Instituto Superior de Engenharia de Lisboa

PROGRAMAÇÃO

Relatório do Segundo Trabalho Prático

Docente

Profª. Matilde Pato

Alunos

44225 Rúben Rosário
44243 Rui Correia
43552 Samuel Costa

Novembro 2016

Índice

Introdução

Este relatório, referente ao segundo de três trabalhos práticos, foi elaborado no âmbito da unidade curricular de Programação e acompanha a realização de três exercícios propostos. Estes exercícios constituíram uma aplicação dos conteúdos cobertos pelo primeiro trabalho prático bem como de outros aspetos da linguagem Java, designadamente, Estruturas de Repetição e *Arrays*.

Ao longo do documento apresentam-se os principais objetivos metodológicos e práticos delineados pelos exercícios; a metodologia adotada; as várias etapas do raciocínio subjacente à resolução de cada problema; e, por fim, uma breve reflexão sobre o trabalho realizado em que se discutem os resultados obtidos, referindo-se a atual incompletude da resolução do exercício 3, explicitando-se, ainda assim, duas conclusões referentes ao uso de Arrays e Estruturas de Repetição na Programação.

Objetivos

O trabalho prático visa, em termos gerais, aplicar as ferramentas introduzidas em contexto de sala de aula, em particular, a aplicação de ciclos e Arrays, aprofundando as suas potencialidades de utilização, por meio de resolução de três exercícios, denominados como Mult, Kaprekar e SpireGraph.

Definem-se os seguintes objetivos:

- do ponto de vista metodológico: determinar possíveis aproximações aos exercícios a partir de pseudocódigo e algoritmo e escolher a abordagem mais adequada a cada problema;
- do ponto de vista prático, empregar e dominar os seguintes instrumentos da linguagem java: (i) respeitar a sintaxe do JAVA; (ii) conhecer características das diferentes Estruturas de Controle de Repetição `for`, `while` e `do while`, de modo a escolher a que melhor se adequa a cada passo da resolução do problema; e por último (iii) criar e manipular `Arrays` do tipo inteiro ou string.

Métodos e Recursos Utilizados

Para a realização deste trabalho foi utilizada a linguagem *Java™ Versão 8 Update 111.* Todos os elementos da linguagem necessários para a resolução dos problemas propostos foram adquiridos nas aulas da unidade curricular Programação (PG). Foram estudadas as diferentes Estruturas de Controlo e as propriedades e potencialidades de utilização de `arrays` presentes no *Java™*.

De seguida resumem-se as especificidades de cada uma das Estruturas de Controlo de repetição estudadas.

Estruturas de Controle

Uma Estrutura de Controle é um elemento de um programa que permite a repetição da execução de instruções, enquanto uma determinada condição for verdadeira.

- *while*

  A estrutura de controle `while` é a estrutura de repetição mais elementar e caracteriza-se por repetir a execução de um bloco de instruções até que uma determinada condição deixe de se verificar.

```
while (condição) {
     Instrução;
     . . .
 }
```

Listagem 1- Exemplo de aplicação da estrutura de controle de repetição "while".

- *do while*

  A estrutura de controle `do while` é uma estrutura de repetição variante do ciclo `while`, com a particularidade de a verificação da condição ser apenas realizada no final da estrutura, isto é, o bloco de instruções dentro da estrutura do `while` é executado pelo menos uma vez.

```
do (condição) {
     Instrução;
     . . .
 } while (condição);
```

Listagem 2- Exemplo de aplicação da estrutura de controle de repetição "do while".

- *for*

  A estrutura de controle de repetição `for` caracteriza-se por repetir um bloco de instruções por um número bem definido de vezes. Torna-se assim, uma estrutura bastante útil, por exemplo, para percorrer caracteres de uma `string`, assim como, linhas e/ou colunas de um `Array`, onde é conhecido previamente, neste caso, o número de caracteres da `string` ou dimensões do `Array`.

```
for ( valor_inicial; valor_final ; incremento) {
     Instrução;
     . . .
}
```

Listagem 3- Exemplo de aplicação da estrutura de controle de repetição "for".

