ESCOLA SUPERIOR DE TECNOLOGIA IPCA

3 DE JANEIRO DE 2019

RELATÓRIO DE TRABALHO PRÁTICO AED1/LP1

DANIEL TORRES
A17442@ALUNOS.IPCA.PT

Resumo

Este trabalho foi realizado tentando aprimorar os meus conhecimentos a nível de estrutura e organização de código em C.

Em todas as questões foi tentado otimizar ao máximo o código de forma a não utilizar recursos desnecessários tornando por vezes o código mais complexo de leitura e interpretação, na medida que se torna imprescindível fazer a separação do código pelas funções.

A maior dificuldade passada durante a realização do trabalho foi a validação dos dados de input do utilizador, pois considero um ponto critico de qualquer aplicação, pois através de más leituras podem haver injeções de SQL, buffer overflows, desta forma foi tentado manter a leitura de dados em funções tais como lerInteiro(), lerString(), pois é mais simples futuramente fazer a limpeza das variáveis e validação das mesmas.

Toda a solução foi desenvolvida utilizando o NetBeans utilizando o compilador GCC de Linux e de Windows (cygwin).

Durante a execução do trabalho foi sendo realizado um ficheiro readme.md no qual fui documentando e mostrando a abordagem realizada nas questões, esse trabalho foi aproveitado para a realização deste relatório na qual foi transcrito quase na integralidade. O mesmo ficheiro é de extrema utilidade para a documentação do código que optei por desenvolver utilizando o GIT para manter as alterações de código e tentar manter um gênero de portfolio de códigos realizados, pois através do código realizado conseguimos ter uma maior visibilidade no mercado.

Autor: Daniel Filipe a17442@alunos.ipca.pt

Índice

Estrutura de Ficheiros

- **Código Principal**
  - /questao**XX**/main.c **(Questão XX)**
  - /questao**XX**/questao**XX**.h **(Header File da Questão XX)**
- **Código Auxiliar**
  - /questao**XX**/funcoes**XX**.c (Funções utilizadas na função XX)
- **Documentação Automática**
  - /doc/índex.html **(**Documentação gerada com o doxygen**)**
- **Relatório**
  - /relatorio/

Instruções de decisão

Questão 01

Trata-se de uma aplicação básica que não devendo utilizar ciclos se torna numa aplicação não muito interessante a nível de se ter que repetir o mesmo código várias vezes, e sempre que for necessário alterar algum parâmetro no mesmo temos de alterar em simultâneo em cinco sítios.

