1.6. Exercícios resolvidos

Nesta seção teremos a especificação de diversos problemas. Para cada um deles iremos escrever um pseudo-código que resolva o problema descrito, utilizando o recurso de Registros. Em seguida, implementar um programa em C.

1.6.1. Programa do cálculo de médias de alunos

Escrever um programa que cadastre o nome, a matrícula e duas notas de vários alunos. Em seguida imprima a matrícula, o nome e a média de cada um deles.

Pseudo-código do programa.

```
REGISTRO Aluno
  matricula: NUMÉRICO
  nome: TEXTO
  nota1, nota2: NUMÉRICO
FIM_REGISTRO

QUANTIDADE_DE_ALUNOS = 3
DECLARA alunos: Aluno[QUANTIDADE_DE_ALUNOS]

PARA i=0 ATÉ QUANTIDADE_DE_ALUNOS FAÇA
  LEIA alunos[i].nome
  LEIA alunos[i].matricula
  LEIA alunos[i].nota1
  LEIA alunos[i].nota2
FIM_PARA

PARA i=0 ATÉ QUANTIDADE_DE_ALUNOS FAÇA
  ESCREVA alunos[i].matricula
  ESCREVA alunos[i].nome
  ESCREVA (alunos[i].nota1 + alunos[i].nota2)/2 ❶
FIM_PARA
```

❶ Imprime a média calculada.

Programa em C: code/registros/calculo_das_medias.c.

```
#include <stdio.h>

typedef struct {
        int matricula;
        char nome[100];
        float nota1;
        float nota2;
} Aluno;


#define QUANTIDADE_DE_ALUNOS 3

int main(){
        Aluno alunos[QUANTIDADE_DE_ALUNOS];

        printf("Dados: nome(sem espacos), matricula, nota1, nota2\n");
        for(int i=0; (i < QUANTIDADE_DE_ALUNOS); i++){
                printf("\nInforme os dados do aluno(%i): ",i+1);
                scanf("%s %i %f %f",alunos[i].nome, &alunos[i].matricula,
                        &alunos[i].nota1, &alunos[i].nota2);
        }

        printf("\nMatricula\tNome\tMedia\n");
        for(int i=0; (i < QUANTIDADE_DE_ALUNOS); i++){
                printf("%i\t%s\t%1.2f\n",alunos[i].matricula,alunos[i].nome,
                        (alunos[i].nota1 + alunos[i].nota2)/2);
        }

        getchar();
```

```
        return 0;
}
```

Resultado ao simular a execução do programa.

```
Dados do aluno: nome(sem espacos), matricula, nota1, nota2

Informe os dados do aluno(1): Jesuíno 2887399 6.0 7.5
Informe os dados do aluno(2): Maria 2887398 7.0 9.0
Informe os dados do aluno(3): Virgulino 2887400 10.0 8.0
Matricula       Nome    Media
2887399 Jesuíno        6.75
2887398 Maria   8.00
2887400 Virgulino       9.00
```

1.6.2. Problema do cálculo e consulta do IMC de uma pessoa

Escrever um programa que cadastre o nome, a altura, o peso, o cpf e sexo de algumas pessoas. Com os dados cadastrados, em seguida localizar uma pessoas através do seu CPF e imprimir o seu IMC.

Pseudo-código do programa.

```
REGISTRO Pessoa
  nome, sexo: TEXTO
  peso, altura, cpf: NUMÉRICO
FIM_REGISTRO

QUANTIDADE_DE_PESSOAS = 3

PARA i=0 ATÉ QUANTIDADE_DE_PESSOAS FAÇA
  LEIA pessoas[i].nome
  LEIA pessoas[i].altura
  LEIA pessoas[i].peso
  LEIA pessoas[i].cpf
  LEIA pessoas[i].sexo
FIM-PARA

DECLARA cpf_localizador: NUMÉRICO
LEIA cpf_localizador ❶

PARA i=0 ATÉ QUANTIDADE_DE_PESSOAS FAÇA      ❷
  SE pessoas[i].cpf == cpf_localizador ENTÃO ❸
    ESCREVE pessoas[i].nome
    ESCREVE pessoas[i].sexo
    // IMC = peso / (altura * altura)
    ESCREVE pessoas[i].peso / (pessoas[i].altura * pessoas[i].altura)
FIM-PARA
```

❶ O ler o campo **identificador** de Pessoa (CPF).