Foi ainda aplicado o método `length` *para avaliar o argumento de uma dada dimensão de um* `array`. Verificou-se que alguns dos métodos usados para variáveis do tipo String são válidas na manipulação de `arrays` de strings, como por exemplo o método `.charAt()`.

Arrays

Em programação, os `arrays` são estruturas que armazenam vários dados do mesmo tipo (`int`, `double`, `string`, etc.) a que se pode ter acesso através de uma identificação, neste caso um índice (números inteiros) da sua posição no `array`. Os `arrays` podem ter uma ou mais dimensões (é comum denominar-se de vector para o caso de um `array` de uma dimensão, e matriz para o caso de ser bidimensional).

```
// declaração de arrays
String[] array = new int[valor]; // array de uma dimensão de strings
int[][] array = new int[valor1][valor2]; // array bidimensional de inteiros
```

Listagem 4- Exemplo de declaração de um array unidimensional e bidimensional em java.

Procurou-se que as soluções cumprissem escrupulosamente o enunciado proposto, tanto no que diz respeito às ferramentas utilizadas como ao output dos programas.

Exercícios

A presente secção deste relatório está dividida em três partes. Cada uma delas corresponde a um dos exercícios propostos. Apresenta-se a resolução na forma de Pseudocódigo, explicando-se cada passo com o detalhe adequado. Sempre que se revele importante para o entendimento de um passo específico, são mostrados trechos de código java no contexto desse passo.

Exercício 1 – *Mult*

Para este exercício foi proposta a criação de um programa que descreve a multiplicação de dois valores inteiros compreendidos no domínio [1, 32767] mostrando os valores parciais da multiplicação de cada dígito do multiplicador pelo multiplicando.
De seguida é apresentado o pseudocódigo desenvolvido para a resolução do problema.

Pseudocódigo:

1. Pedir ao utilizador dois números inteiros compreendidos entre 1 e 32767;
2. Input: Ler os valores introduzidos e guardar numa variável do tipo inteiro;
3. Realizar um output com os valores de `M` e `m` introduzidos pelo utilizador;
4. Verificar o número de dígitos do inteiro `m` recorrendo à utilização de um ciclo `do while` que realiza a multiplicação de `M` por cada dígito de `m` e guarda o valor num array, repetindo o processo até ter percorrido todos os dígitos de `m`, ou seja, enquanto a divisão inteira de `m` por 10 for maior que zero (ver Listagem 5).

```
n2aux = n2;
      int p, i = 0;
      int[] array = new int[lengthN2];
      do{
          p = n2aux%10;
          n2aux /=10;
          array[i] = p*n1;
          i++;
      }while( n2aux > 0);
```

Listagem 5 - Ciclo do while realiza a multiplicação de cada dígito do valor m com o M, guardando-o num array unidimensional.

5. Realizar o output que mostra ao utilizador os resultados parciais da multiplicação na forma: `Pi = M x mi = value_M x value_m = resultado_Pi` (ver Listagem 6);

6. Realiza o calculo e o respetivo output para demonstração que M*m é igual à soma dos parciais Pi multiplicados por um “n” que é incrementado para 10 vezes mais, de i em i (ver Listagem 6);
7. Output: mostra somatório dos parciais e o seu resultado (ver Listagem 6).

```
            //Output e calculo do total
            for (int j = 0; j < array.length; j++) {
                System.out.println("P"+j + " = M x m"+j + " = " + n1 + " x
" + array[j]/n1 + " = " + array[j]);
            }

            int n = 10, total=array[0]; // variavel total inicializada com
o P0
            System.out.print("M x n = ");
            System.out.print("P0 ");
            for (int j = 1; j < array.length; j++) {
                System.out.print("+ P"+j + " x " + n + " ");
                array[j] = array[j]*n;
                total += array[j]; // somatorio para o total
                n *= 10;
            }
            System.out.println("");

            System.out.print("M x n = ");
            for (int j = 0; j < array.length; j++) {
                if (j == array.length - 1) {
                    System.out.print(array[j] + " ");
                } else {
                    System.out.print(array[j] + " + ");
                }
            }
            System.out.print("= " + total);
```

Listagem 6 - Ciclos “for” para percorrer os índices do array, realizar multiplicação por “n” e o respectivo somatório dos parciais, assim como o output.

