Prova sobre Arquitetura da Memória

Prof. Abrantes Araújo Silva Filho

Uma empresa de logística é responsável pelo transporte e armazenamento de produtos alimentícios que precisam permanecer refrigerados. Para garantir a qualidade desses produtos, a empresa utiliza sensores eletrônicos que registram periodicamente a temperatura no interior dos compartimentos de armazenamento.

Para determinada categoria de produtos, a temperatura deve permanecer entre 2,0 °C e 8,0 °C, inclusive. Temperaturas inferiores ou superiores a esses limites representam condições inadequadas de armazenamento e devem ser sinalizadas.

Os sensores utilizados pela empresa possuem um pequeno erro sistemático de calibração. Para compensar esse erro, o sistema deve adicionar um valor de correção às temperaturas informadas pelos sensores.

Como o número de medições pode variar a cada execução, o programa deve reservar memória suficiente para armazenar exatamente a quantidade de medições solicitada pelo operador.

Além disso, como o sistema poderá analisar diferentes lotes de medições durante uma mesma execução, a empresa deseja contabilizar quantas análises foram realizadas desde a inicialização do programa.

Você foi contratado para desenvolver um programa em C23, executado em ambiente Linux, para registrar e analisar essas medições.

I. Requisitos funcionais

1. Permitir que o operador realize sucessivas análises de lotes de medições. Ao final de cada análise, o programa deverá perguntar se o operador deseja analisar outro lote.

2. Para cada lote, solicitar a quantidade de medições que serão realizadas, aceitando valores inteiros entre 1 e 1.000.

3. Solicitar ao operador um valor de correção para a calibração dos sensores, em graus Celsius, entre -10,0 °C e 10,0 °C.

4. Ler as temperaturas registradas pelos sensores (todas entre -100,0 °C e 100°C), aplicar a correção informada e armazenar os valores corrigidos em memória.

5. Para cada temperatura corrigida, verificar se ela está dentro do intervalo permitido de 2,0 °C a 8,0 °C, inclusive.

6. Contabilizar quantas medições apresentaram temperaturas fora do intervalo permitido.

7. Calcular a temperatura média das medições corrigidas.

8. Ao final de cada análise, apresentar um relatório contendo:
   - Quantidade total de medições realizadas no lote

- Temperatura mínima permitida
- Temperatura máxima permitida
- Valor de correção utilizado
- Temperatura média das medições corrigidas
- Quantidade de medições fora do intervalo permitido
- Percentual de medições fora do intervalo permitido
- Quantidade total de análises realizadas desde a inicialização do programa.

9. Liberar os recursos de memória alocados dinamicamente para cada lote antes de iniciar a análise do lote seguinte.

II. Requisitos técnicos obrigatórios

1. Parâmetros constantes:
   Os limites mínimo e máximo de temperatura deverão ser armazenados em um vetor global denominado `limites_temperatura`, declarado com o qualificador `const` e inicializado com os valores `2,0` e `8,0`.
   Esses valores não poderão ser modificados durante a execução do programa.

2. Variável global inicializada:
   O valor de correção dos sensores deverá ser armazenado em uma variável global denominada `correcao_sensor`, inicialmente configurada com o valor `0,5`.
   Após receber o valor de correção informado pelo operador, o programa deverá atualizar essa variável global.

3. Variáveis globais não inicializadas explicitamente:
   O programa deverá declarar duas variáveis globais, sem inicializadores explícitos:
   `total_alertas`: para contabilizar as medições fora do intervalo permitido no lote atual.
   `soma_temperaturas`: para acumular as temperaturas corrigidas no lote atual.
   Essas variáveis deverão ser utilizadas durante o processamento das medições e reinicializadas adequadamente antes da análise de cada novo lote.