❷❸ Pesquisa pelo registro Pessoa identificado pelo CPF lido.

Programa em C: code/registros/imc_calculo.c.

```
#include <stdio.h>

typedef struct{
        char  nome[100];
        char  sexo; // 'm': masculino, 'f': femino
        float peso;
        float altura;
        long long cpf;
} Pessoa;

#define QUANTIDADE_DE_PESSOAS 3

int main(){
        Pessoa pessoas[QUANTIDADE_DE_PESSOAS];
```

```
        printf("Campos: nome, altura, peso, cpf, sexo\n");
        for(int i=0; (i < QUANTIDADE_DE_PESSOAS); i++){
                printf("\nInforme os dados da pessoa(%i): ",i+1);
                scanf("%s %f %f %Lu %c",pessoas[i].nome, &pessoas[i].altura,
                        &pessoas[i].peso, &pessoas[i].cpf, &pessoas[i].sexo);
        }

        printf("\nInforme o CPF da pessoa: ");
        long long cpf_localizador;
        scanf("%Lu",&cpf_localizador); // ❶

        printf("\nSexo\tNome\tIMC");
        for(int i=0; (i < QUANTIDADE_DE_PESSOAS); i++){ //❷
                if (cpf_localizador == pessoas[i].cpf){ //❸
                        float imc = pessoas[i].peso / (pessoas[i].altura *
                                pessoas[i].altura);
                        printf("\n%c\t%s\t%1.2f\n",pessoas[i].sexo,
                                pessoas[i].nome, imc);
                        break;
                }
        }

        getchar();
        return 0;
}
```

❶ O ler o campo **identificador** de Pessoa (`cpf`).

❷❸ Pesquisa pelo registro Pessoa identificado pelo CPF lido.

Resultado ao simular a execução do programa.

```
Campos: nome, altura, peso, cpf, sexo

Informe os dados da pessoa(1): Jesuíno 1.82 79 48755891748 m
Informe os dados da pessoa(2): Maria 1.66 52 72779162201 f
Informe os dados da pessoa(3): Virgulino 1.75 80 71443626406 m
Informe o CPF da pessoa: 72779162201
Sexo    Nome    IMC
f       Maria   18.87
```

1.6.3. Problema de pontos no plano cartesiano

Escrever um programa que leia 5 pontos. Em seguida imprima qual o ponto mais próximo do primeiro ponto lido.

Pseudo-código do programa.

```
REGISTRO Ponto
  x, y: NUMÉRICO
FIM_REGISTRO

QUANTIDADE_DE_PONTOS = 5

PARA i=0 ATÉ QUANTIDADE_DE_PONTOS FAÇA
  LEIA p[i].x
  LEIA p[i].y
FIM_PARA

menor_distancia_ao_quadrado = MAIOR_INTEIRO ❶
ponto_mais_proximo = 1 ❷

PARA i=1 ATÉ QUANTIDADE_DE_PONTOS FAÇA
  distancia_ao_quadrado = (pontos[i].x-pontos[0].x)*
    (pontos[i].x-pontos[0].x)+(pontos[i].y-pontos[0].y)*
    (pontos[i].y-pontos[0].y) ❸
  SE distancia_ao_quadrado < menor_distancia_ao_quadrado ENTÃO ❹
    ponto_mais_proximo = i ❺
    menor_distancia_ao_quadrado = distancia_ao_quadrado ❻
FIM_PARA
```

```
ESCREVA p[ponto_mais_proximo].x,p[ponto_mais_proximo].y
```

❶❹❻ `MAIOR_INTEIRO` representa o maior número inteiro que podemos armazenar numa variável. Geralmente atribuimos **o maior** inteiro quando procuramos por **um menor** valor. No código, comparamos `menor_distancia_ao_quadrado` com `distancia_ao_quadrado` e salvamos o **menor** deles. Se executarmos isso sucessivamente, ao final, `menor_distancia_ao_quadrado` conterá o **menor** valor comparado.[4]

❷❺ Esta variável irá guardar a posição do ponto mais próximo. Ela é atualizada, sempre que encontramos outro ponto com menor distância.