Exercício 2 –*Kaprekar*

Para este exercício foi proposta a criação de um programa que recebesse da linha de comando um valor inteiro com 4 dígitos e que apresenta a sucessão de acordo com a rotina de Kaprekar, até que o valor encontrado na última iteração seja igual ao valor da anterior. Como controlo de input, é exigido que o programa termine se não for indicado um valor na linha de comando, ou se o valor indicado não tenha exatamente 4 dígitos ou, por ultimo, se os dígitos forem todos iguais.

Pseudocódigo:

1. Verificar se foi introduzido um valor como argumento no método `main` (exemplo: *java Kaprekar 2016*). Se não tiver sido introduzido nenhum valor, apresenta a mensagem `"Use: java Kaprekar DDDD"`. Se tiver sido introduzido o valor como input, então realiza as seguintes verificações:
    - O valor tem 4 dígitos?
    - O valor tem 4 dígitos e são todos iguais? Se os dígitos forem todos iguais apresenta a mensagem `"Os dígitos não podem ser todos iguais"` e termina o programa.
    - Foi introduzido algum caractere não válido? Se sim, então apresenta a mensagem `"Dígito X inválido"` e termina o programa.
2. Passar a `string` dada como input, através do argumento do método `main` (`array` de `strings`), para um `array` de inteiros;
3. Inicio da rotina de Kaprekar, realizada através de um ciclo `while`:
    - As instrução do ciclo while são repetidas enquanto uma variável do tipo `boolean` for igual a `true` (`boolean kaprekar = true;`);
    - Ordenar de forma crescente os dígitos do valor introduzido, inverter-lo para ordenár de forma decrescente (ver Listagem 7);

```
 //montar valores de forma crescente
int temp;
for (int i = 0; i < vectCresc.length; i++) {
        for (int j = i+1 ; j < vectCresc.length; j++ ) {
                if (vectCresc[i] > vectCresc[j]) {
                        temp = vectCresc[i];
                        vectCresc[i] = vectCresc[j];
                        vectCresc[j] = temp;
                }
        }
}
// inverter vector crescente para montar decrescente
for (int i = 0; i < vectDec.length ; i++) {
        vectDec[vectCresc.length-i-1] = vectCresc[i];
}
```

Listagem 7 - Código para ordenar valor de forma crescente e decrescente.

    - Realizar os os outputs dos vectores ordenados de forma decrescente e crescecnte, vector A e B, respectivamente;
    - Realizar a subtração entre os vectores A e B;

- Mostrar resultado (`A-B=resultado`), guardar e comparar com o resultado da iteração anterior;
    i. Se o resultado for igual ao da iteração anterior, então a variável boolean `Kaprekar` adquire o valor de `false` e o programa termina;
    ii. Se não, ciclo while continua a repetir as iteração até o resultado da subtração de A-B convergir para o valor 6174 (ponto fixo, ou constante de kaprekar);
- Realiza o output com a forma: "`iteração : N=value_N A=value_A B=value_B A-B=resultado`";
- Actualizar valor de `N` para o resultado da subtração de `A-B`;

Exercício 3 – *SpireGraph*

Para este exercício foi proposta a elaboração de um programa que leia uma sequência de valores inteiros positivos e apresente um gráfico de pináculos. Relativamente ao input do programa dado pelo utilizador, era exigido que o programa terminasse a sua leitura quando fossem introduzidos 10 valores ou um valor negativo. O programa teria de ler uma nova sequência de valores se um dos valores introduzido fosse superior a 15, ou se a sua soma fosse superior a 30.