4. Armazenamento dinâmico:
   As temperaturas corrigidas deverão ser armazenadas em um vetor alocado dinamicamente, com capacidade suficiente para armazenar a quantidade de medições informada pelo operador. A alocação deverá ser realizada utilizando `malloc()`. O programa deverá verificar se a alocação foi realizada com sucesso e liberar a memória utilizando `free()` quando ela não for mais necessária. Não será permitido utilizar vetores globais ou vetores locais de tamanho fixo para armazenar as medições.

5. Variáveis locais automáticas:
   O programa deverá utilizar variáveis locais automáticas para armazenar:
   - A quantidade de medições informada pelo operador.
   - O índice utilizado para percorrer o vetor de medições.
   - A temperatura lida em cada iteração.
   - A temperatura média calculada.

   Essas variáveis deverão ser declaradas no interior das funções, sem a utilização do especificador `static`.

6. Variável local com duração de armazenamento estática:
   A função **`analisar_temperaturas()`** deverá manter um contador denominado **`total_analises`**, responsável por registrar quantas vezes essa função foi executada com sucesso desde a inicialização do programa. Esse contador deverá atender obrigatoriamente às seguintes condições:
   - Ser declarado no interior da função **`analisar_temperaturas()`**.
   - Ser inicializado com o valor zero.
   - Preservar seu valor entre chamadas sucessivas da função.
   - Ser incrementado uma única vez a cada análise concluída com sucesso.
   - Não ser declarado como variável global.
   - Não ser acessado diretamente por outras funções.

   A função deverá disponibilizar a quantidade de análises realizadas para que essa informação possa aparecer no relatório final de cada lote.

7. Organização em funções:
   O programa deverá conter, no mínimo, as seguintes funções:
   - **`main()`**: responsável pelo controle geral do programa e pela repetição das análises.
   - **`registrar_temperaturas()`**: responsável pela leitura das temperaturas, aplicação da correção e armazenamento no vetor.
   - **`analisar_temperaturas()`**: responsável por verificar os limites, contabilizar alertas, calcular a média e atualizar o contador de análises.
   - **`exibir_relatorio()`**: responsável pela apresentação dos resultados.

   Os parâmetros e os tipos de retorno deverão ser definidos pelo aluno de acordo com as necessidades da implementação.

III. Outros requisitos

O código deverá ser organizado em funções, utilizar nomes de variáveis significativos e respeitar todos os requisitos funcionais e técnicos apresentados.

O programa deve poder ser executado com pelo menos três lotes consecutivos, sem encerrar o processo entre eles.

O programa deve ser compilado e executado, sem erros, em um ambiente Linux recente, usando o padrão C23 e o compilador GCC ou Clang.

É permitido o uso da biblioteca **`<CRpaic.h>`** ou da **`<cs50.h>`** para facilitar a obtenção de dados do usuário.

IV. Exemplo de execução

Para que você consiga entender o fluxo de execução do programa em um terminal Linux, a próxima página traz impresso tudo o que é exibido pelo programa em um terminal. As entradas que o usuário digitou estão destacadas em **negrito sublinhado**.

Estude a saída do programa a seguir para você entender o que o programa solicita de informações ao usuário e o que ele faz e imprime no terminal.

=== MONITORAMENTO DE TEMPERATURA ===

Quantidade de medições: **3**
Correção do sensor (°C): **0.5**

Temperatura 1: **3.0**
Temperatura 2: **9.0**
Temperatura 3: **5.0**

=== RELATÓRIO ===
Total de medições coletadas : 3
Temperatura mínima permitida: 2.00 °C
Temperatura máxima permitida: 8.00 °C
Correção aplicada ao sensor : 0.50 °C

Temperatura média: 6.17 °C
Menor temperatura: 3.50 °C
Maior temperatura: 9.50 °C
Medições problemáticas: 1
Percentual de alertas : 33.33 %

Total de análises: 1

Deseja analisar outro lote (S/N)? **S**

Quantidade de medições: **2**
Correção do sensor (°C): **0.0**

Temperatura 1: **2.0**
Temperatura 2: **4.5**

=== RELATÓRIO ===
Total de medições coletadas : 2
Temperatura mínima permitida: 2.00 °C
Temperatura máxima permitida: 8.00 °C
Correção aplicada ao sensor : 0.00 °C

Temperatura média: 3.25 °C
Menor temperatura: 2.00 °C
Maior temperatura: 4.50 °C
Medições problemáticas: 0
Percentual de alertas : 0.00 %

Total de análises: 2

Deseja analisar outro lote (S/N)? **N**

Programa encerrado.

