtd\_disp\_estoque);

87 }

88 }

89

90 printf("\n ====== DADOS DIGITADOS ====== \n");

91 **for**(i=0;i<n;i++){

92 printf("\n ================");

93 printf("\n ==== Produto %d ==== ", i);

94 printf("\n Codigo: %d", p[i].codigo\_id\_produto);

95 printf("\n Quantidade em estoque: %d", p[i].qtd\_disp\_estoque);

96 printf("\n Nome: %s", p[i].nome);

97 printf("\n Preco: %.2f", p[i].preco\_de\_venda);

98 printf("\n ================");

99 }

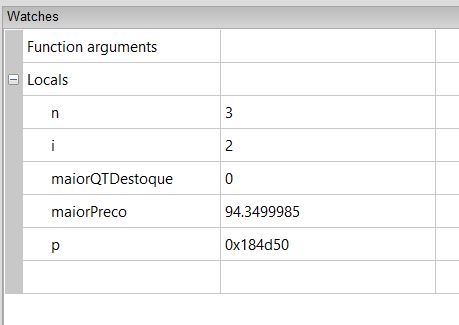
100

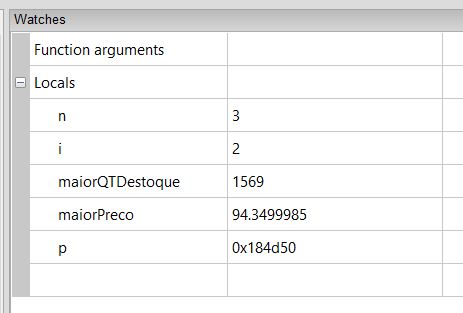
101 free(p);

102 }

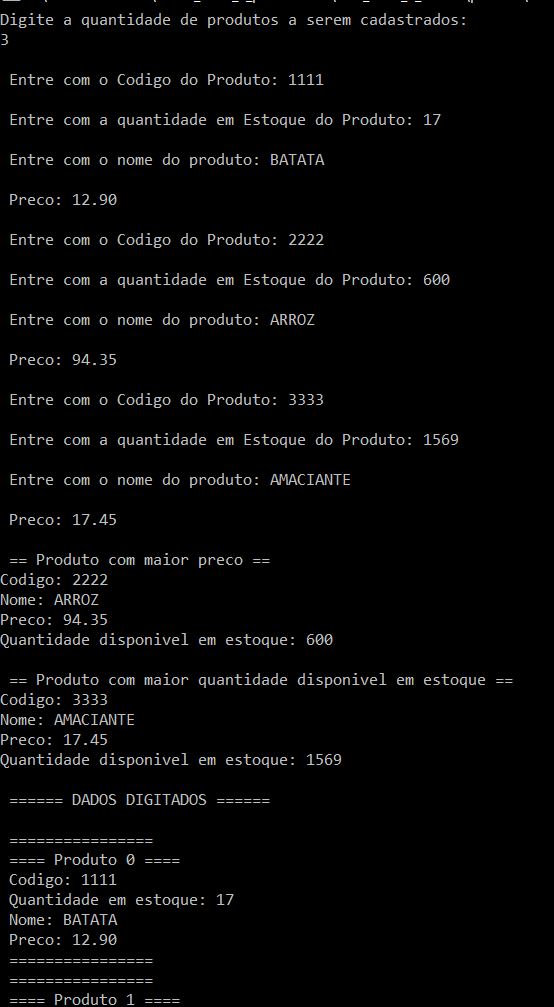
Watches:

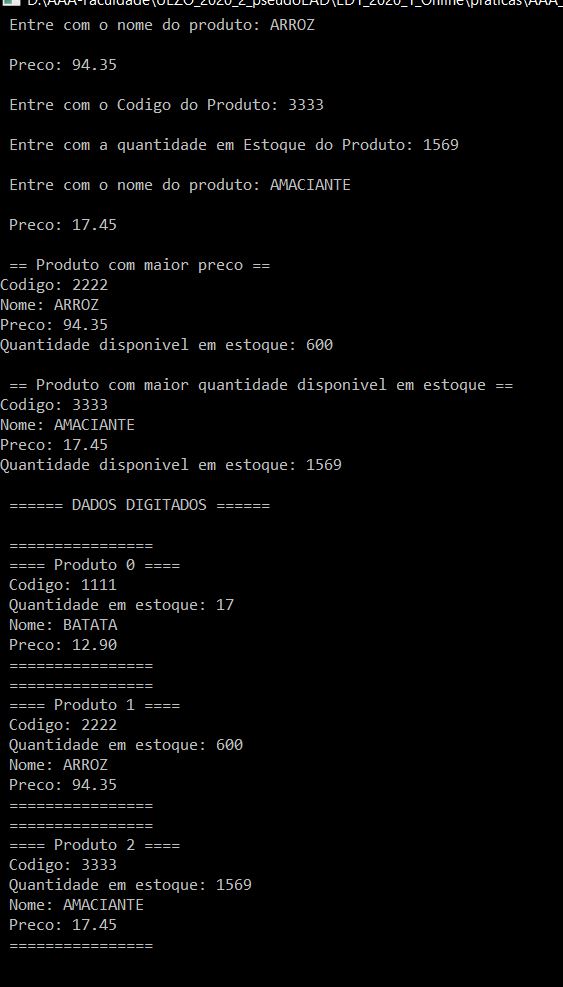






Console:





11)

1 /\*

2 11. O cÃ³digo abaixo implementa parte de uma agenda de telefones bem simples.