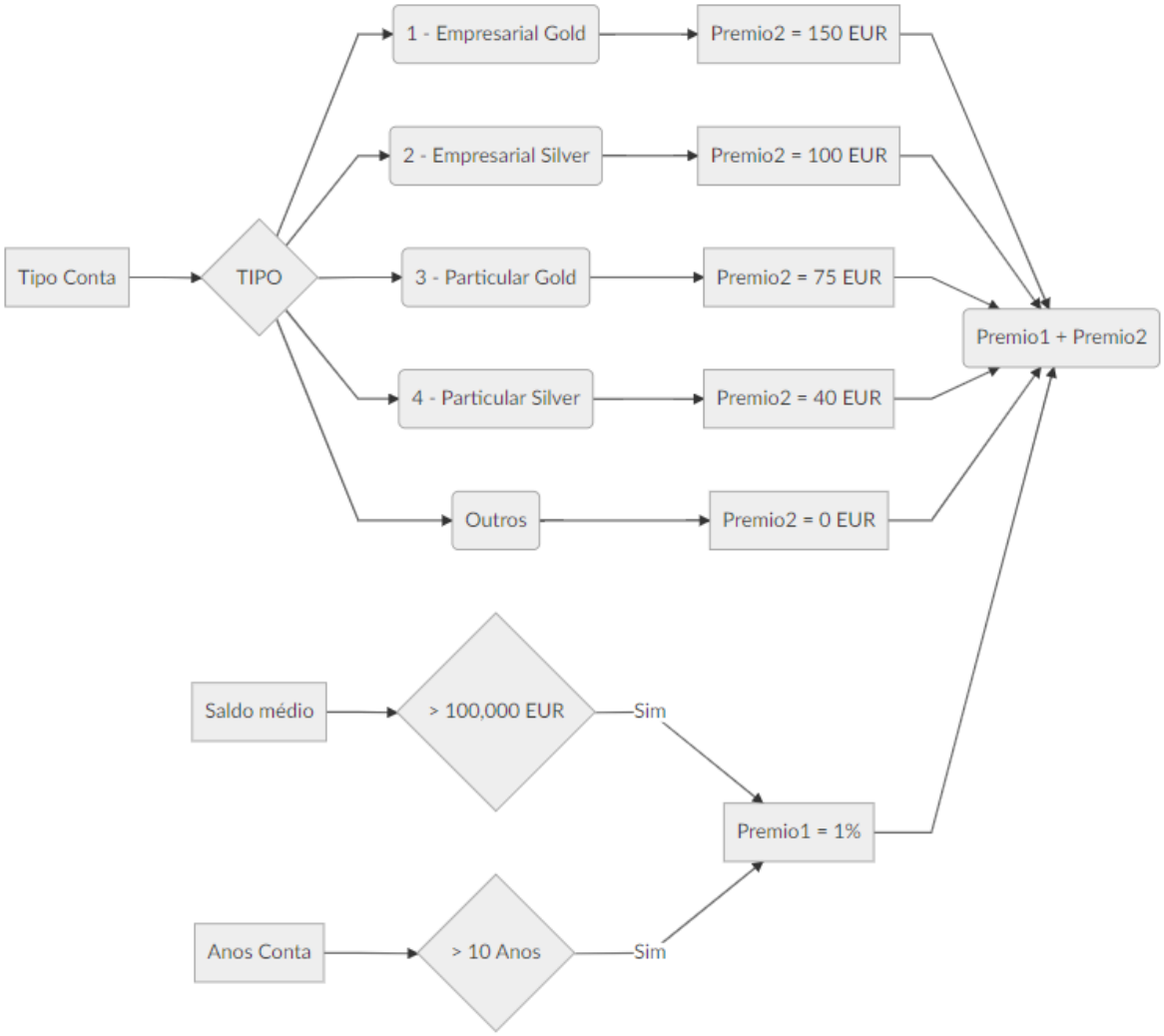
Foi utilizada recursividade na função que lê a idade para fazer uma validação muito básica de forma a não utilizar ciclos.

Com esta aplicação temos a clara noção da vantagem dos ciclos em programação, pelo que nos leva a equacionar em não utilizar este tipo de abordagem em futuras implementações do género.

Questão 02

Esta questão trata-se de uma abordagem um pouco mais complexa à utilização de instruções condicionais, dada a complexidade do enunciado. São pedidos 3 dados ao utilizador e mediante as condições que nos são indicadas no enunciado são calculados prémios a atribuir a uma dada conta bancária.

Podemos com esta questão tirar a conclusão de que devemos esquematizar as condições lógicas de forma a que se tornem de simples compreensão, comentando claramente as condições lógicas utilizadas de forma a que seja facilmente modificável.



De forma a simplificar futuras alterações na solução optei por utilizar um array global onde estão definidos os prémio para cada tipo de conta e a sua descrição, pois desta forma pode ser adicionada a qualquer momento uma nova conta sem necessidade de alteração de qualquer código.

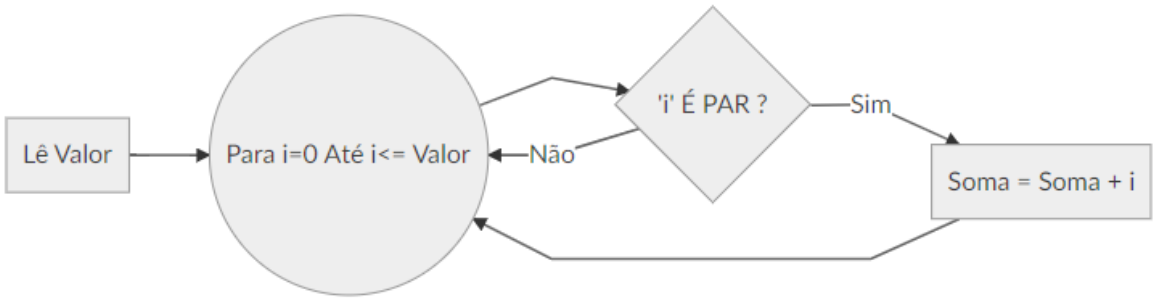
```
const char* descricao_conta[5] = {
    "Empresarial Gold",   // Descrição para conta tipo 1
    "Empresarial Silver", // Descrição para conta tipo 2
    "Particular Gold",    // Descrição para conta tipo 3
    "Particular Silver",  // Descrição para conta tipo 4
    "Outros"              // Descrição para conta tipo 5
};
const int premio_conta[5] = {
    150,  // Bonus para conta tipo 1
    100,  // Bonus para conta tipo 2
    75,   // Bonus para conta tipo 3
    40,   // Bonus para conta tipo 4
    0     // Bonus para conta tipo 5
};
const int numero_contas = sizeof(descricao_conta) / sizeof(char*);
....
/**
 * Mostra as opções da constante descricao_conta
 * Foi utilizada recursividade para evitar o uso de ciclos
 */
void mostra_opcoes(int n) {
    printf("%d \t-\t %s\n",n,descricao_conta[n-1]);

    if ((n-1) != 0) mostra_opcoes(n-1);
}
```