V. Programa de referência

A seguir é apresentado o código fonte completo do programa que implementa os requisitos funcionais e técnicos exigidos nas seções I, II, III e IV deste documento. O programa utiliza a biblioteca **`<CRpaic.h>`** para falicitar a entrada de dados do usuário (fornece as funções **`get_int()`**, **`get_double()`**, **`get_char()`**).

Estude o código fonte do programa mas lembre-se de que esta é uma avaliação sobre Arquitetura de Memória: preste mais atenção à declaração e inicialização das variáveis e nas regiões de memória onde essas variáveis estão alocadas.

```c
#include <CRpaic.h>
#include <float.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>

/* Constantes simbólicas */
#define QTDREF 2
#define IMIN 0
#define IMAX 1
#define TMIN 2.0
#define TMAX 8.0
#define LOTE_MIN 1
#define LOTE_MAX 1000
#define CORRECAO 0.5
#define TEMPMIN -100.0
#define TEMPMAX 100.0
#define SENSORMIN -10.0
#define SENSORMAX 10.0

/* Tipos de dados */
struct st_resultado
{
    double media;
    double minima;
    double maxima;
    int qtd_alertas;
    int qtd_analises;
};
typedef struct st_resultado resultado_t;

/* Variáveis globais */
double const limites_temperatura[QTDREF] = {TMIN, TMAX};
double correcao_sensor = CORRECAO;
int total_alertas;
double soma_temperaturas;

/* Protótipos das funções */
void registrar_temperaturas (double *medicoes, int qtd_medicoes);
resultado_t analisar_temperaturas (double const *medicoes, int qtd_medicoes);
void exibir_relatorio (int qtd_medicoes, resultado_t resultado);

/* FUNÇÃO MAIN */
int main (void)
{
    printf("%s\n", "=== MONITORAMENTO DE TEMPERATURA ===");

    int qtd_medicoes;
    char stop;

    do
    {
        do
        {
```

```
55              qtd_medicoes = get_int("\nQuantidade de medições: ");
56          }
57          while (qtd_medicoes < LOTE_MIN || qtd_medicoes > LOTE_MAX);
58
59          double *medicoes = malloc(sizeof(*medicoes) * (size_t) qtd_medicoes);
60          if (!medicoes)
61          {
62              fprintf(stderr, "ERRO: não foi possível alocar a memória.\n");
63              return EXIT_FAILURE;
64          }
65
66          do
67          {
68              correcao_sensor = get_double("Correção do sensor (°C): ");
69          }
70          while (!isfinite(correcao_sensor)  ||
71                 correcao_sensor < SENSORMIN || correcao_sensor > SENSORMAX);
72
73          registrar_temperaturas(medicoes, qtd_medicoes);
74          resultado_t resultado = analisar_temperaturas(medicoes, qtd_medicoes);
75
76          free(medicoes);
77          medicoes = NULL;
78
79          exibir_relatorio(qtd_medicoes, resultado);
80
81          stop = get_char("\nDeseja analisar outro lote (S/N)? ");
82      }
83      while (stop == 'S' || stop == 's');
84
85      printf("\nPrograma encerrado.\n");
86      return EXIT_SUCCESS;
87  }
88
89  /* Implementação das funções */
90  void registrar_temperaturas (double *medicoes, int qtd_medicoes)
91  {
92      printf("\n");
93      double temp = 0;
94      for (int i = 0; i < qtd_medicoes; i++)
95      {
96          do
97          {
98              temp = get_double("Temperatura %d: ", i + 1);
99          }
100         while (!isfinite(temp) ||
101                temp < TEMPMIN || temp > TEMPMAX);
102         medicoes[i] = temp + correcao_sensor;
103     }
104 }
105
106 resultado_t analisar_temperaturas (double const *medicoes, int qtd_medicoes)
107 {
108     static int total_analises = 0;
109     total_alertas = 0;
110
111     resultado_t resultado = {0};
112     soma_temperaturas = 0.0;
113     double tempmin = DBL_MAX;
114     double tempmax = -DBL_MAX;
115
116     for (int i = 0; i < qtd_medicoes; i++)
117     {
118         if (medicoes[i] < tempmin)
119             tempmin = medicoes[i];
120         if (medicoes[i] > tempmax)
121             tempmax = medicoes[i];
```

```
122         if (medicoes[i] < limites_temperatura[IMIN] ||
123             medicoes[i] > limites_temperatura[IMAX])
124             total_alertas++;
125         soma_temperaturas += medicoes[i];
126     }
127
128     total_analises++;
129
130     resultado.media = soma_temperaturas / (double) qtd_medicoes;
131     resultado.minima = tempmin;
132     resultado.maxima = tempmax;
133     resultado.qtd_alertas = total_alertas;
134     resultado.qtd_analises = total_analises;
135
136     return resultado;
137 }
138
139 void exibir_relatorio (int qtd_medicoes, resultado_t resultado)
140 {
141     printf("\n");
142     printf("=== RELATÓRIO ===\n");
143     printf("Total de medições coletadas : %d\n", qtd_medicoes);
144     printf("Temperatura mínima permitida: %.2f °C\n",
145            limites_temperatura[IMIN]);
146     printf("Temperatura máxima permitida: %.2f °C\n",
147            limites_temperatura[IMAX]);
148     printf("Correção aplicada ao sensor : %.2f °C\n", correcao_sensor);
149     printf("\nTemperatura média: %.2f °C\n", resultado.media);
150     printf("Menor temperatura: %.2f °C\n", resultado.minima);
151     printf("Maior temperatura: %.2f °C\n", resultado.maxima);
152     printf("Medições problemáticas: %d\n", resultado.qtd_alertas);
153     printf("Percentual de alertas : %.2f %%\n",
154            (double) resultado.qtd_alertas / (double) qtd_medicoes * 100.0);
155     printf("\nTotal de análises: %d\n", resultado.qtd_analises);
156 }
```