❸ Calculo para encontrar a distância entre dois pontos. Na realizadade, a distância entre os dois pontos seria a raiz de `distancia_ao_quadrado`. Mas não há diferença em comparar a distância ao quadrado. Sabemos, por exemplo, que a **raiz** de `x` é **menor do que a raiz** de `y` se `x` for **menor** do que `y`.

Programa em C: code/registros/ponto_proximo.c.

```
#include <stdio.h>
#include <limits.h> // contém definição de INT_MAX

typedef struct{
        int x;
        int y;
} Ponto;

#define QUANTIDADE_DE_PONTOS 5

int main(){
        Ponto pontos[QUANTIDADE_DE_PONTOS];

        printf("Campos: x, y\n");
        for(int i=0; (i < QUANTIDADE_DE_PONTOS); i++){
                printf("\nInforme as coordenadas do ponto(%i): ",i+1);
                scanf("%d %d",&pontos[i].x,&pontos[i].y);
        }

        int menor_distancia_ao_quadrado = INT_MAX; // maior inteiro
        int ponto_mais_proximo = 1;

        for(int i=1; (i < QUANTIDADE_DE_PONTOS); i++){
                int distancia_ao_quadrado = (pontos[i].x-pontos[0].x)*
                        (pontos[i].x-pontos[0].x)+(pontos[i].y-pontos[0].y)*
                        (pontos[i].y-pontos[0].y);
                if(distancia_ao_quadrado < menor_distancia_ao_quadrado){
                        ponto_mais_proximo = i;
                        menor_distancia_ao_quadrado = distancia_ao_quadrado;
                }
        }

        printf("\nPonto mais proximo: (%d,%d)\n",
                pontos[ponto_mais_proximo].x, pontos[ponto_mais_proximo].y);

        getchar();
        return 0;
}
```

Resultado ao simular a execução do programa.

```
Campos: x, y

Informe as coordenadas do ponto(1): 0 0
Informe as coordenadas do ponto(2): 4 6
Informe as coordenadas do ponto(3): 6 1
Informe as coordenadas do ponto(4): 5 3
Informe as coordenadas do ponto(5): 7 2
Ponto mais proximo: (5,3)
```

1.6.4. Problema sobre cadastro de produtos no supermercado

Escrever um programa que cadastre vários produtos. Em seguida, imprima uma lista com o código e nome da cada produto. Por último, consulte o preço de um produto através de seu código.

Pseudo-código do programa.

```
REGISTRO Produto
  codigo: NUMÉRICO
  nome: TEXTUAL
  preco: NUMÉRICO
FIM_REGISTRO

QUANTIDADE_DE_PRODUTOS = 5
DECLARA produtos: Produto[QUANTIDADE_DE_PRODUTOS]

PARA i=0 ATÉ QUANTIDADE_DE_PRODUTOS FAÇA
  LEIA produtos[i].codigo
  LEIA produtos[i].nome
  LEIA produtos[i].preco
FIM_PARA

PARA i=0 ATÉ QUANTIDADE_DE_PRODUTOS FAÇA
  ESCREVA produtos[i].codigo
  ESCREVA produtos[i].nome
FIM_PARA

DECLARA codigo_digitado: NUMÉRICO
LEIA codigo_digitado

PARA i=0 ATÉ QUANTIDADE_DE_PRODUTOS FAÇA
  SE produtos[i].codigo == codigo_digitado ENTÃO
    ESCREVA produtos[i].preco
FIM_PARA
```