Instruções de Repetição

Questão 03

Foi criada uma função que valida se um número é par e ao mesmo tempo para validar se um número é ímpar apesar de a mesma não ser necessária nesta solução. Não sendo nativo do C as variáveis booleanas, optei por utilizar a função a retornar um inteiro considerando que o valor retornado é o valor booleano da operação ((numero % 2) == 0).

Para fazer o somatório dos números pares de 0..[n] optei por utilizar um ciclo for pois torna-se um ciclo mais simples de implementar quando conhecemos o inicio e fim do ciclo.



Questão 04

Pessoalmente esta questão suscitou um desafio interessante, foi uma forma muito útil de perceber a importância de operações básicas de matemática e as conseguir aplicar a situações reais.

A parte mais interessante nesta solução foi a realização da função inverte que inverte qualquer número inteiro, para tal passo a explicar de forma suscita a maneira como foi implementado.

Após a criação da função inverte apenas é necessário efetuar dois ciclos um dentro do outro de forma a iterar todas as possibilidades de multiplicação entre a seguinte formula:

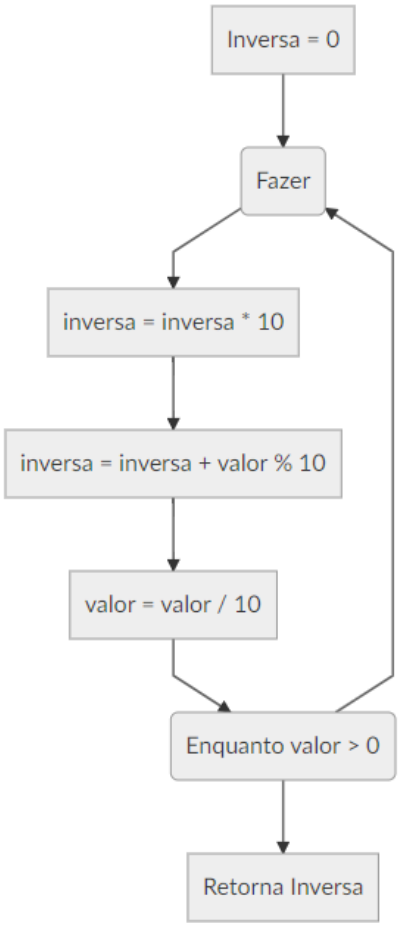
multiplicação = **primeiro** * **segundo**

Se o valor for igual à sua inversa logo é capicua.

Foi também muito interessante otimizar e ver a forma melhor de reduzir cálculos desnecessários tais como:

- 999**998 e em seguida fazer 998*999*
- 500*400 e todos os seguintes quando a maior capicua já é um valor superior a esta multiplicação.
  Isto mostra uma forma muito interessante de usar a função break em ciclos.

Função inverte:




```c
/*!
 * Função para inverter um numero inteiro
 * @function inverte
 * @param valor
 * @return valor inverso
 */
int inverte(int valor)
{
   int inversa=0;
   do
   {
      inversa   = inversa * 10 + (valor%10);
      valor     = valor / 10;
   } while( valor>0 );
   return (inversa);
}
```

A função inverte está descrita acima:

Questão 05

Foi desenvolvida uma solução que lê valores superiores ou iguais a zero usando um ciclo (do---while), de forma a que guarda numa variável **[contador_condicao]** uma incrementação dos valores dentro da condição esperada.

Questão 06

Foi utilizada uma abordagem comum. Uma vez que não podiam ser utilizadas estruturas foi utilizada uma variável "runtime" de modo a armazenar os vários resultados necessários entre chamadas de funções.

```
// runtime[0] ==> Numero de estudantes
// runtime[1] ==> Masculinos
// runtime[2] ==> Femininos
// runtime[3] ==> Numero de estudantes M sub23 insatisfeitos
int runtime[4] = {0 ,0 ,0 ,0};
```

Questão 07

Função is_primo:

Foi implementada a função is_primo tentando usar uma forma mista do ***Crivo de Eratóstenes*** com a forma mais vulgar de mediante um ciclo até à raiz quadrada do número a testar.