Agora que você já estudou o código, responda às questões a seguir, sobre a Arquitetura da Memória nos computadores. As perguntas foram elaboradas para avaliar não apenas a memorização das características de cada região, mas a capacidade de relacionar a organização da memória com as declarações, instruções e operações efetivamente utilizadas no programa.

VI. Perguntas discursivas

Questão 1:

Considere as funções abaixo, implementadas no programa:

- **`main()`**
- **`registrar_temperaturas()`**
- **`analisar_temperaturas()`**
- **`exibir_relatorio()`**

Explique qual é a relação entre essas funções e o segmento de memória **TEXT** do executável. Identifique uma instrução do seu código-fonte que envolva um cálculo aritmético e explique como essa instrução se relaciona com as instruções de máquina executadas pelo processador.

Questão 2:

Durante a execução, a função **`main()`** chama a função **`analisar_temperaturas()`**. Explique o que acontece com o fluxo de execução do processador durante essa chamada, considerando o

endereço das instruções, o **contador de programa** (Program Counter) e o retorno à função `main()`.

As instruções de máquina correspondentes à função `analisar_temperaturas()` precisam ser copiadas para outra região de memória a cada chamada? Justifique.

Questão 3:

Identifique no código a variável `correcao_sensor`. Identifique qual o segmento de memória onde essa variável está e explique por que essa variável normalmente pertence a esse segmento. Descreva como seu valor inicial é disponibilizado quando o programa começa a executar.