Pseudocódigo:

1. Pedir ao utilizador para introduzir uma sequência de valores inteiros positivos (exemplo: `System.out.print("Sequência? ");`);
2. Guardar cada valor introduzido em posições consecutivas de um `array` de inteiros até serem introduzidos 10 valores ou um valor negativo. Iniciar uma variável booleana `ask`, que tomará valor verdadeiro se o valor introduzido for maior que 15 ou a soma dos valores introduzidos for maior que 30. Inicializar uma variável `count` para determinar quantas posições válidas tem o `array`;
3. Se for introduzida uma sequência inválida, i.e., a soma dos valores for superior a 30 ou um dos valores for superior a 15, pedir ao utilizador que introduza uma nova sequência (ver Listagem 8);

```
   while(ask) {
               System.out.print("Sequência ? ");
               while ((num = keyboard.nextInt()) >= 0){ // enquanto num e' maior que
0, armazena-o no array (cond. paragem)
                   seq[count++] = num;
                   sum += num;          // adiciona a var. sum (outra cond. paragem)
      }

               for(int i=0; i< seq.length; i++){
                   if(seq[i] <= 15 && sum <= 30)
                       ask=false; // enquanto a condicao se verificar, continua neste
ciclo
                   else{
                       ask = true;         //ha entrada de nova sequencia
                       count=0;            //contador e soma voltam a zero
                       sum=0;
                       break;
                   }
               }
           }
           int[] aux = new int[count]; //array auxiliar com dimensao dos elementos
validos (max 10, sum <=30 e num <=15)
```

Listagem 8 -Código para input do utilizador.

4. Criar um `array` auxiliar com a mesma composição do `array` da sequência lida e dimensão igual ao valor de `count`;
5. Percorrer esse `array` de inteiros com os valores lidos e encontrar o máximo, por forma a saber quantas linhas terá o gráfico a apresentar;
    - Criar uma variável do tipo inteiro para guardar a altura máxima. Atribuir-lhe o valor na posição zero do `array`;
    - Comparar a variável da altura máxima com o valor na posição seguinte do `array`. Se o valor nessa posição do `array` for maior que o valor de altura máxima, atribuir a altura máxima o valor nessa posição do `array`. Repetir o processo para todas as posições do `array`;
6. Iniciar uma variável com o número de linhas, que será a altura máxima mais três, por forma a acomodar a base do gráfico e o fundo;
7. Percorrer todas as linhas e colunas do gráfico. Percorrer todas as colunas de uma linha, decidir de que tipo de linha se trata, e mostrar ao utilizador o resultado por meio de uma instrução `System.out.print()`; No final de cada linha, mudar de linha (`System.out.println();`) A variável usada para percorrer as linhas varia de um ao número de linhas, e a variável usada para percorrer as colunas varia de zero até ao valor do comprimento do `array` menos 1. Iniciar uma variável que calcula a largura de linha (valor na posição coluna do `array` de inteiros mais 3)

- Se o número da linha for igual ao número de linhas do gráfico, trata-se de uma linha de fundo. Implementar um ciclo que vá de zero até largura da linha, na segunda iteração e na penúltima, mostrar ao utilizador o caracter '|', em todas as outras, mostrar " " (espaço);

```
// base do grafico
                if (nl == nlinhas) {    //ultima linha
                    for (int j = 0; j <= larguralinha; j++) {
                        if (j == 1 || j == larguralinha - 1) System.out.print('|');
                        else System.out.print(" ");
```

Listagem 9 -Instrução de decisão para escrever a linha de fundo (última linha do gráfico).