Utilizando o ***Crivo de Eratóstenes*** logo numa primeira instância aceitamos os primos que conhecemos

```
    if (n==2)
           return true;
```

E descartamos todos os múltiplos dele próprio

```
   // Todos os pares não são primos

  if (n % 2)
      return false;
```

É efetuada a mesma operação para todos os seguintes até ao número 19:

```
        // Próximos números primos
        if (n==3 || n==5 || n==7 || n==11 || n==13 || n== 17 || n==19)
              return true;
        // Verifica o resto da divisao por um deles possivel
        if (n%3==0 || n%5==0 || n%7==0 || n%11==0 || n%13==0 || n%17==0
|| n%19==0 )
             return false;
```

Após esta implementação apenas será necessário testar um numero reduzido de hipóteses.

```
    // Caso nao seja um caso destes itera
    // Para i=19 ate i<=sqrt(n)
    int tocheck = sqrt(n);//n / 2; // se n>n/2 então n % n > 0
    for (int i = 19; i < tocheck; i = i + 2) { // Os pares já foram todos
verificados
        // Verifica se é divisível por i
        if (n % i ==0)
            return false;
    }
```

Conclusão: Sobre a função is_prime conseguiu-se obter os números primos até 1,000,000 em cerca de ***1 Segundo*** poupando assim iterações desnecessárias

Função gera_sequencia:

Esta função tem por objetivo gerar um array de tamanho variável definido num parâmetro cujos números estão cumpridos entre o parâmetro min e max.

```
/**
 * Função que gera uma quantidade de números aleatórios definida nos parâ
metros
 * @param min numero mínimo a gerar
 * @param max numero máximo a gerar
 * @param numeros quantidade de números
 * @return pointer do array de números
 */
int * gera_sequencia(int min, int max, int numeros) {
    int *numero_aleatorio;

    numero_aleatorio = (int*) calloc(numeros, sizeof(int));

    srand((unsigned) time(NULL));
    for (int i=0; i < numeros ;i++)
        numero_aleatorio[i] = rand() % max + min;
    return numero_aleatorio;
}
```

É utilizado a função rand() que necessita de um seed para ser pseudo-random que é chamado pela função srand().

A função faz uso de de apontadores de endereço de memoria de forma a que seja passada no return o apontador do endereço.

Para evocar a função pode ser da seguinte forma:

```
    int *numeros_aleatorios;
   // Chama função que gera a quantidade necessaria.
   numeros_aleatorios = gera_sequencia(NUMERO_MIN, NUMERO_MAX, NUMEROS);
```

Estrutura do Main:

Utilizando as duas funções acima descritas o main encarrega-se então agora de iterar o array definido de forma a determinar quais os números primos gerados no array, fazer o somatório dos mesmos e a contagem.

Funções e Procedimentos

Questão 08

O pedido neste enunciado é a melhoria da questão 3 de forma a utilizar uma função:

```
/**
 * Função somatório questão 08 usando uma subrotina
 * @param numero Numero processar
 * @return somatorio dos pares
 */
unsigned int somatorio(int numero) {
    unsigned int sum=0;
    for (int i=1 ;i<=numero ;i++) {
        if (par(i))
            sum += i;
    }
    return sum;
}
```

Verificamos que a leitura do código presente no main torna-se bastante mais legível e simples, pelo que devemos sempre que utilizar funções para tarefas que se tratam de funções especificas.

```
    printf("Soma dos pares: %u - ",somatorio(n));
```

Foi utilizado um **unsigned int** pois é um inteiro sem sinal logo pela especificidade da função pedida apenas iriam ser somados os números positivos ganhando assim.