Em seguida, explique o que acontece com o conteúdo dessa variável quando o operador informa um novo valor de correção.

Questão 4:

Considere a variável `correcao_sensor` e a variável `total_analises`, declaradas na função `analisar_temperaturas()`. Qual delas é global e qual é local?

Embora ambas possuam duração de armazenamento estática, seus identificadores apresentam diferentes regras de visibilidade no código-fonte. Explique essas diferenças, considerando o escopo e a duração de armazenamento de cada objeto.

Em seguida, responda: se a variável `total_analises` fosse inicializada explicitamente com um valor diferente de zero, em qual seção da memória ela normalmente seria armazenada? Justifique sua resposta.

Questão 5:

Identifique no programa a declaração do vetor `limites_temperatura`. Explique por que seus elementos não podem ser modificados por meio desse identificador. Indique também em qual segmento de memória esse vetor está armazenado e explique porque esse vetor é realmente protegido contra alterações durante a execução do programa.

O que aconteceria se o programa tentasse modificar diretamente um elemento desse vetor? Diferencie as verificações realizadas pelo compilador das possíveis proteções oferecidas pelo sistema operacional durante a execução.

Questão 6:

Considere as seguintes declarações:

```
double const limites_temperatura[2] = {2.0, 8.0};

double correcao_sensor = 0.5;
```

Explique as diferenças entre esses dois objetos quanto à possibilidade de modificação, à duração de armazenamento e à localização típica na memória do processo.

Em seguida, explique por que o uso do qualificador `const`, isoladamente, não garante que todo objeto declarado com esse qualificador seja armazenado na região **DATA RO**.

Questão 7:

Identifique no código as variáveis `total_alertas` e `soma_temperaturas`.

Identifique a região de memória na qual essas variáveis estão e explique por que essas variáveis normalmente pertencem à essa região. Explique também quais são os valores que elas possuem quando o programa inicia sua execução, mesmo não tendo recebido inicializadores explícitos.

Compare esse comportamento com o de uma variável local automática declarada sem inicializador dentro de uma função.

Questão 8:

Considere as seguintes declarações:

```
int contadores_a[1000000];

int contadores_b[1000000] = {0};
```

Ambas foram realizadas no escopo global.

Explique em quais segmentos da memória esses vetores normalmente seriam armazenados e discuta por que esse segmento permite representar grandes quantidades de dados inicialmente zerados sem exigir que todos esses bytes sejam armazenados fisicamente no arquivo executável.

Em seguida, considere a declaração utilizada na função `analisar_temperaturas()`:

```
static int total_analises = 0;
```

Explique por que essa variável está armazenada na seção BSS, apesar de ter sido declarada no interior de uma função e possuir um inicializador explícito.

Questão 9:

Identifique o trecho no programa responsável pela alocação dinâmica do vetor de temperaturas.

Explique como a função `malloc()` permite reservar memória durante a execução e por que essa estratégia é apropriada para um programa em que a quantidade de medições somente será conhecida após a entrada de dados.

Em sua resposta, diferencie o ponteiro utilizado para acessar o vetor da região de memória em que os elementos do vetor são efetivamente armazenados.

Questão 10:

Considere que o operador solicite o registro de 1.000 medições e que o programa consiga alocar a memória necessária.

Explique o que acontece com essa memória quando a função **`registrar_temperaturas()`** termina sua execução.

A memória alocada é automaticamente liberada quando a função retorna? Qual é a função responsável por liberar essa memória e quais seriam as consequências de não realizar essa operação?

Questão 11:

Identifique pelo menos três variáveis locais automáticas declaradas nas funções do seu programa.

Explique como essas variáveis se relacionam com os registros de ativação (stack frames) criados durante a execução das funções.

Considere uma chamada de **`main()`** para **`registrar_temperaturas()`**. Descreva como a pilha de execução pode ser utilizada para armazenar informações necessárias à chamada e ao retorno da função.