Programa em C: code/registros/supermercado.c.

```
#include <stdio.h>

typedef struct {
        long  codigo;
        char  nome[100];
        float preco;
} Produto;

#define QUANTIDADE_DE_PRODUTOS 5

int main(){
        Produto produtos[QUANTIDADE_DE_PRODUTOS];

        printf("Campos: codigo-do-produto nome preco\n");
        for(int i=0; (i < QUANTIDADE_DE_PRODUTOS); i++){
                printf("\nInforme os dados do produto(%i): ",i+1);
                scanf("%ld %s %f",&produtos[i].codigo,produtos[i].nome,
                        &produtos[i].preco);
        }

        for(int i=0; (i < QUANTIDADE_DE_PRODUTOS); i++){
                printf("\n%ld\t%s R$ %1.2f", produtos[i].codigo,
                        produtos[i].nome,produtos[i].preco);
        }

        long codigo_digitado;
        printf("\nInforme o codigo do produto: ");
        scanf("%ld", &codigo_digitado);

        for(int i=1; (i < QUANTIDADE_DE_PRODUTOS); i++){
                if (produtos[i].codigo == codigo_digitado) {
                        printf("\nPreço: R$ %1.2f\n", produtos[i].preco);
                }
        }
```

```
        getchar();
        return 0;
}
```

Resultado ao simular a execução do programa.

```
Campos: codigo-do-produto nome preco

Informe os dados do produto(1): 1 laranja 1.4
Informe os dados do produto(2): 2 rosquinha 3
Informe os dados do produto(3): 3 leite-moca 4.5
Informe os dados do produto(4): 4 farinha-de-trigo 2.7
Informe os dados do produto(5): 5 coxinha 1.5
1       laranja R$ 1.40
2       rosquinha R$ 3.00
3       leite-moca R$ 4.50
4       farinha-de-trigo R$ 2.70
5       coxinha R$ 1.50
Informe o codigo do produto: 4
Preço: R$ 2.70
```

1.6.5. Problema sobre gerenciamento de contas bancárias

Escreva um programa que simule contas bancárias, com as seguintes especificações:

- Ao iniciar o programa vamos criar contas bancárias para três clientes.
  - Cada conta terá o nome e o CPF do cliente associado a ela.
  - No ato da criação da conta o cliente precisará fazer um depósito inicial.
- Após as contas serem criadas, o sistema deverá possibilitar realizações de saques ou depósitos nas contas.
  - Sempre que uma operação de saque ou depósito seja realizada, o sistema deverá imprimir o nome do titular e o saldo final da conta.

Pseudo-código do programa.

```
REGISTRO Conta
  numero_da_conta, cpf_do_cliente, saldo: NUMÉRICO
FIM_REGISTRO

REGISTRO Cliente
  cpf: NUMÉRICO
  nome: TEXTUAL
FIM_REGISTRO

QUANTIDADE_DE_CLIENTES = 3
DECLARA clientes: Cliente[QUANTIDADE_DE_CLIENTES]
DECLARA contas:   Conta[QUANTIDADE_DE_CLIENTES]

PARA i=0 ATÉ QUANTIDADE_DE_CLIENTES FAÇA
  LEIA clientes[i].cpf
  LEIA clientes[i].nome
  LEIA contas[i].saldo // depósito inicial

  clientes[i].codigo = i
  contas[i].numero_da_conta = i
  contas[i].codigo_do_cliente = clientes[i].codigo
FIM_PARA

DECLARA operacao: TEXTUAL
DECLARA num_conta, valor, sair=0: NUMÉRICO

ENQUANTO sair == 0 FAÇA
  LEIA operacao

  SE operacao == "saque" OU operacao == "deposito" ENTÃO
    LEIA num_conta, valor
```

```
    PARA i=0 ATÉ QUANTIDADE_DE_CLIENTES FAÇA
      SE contas[i].numero_da_conta == num_conta ENTÃO
        SE operacao == "saque" ENTÃO
          contas[i].saldo = contas[i].saldo - valor
        SE operacao == "deposito" ENTÃO
          contas[i].saldo = contas[i].saldo + valor
        PARA j=0 ATÉ QUANTIDADE_DE_CLIENTES FAÇA
          SE clientes[j].codigo == contas[i].codigo_do_cliente ENTÃO
            ESCREVE clientes[j].nome, contas[i].saldo
        FIM_PARA
    FIM_PARA
  SENÃO operacao == "sair" ENTÃO
    sair = 1
FIM_ENQUANTO
```