- Se não se verificar a situação anterior, avaliar se o número da linha é menos uma unidade que a altura total do gráfico. Nesse caso trata-se de uma linha de base. Implementar um ciclo nas mesmas condições que o primeiro, mas cujo corpo contenha a instrução de mostrar na primeira iteração o caracter '<',na última '>', e em todas as outras '=';
- Se não se verificar a situação anterior, verificar se o número da linha é menor ou igual à altura total do gráfico menos o valor da sequência considerado mais dois. Nesse caso, trata-se de uma linha de topo ou de uma linha sem preenchimento. Implementar de novo um ciclo for nas mesmas condições que os anteriores. Mostrar o caracter espaço para todas as iterações exceto quando o número da linha a considerar for igual ao número de linhas menos o valor da sequência considerado mais dois, e o valor da variável contadora do ciclo for igual ao valor da sequência considerado mais um. Nesse caso mostrar '_';
- Quando não se verificar nenhum das situações anteriores, trata-se de uma linha de um dos pináculos com preenchimento. Implementar um ciclo nas mesmas condições que os anteriores.
- Se a variável contadora do ciclo for igual ao valor da sequência lida mais um, mostrar '|', o meio do pináculo. Se não for o caso, verificar a variável interna do ciclo é igual à soma da largura da linha menos o valor da sequência lida mais um com o número da linha menos a altura total do gráfico menos o valor da sequência mais dois. Nesse caso, trata-se da parede esquerda do pináculo. Se a variável contadora do ciclo for igual ao primeiro argumento da soma anterior menos o segundo, trata-se de uma parede direita do pináculo. Nesse caso, mostrar '\'. Em todos os outros casos, mostrar " " (espaço).

```
for(int j = 0; j <= larguralinha; j++){

                        if(j == seq[i] + 1) System.out.print("|");

                        else if(j == larguralinha - ((seq[i] + 1) + (nl - (nlinhas -
(seq[i] + 2)))))  System.out.print("/");

                        else if(j == larguralinha - ((seq[i] + 1) - (nl - (nlinhas -
(seq[i] + 2)))))  System.out.print("\\");

                        else System.out.print(" ");

                    }
```

Listagem 9 - Código para escrever linha com preenchimento.

Conclusões

Com a realização deste trabalho (exercícios práticos e relatório) foi possível aplicar as ferramentas introduzidas nas aulas de Programação, consolidando e aprofundando as suas potencialidades de utilização, assim como aprender a melhor estruturar o pensamento lógico para a resolução de problemas através da programação, em particular na resolução de problemas com recurso a Arrays e a Estruturas de Controlo de Repetição.

Todos os programas desenvolvidos foram testados e estão em conformidade com o enunciado do trabalho.

É possível destacar duas considerações gerais que decorreram do trabalho desenvolvido:

1) A utilização de Arrays revelou-se de extrema importância para a resolução dos problemas devido capacidade de armazenar vários dados do mesmo tipo, evitando-se assim a declaração de várias variáveis. Estes podem ser definidos com uma ou duas dimensões, tendo sido neste trabalho utilizado apenas Arrays unidimensionais.
2) Neste trabalho foi possível identificar de uma forma prática, que tipo de estruturas de controle de repetição se adequava melhor à resolução de determinado problema. Assim, sempre que era conhecido o número finito de vezes que um determinado conjunto de instruções teria de ser executado, foi utilizado um ciclo `for`. Quando não era possível determinar, à partida, o número de repetições das instruções, foi usado o ciclo `while`, ou o `do while` quando foi necessário executar as instruções pelo menor uma vez, independentemente da condição de término do ciclo.

É de salientar, que durante a resolução do exercício 3, foi iniciada uma abordagem de resolução que envolvia a utilização de um `array` bidimensional de `strings` que estaria associado às linhas de cada pináculo do gráfico a representar. No entanto, por sugestão da professora, tal abordagem foi abandonada em detrimento de uma solução que passa por utilizar instruções de saída de dados para representar o gráfico em questão.

Em suma, a utilização de estruturas de controlo em conjunto com Arrays permite a construção de programas mais dinâmicos e generalizados, i.e., não se sabendo à partida o input dado pelo utilizador, o programa tem a capacidade de se adaptar à dimensão do mesmo.

Referências

[1] Savitch W. *Java: An Introduction to Problem Solving and Programming*. 7st Ed ed: Pearson; 2015.

[2] Java Documentation - Java Platform, Standard Edition (Java SE) 8. https://docs.oracle.com/javase/8/. Accessed 29-10-2016.