Questão 12:

Considere a seguinte implementação hipotética de uma função para registrar temperaturas:

```
double *registrar_temperaturas(size_t n)
{
    double temperaturas[1000];

    /* Leitura e armazenamento das temperaturas */

    return temperaturas;
}
```

Explique por que essa implementação apresenta um problema relacionado ao tempo de vida dos objetos e à organização da memória.

Compare essa implementação com a estratégia utilizada no programa, que emprega **`malloc()`**.

Em sua resposta, explique por que retornar o endereço de um objeto local automático é diferente de retornar um ponteiro para uma região alocada dinamicamente.

`Questão 13:`

Considere a variável **`total_analises`**, declarada em sua implementação da função **`analisar_temperaturas()`**. Explique por que essa variável preserva seu valor entre diferentes chamadas da função, mesmo sendo uma variável local. Em sua resposta, diferencie os conceitos de:

- Escopo de uma variável.
- Duração de armazenamento de um objeto.

- Localização do objeto na memória.

Explique também o que aconteceria com o comportamento do programa se o especificador **`static`** fosse removido dessa declaração.

Questão 14:

Considere as seguintes declarações hipotéticas, realizadas no interior da função **`analisar_temperaturas()`**:

```
int contador_a = 0;
static int contador_b = 0;
```

Suponha que a função seja executada cinco vezes e que, em cada execução, ambos os contadores sejam incrementados uma única vez. Responda:

a) Quais serão os valores de **`contador_a`** e **`contador_b`** ao final da quinta execução da função?
b) Explique por que os dois contadores apresentam comportamentos diferentes, mesmo sendo declarados no interior da mesma função e inicializados com o mesmo valor.
c) Em quais regiões de memória esses objetos normalmente seriam armazenados em um programa compilado para Linux?
d) O que aconteceria se outra função tentasse acessar diretamente qualquer um desses contadores pelo nome? Justifique.

Questão 15:

Na função **`main()`**, a memória alocada dinamicamente para o vetor **`medicoes`** é liberada antes da chamada de **`exibir_relatorio()`**. Explique por que essa operação não compromete a apresentação do relatório, considerando as diferenças entre os objetos apontados por **`medicoes`** e os dados armazenados na variável **`resultado`**.

Questão 16:

Considere os objetos **`medicoes`** e **`resultado`**, declarados no interior da função **`main()`**. Explique em quais regiões de memória esses objetos normalmente são armazenados. Em seguida, explique por que a memória utilizada para armazenar as temperaturas não é necessariamente a mesma região de memória utilizada para armazenar o ponteiro **`medicoes`**. Finalmente, explique por que o programa consegue utilizar a estrutura **`resultado`** após liberar a memória alocada dinamicamente para o vetor de temperaturas.

Questão 17:

Considere as declarações abaixo, realizadas na função **`analisar_temperaturas()`**:

```
static int total_analises = 0;
resultado_t resultado = {0};
```

Explique por que esses objetos normalmente são armazenados em regiões diferentes da memória, embora ambos possuam inicializadores com valor zero.

Questão 18:

Considere as seguintes variáveis utilizadas no programa: **`total_alertas`**, **`total_analises`** e **`resultado`**. Todas elas podem ser utilizadas durante a execução da função **`analisar_temperaturas()`**. Explique as diferenças entre essas três variáveis considerando seu escopo, sua duração de armazenamento e sua localização típica na memória.

Em seguida, explique por que a declaração **`static int total_analises = 0;`** não faz com que a variável seja reinicializada com zero a cada chamada da função.

Boa prova! Não esqueça de devolver o papel almaço para o Prof. Abrantes Araújo Silva Filho. Não esqueça de escrever também:

- Seu nome completo
- Sua turma
- Sua matrícula
- A data de hoje

Dúvidas a respeito desta prova serão esclarecidas pelo professor, durante as próximas aulas.