Questão 09

O pedido desta questão é a implementação do somatório da questão anterior desta vez, recursivamente.

```
/**
 * Função somatório questão 09
 * @param numero Numero processar
 * @return somatorio dos pares
 */
unsigned int somatorio_recursivo(int numero) {
    // Se chegar a zero ou numero for negativo retorna 0 e sai da recursi
vidade
    if (numero <= 0) return(0);

    // Se for par soma o numero à proxima recursividade senao soma 0
    return ( ( par(numero) ? numero : 0 )  + somatorio_recursivo(numero -
1) );
}
```

Questão 10

Foi reimplementada a questão 5 utilizando uma função que recebe o valor mínimo de peso a contabilizar no contador, máximo e valor de paragem.

```
/**
 * Função que lê um valor inserido por scanf e devolve o
 * mesmo caso esteja no intervalo definido
 * @param minimo minimo para o valor
 * @param maximo maximo para o valor
 * @param paragem valor de paragem
 * @return retorna o valor inserido caso esteja na condição
 */
int lerDados(int minimo, int maximo, int paragem) {
    int userinput = 0;

    if (maximo < minimo) return 0;

    scanf("%d",&userinput);
    if (userinput == paragem)
        return paragem;
    else if (userinput >= minimo && userinput <= maximo )
        return userinput;
    else
        return 0;
}
```

Questão 11

O pedido é a implementação do algoritmo criado na questão 7, mas utilizando uma função para verificar se o número é primo.

Essa abordagem já foi tida em conta na questão 7 pelo que apenas foi compilado o executável questao10 usando toda a estrutura da questão 7

```
gcc -o ../bin/questao10 ../questao07/main.c ../questao07/funcoes07.c -O3
-g3  -W -Wall -Wextra -Wuninitialized -Wstrict-aliasing
```

Arrays

Questão 12

Foi feita uma abordagem ao problema o mais abstrata possível de forma a que futuras alterações no algoritmo possam a surgir sem ter que se refazer todo o código.

De forma a simplificar a leitura de código foi utilizada um enum para identificar qual o apontador no array que pertence a cada tipo de temperatura

```
enum tipo_temperatura {
    ENUM_C  = 0,
    ENUM_K  = 1,
};
```

Numa primeira fase do programa é lido um caracter K ou C conforme o que o utilizador pretenda inserir

```
    // Escala a ler
    char escala = lerChar("Insira a escala [K]elvin ou [C]elsius: "," %1[
kKcC]");
    int escala_orig = (escala == 'K' ? ENUM_K : ENUM_C);
    int escala_dest = (escala == 'K' ? ENUM_C : ENUM_K);
```

Mediante o tipo de leituras pedido pelo utilizador atribui-se à variável escala_orig o valor da escala que será inserida.

O array terá a seguinte forma:

LEITURA CELSIUS (ENUM_C = 0)

```
temperaturas[LINHA][ENUM_C] = temperaturas[LINHA][0]
```

LEITURA KELVIN (ENUM_C = 1)

```
temperaturas[LINHA][ENUM_K] = temperaturas[LINHA][1]
```

Estruturas

Questão 13

Fiz uma abordagem ao problema separando a layer de leitura de sensores da layer de dashboard, tentando fazer com que a leitura dos sensores possa ser feita externamente via uma API desenvolvida para o efeito.

- O ficheiro **main.c** dará origem ao executável **questao13-main** que fará toda a layer de dashboard.
- O ficheiro **main-sensor.c** dará origem ao executável **questao13-sensor** que fará entrada de leituras, foi pensado fazer a abordagem de leituras via TCP/IP no entanto e dada a falta de tempo foi implementada apenas pela linha de comandos pelo argumentos do programa.
  Logo para inserir uma leitura apenas será necessário:
  `./questao13-sensor 6 53`
  onde o valor **6** é o código do sensor e o valor 53 é a leitura
- O ficheiro **main-demodata.c** serve para criar dados de demo para se testar o dashboard, o mesmo dará origem ao executável **questao13-demodata**.

Estes tipos de programas são mais simples de abordar utilizando sistemas de bases de dados, pois é mais fácil manter integridade de dados, no entanto foi utilizado ficheiros binários de forma a permitir um acesso fácil e rápido. No entanto e sendo a leitura binária bastante rápida e flexível, torna-se um programa rápido sem dependências de outros sistemas de base de dados.

Na pasta bin/ existem 3 ficheiros .dat que podem ser utilizados pois tem dados dummies para testes os mesmos foram gerados pelo **questao13-demodata**.

Bibliografia

- Crivo de Eratóstenes (https://pt.wikipedia.org/wiki/Crivo_de_Eratóstenes)
- GNU coding standards (https://www.gnu.org/prep/standards/)

Utilitários Utilizados

- Mermaid Diagrams (https://github.com/knsv/mermaid)
- Markdown Editor (https://stackedit.io/)
- GIT - Repositório Privado (https://bitbucket.org/nargotik/trabalho1-lp1-aed1/)