Programa em C: code/registros/conta_bancaria.c.

```
#include <stdio.h>

typedef struct  {
        char nome[256];
        long long cpf;
} Cliente;

typedef struct {
        long    numero_da_conta;
        long    cpf_do_cliente;
        double  saldo;
} Conta;

#define QUANTIDADE_DE_CLIENTES 3
#define OPERACAO_SAQUE 1
#define OPERACAO_DEPOSITO 2

int main(){
  Cliente clientes[QUANTIDADE_DE_CLIENTES];
  Conta   contas[QUANTIDADE_DE_CLIENTES];

  printf("Campos: cpf nome deposito-inicial\n");
  for(long i=0; (i < QUANTIDADE_DE_CLIENTES); i++){
    printf("\nDados para abertura da conta(%ld): ",i+1);
    scanf("%Ld %s %lf",&clientes[i].cpf,clientes[i].nome,
      &contas[i].saldo);

    contas[i].numero_da_conta = i;
    contas[i].cpf_do_cliente = clientes[i].cpf;

    printf("\nCliente: %s Conta: %ld Saldo inicial: %1.2lf\n",
      clientes[i].nome, contas[i].numero_da_conta, contas[i].saldo);
  }

  int operacao; // como ainda não aprendemos a comparar strings,
                // vamos usar 'operação' como numérico.
  long num_conta;
  double valor;
  int sair=0; // FALSE

  while (!sair){
    printf("\nInforme a operação: 1-Saque 2-Deposito 3-Sair: ");
    scanf("%d", &operacao);

    if (operacao == OPERACAO_SAQUE || operacao == OPERACAO_DEPOSITO){
      printf("\nInforme numero-da-conta e valor: ");
      scanf("%ld %lf", &num_conta, &valor);
      for(int i=0; (i < QUANTIDADE_DE_CLIENTES); i++){
        if (contas[i].numero_da_conta == num_conta) {
          if (operacao == OPERACAO_SAQUE){
            contas[i].saldo -= valor;
            printf("\nSAQUE: %1.2lf", valor);
          }
          if (operacao == OPERACAO_DEPOSITO){
            contas[i].saldo += valor;
            printf("\nDEPOSITO: %1.2lf", valor);
```

```
            }
            for(int j=0; j < QUANTIDADE_DE_CLIENTES; j++){
              if (clientes[j].cpf == contas[i].cpf_do_cliente)
                printf("\nCliente: %s Saldo atual: %1.2lf",
                  clientes[j].nome, contas[i].saldo);
            }
          }
        }
      }else{
        sair = 1; // TRUE
      }
    }

    getchar();
    return 0;
}
```

Resultado ao simular a execução do programa.

```
Campos: cpf nome deposito-inicial

Dados para abertura da conta(1): 48755891748 Jesuíno 1500
Cliente: Jesuíno Conta: 0 Saldo inicial: 1500.00

Dados para abertura da conta(2): 72779162201 Maria 200
Cliente: Maria Conta: 1 Saldo inicial: 200.00

Dados para abertura da conta(3): 71443626406 Virgulino 600
Cliente: Virgulino Conta: 2 Saldo inicial: 600.00

Informe a operação: 1-Saque 2-Deposito 3-Sair: 1
Informe numero-da-conta e valor: 0 300
SAQUE: 300.00
Cliente: Jesuíno Saldo atual: 1200.00
Informe a operação: 1-Saque 2-Deposito 3-Sair: 2
Informe numero-da-conta e valor: 2 400
DEPOSITO: 400.00
Cliente: Virgulino Saldo atual: 1000.00
Informe a operação: 1-Saque 2-Deposito 3-Sair: 3
```

Após todos estes programas, agora vamos ver uma técnica que não utilizada ainda, a inicialização de *registro* com valores pré-definidos.

---

[4] Caso tivéssemos inicializado a variável `menor_distancia_ao_quadrado` com `0`, ao compará-lo com outro número, ele seria o **menor**, impossibilitando encontrar a **menor** distância.