3 Cada registro da agenda (contato) Ã© composto apenas de nome e telefone.

4 O usuÃ¡rio poderÃ¡ inserir, remover ou listar os contatos que serÃ£o mantidos

5 em um vetor de ponteiros de contatos, alocado dinamicamente. Pede-se:

6

7 a. Implementar a funÃ§Ã£o inic\_agenda() que inicializa a agenda,

8 alocando um espaÃ§o inicial em memÃ³ria para BLOCK ponteiros

9 de contatos.

10

11 b. Implementar a funÃ§Ã£o insere() que insere um contato

12 fornecido pelo usuÃ¡rio. Lembrar que tal funÃ§Ã£o deve verificar se

13 o espaÃ§o de memÃ³ria alocado para os ponteiros de contatos

14 serÃ¡ esgotado com a inserÃ§Ã£o do novo contato, devendo reajusta-lo,

15 se necessÃ¡rio, em incrementos de tamanho BLOCK.

16

17 c. Fazer o rastreio da funÃ§Ã£o deleta(), mostrando o que

18 ocorre se o usuÃ¡rio optar por deletar o terceiro registro

19 de uma agenda com os cinco registros abaixo:

20

21 Maria 2222-2222

22 JosÃ© 3333-3333

23 Danilo 4444-4444

24 Carla 5555-5555

25 JoÃ£o 6666-6666

26

27 \*/

28

29 #include <stdio.h>

30 #include <stdlib.h>

31 #define MAX 3 //numero mÃ¡ximo de BLOCKs que podem ser alocados

32 #define BLOCK 5

33

34 /\*

35 realloc ok

36

37 \*/

38

39 **typedef struct** \_contato{

40 **char** nome[30];

41 **char** tel[8];

42 }contato;

43

44 contato \*\*agenda;

45 **int** num\_contatos = 0;

46 **int** num\_blocos = 0;

47 **void** inic\_agenda(**void**);

48 **void** insere(**void**);

49 **void** deleta(**void**);

50 **void** lista(**void**);

51 **int** item\_menu(**void**);

52

53 **int** main(**void**)

54 {

55 **int** item;

56 inic\_agenda();

57 **for**(;;)

58 {

59 item = item\_menu();

60 **switch**(item)

61 {

62 **case** 1:

63 insere();

64 **break**;

65 **case** 2:

66 deleta();

67 **break**;

68 **case** 3:

69 lista();

70 **break**;

71 **case** 4:

72 exit(0);

73 }

74 }

75 }

76

77 //letra a

78 //Inicializa a agenda

79 **void** inic\_agenda(**void**)

80 {

81 agenda = (contato \*\*)malloc(BLOCK\***sizeof**(contato\*));

82 }

83

84 //letra b

85 //Insere um novo contato

86 **void** insere(**void**)

87 {

88 **if**((num\_contatos != 0) &&((num\_contatos%BLOCK) == 0))

89 {

90 **if** (num\_blocos >= MAX)

91 {

92 printf("\n Agenda cheia \n");

93 **return**;

94 }

95 agenda = (contato \*\*)realloc(agenda,(num\_blocos + 1) \* BLOCK \* (**sizeof**(contato\*)));

96 num\_blocos++;

97 }

98 agenda[num\_contatos]= (contato\*) malloc(**sizeof**(contato));

99 printf("\n Entre com o nome: ");

100 gets(agenda[num\_contatos]->nome);

101 printf("Entre com o tel: ");

102 gets(agenda[num\_contatos]->tel);

103 num\_contatos++;

104 }

105

106

107 /\* Retorna item de menu selecionado\*/

108 **int** item\_menu(**void**)

109 {

110 **char** s[80];

111 **int** c;

112 printf("\n");

113 printf("1. Inserir um contato\n");

114 printf("2. Excluir um contato\n");

115 printf("3. Listar contatos\n");

116 printf("4. Sair\n");

117

118 **do**

119 {

120 printf("\nEntre com sua escolha: ");

121 gets(s);

122 c = atoi(s);

123 }

124 **while**(c<0 || c>4);

125

126 **return** c;

127 }

128

129 /\*

130 letra c

131 Quando contato for excluido o ultimo contado da lista passa a ocupar seu lugar.

132 \*/

133

134 /\* Apaga um contato \*/

135 **void** deleta(**void**)

136 {

137 **int** indice;

138 **char** s[10];

139

140 **if** (num\_contatos ==0)

141 {

142 printf("\nAgenda vazia\n");

143 **return**;

144 }

145 printf("\nEntre com o no. do contato: ");

146 gets(s);

147 indice = atoi(s);

148 free(agenda[indice-1]);

149 agenda[indice-1] = agenda[num\_contatos -1];

150 num\_contatos--;

151

152 **if** (num\_contatos<(num\_blocos \* BLOCK))

153 {

154 agenda =(contato \*\*)realloc(agenda,(num\_blocos - 1)\***sizeof**(contato \*));

155 }

156 }

157

158 /\* Mostra a lista de contatos na tela. \*/

159 **void** lista(**void**)

160 {

161 **int** t;

162 **if** (num\_contatos ==0)

163 {

164 printf("\nAgenda vazia\n");

165 **return**;

166 }

167 **for**(t=0; t<num\_contatos; ++t)

168 {

169 printf("(%d) %s %s\n", t+1, agenda[t]->nome, agenda[t]->tel);

170 }

171 printf("\n\n");

172 }

Console:

