Faculdade de Engenharia da Universidade do Porto

Mestrado Integrado em Engenharia Informática e Computação

PROGRAMAÇÃO

INTRODUÇÃO À PROGRAMAÇÃO EM LINGUAGEM C++

1. Tipos de dados. Expressões aritméticas. Entrada e saída.

1.1

Escreva um programa que leia do teclado uma letra e mostre o respectivo código ASCII.

1.2

Escreva um programa que leia do teclado 3 números inteiros e apresente no ecrã a sua média e a diferença de cada um dos números em relação à média. O diálogo deve aparecer no ecrã com o seguinte aspecto:

```
A ? 23
B ? 47
C ? 30
media   = 33.333
A-media = -10.333
B-media = 13.667
C-media = -3.333
```

Experimente usar diferentes tipos de variáveis para guardar os valores de **A**, **B**, **C** e **media**. Varie os valores introduzidos e o número de casas decimais usadas para mostrar os resultados. Interprete os resultados.

1.3

A massa de uma esfera é dada pela expressão $M = 4/3(\rho\pi r^3)$ em que **M**, ρ e **r** representam, respectivamente, a massa da esfera, a massa específica do material de que ela é feita e o seu raio. Escreva um programa que, dados os valores de ρ e **r**, determine o valor de **M**. Deve ser dada ao utilizador indicação das unidades em que devem ser introduzidos os dados e em que é apresentado o resultado, respectivamente, Kg/m^3, m e Kg para ρ, **r** e **M**. Use uma constante para representar o valor de π.

1.4

A solução de um sistema de equações lineares a 2 incógnitas (**x** e **y**)

$$a \cdot x + b \cdot y = c$$
$$d \cdot x + e \cdot y = f$$

é dada por

$$x = (c \cdot e - b \cdot f) / (a \cdot e - b \cdot d)$$
$$y = (a \cdot f - c \cdot d) / (a \cdot e - b \cdot d)$$

Escreva um programa que leia os valores de **a**, **b**, **c**, **d**, **e** e **f** e determine a solução do sistema de equações correspondente. Considere apenas situações em que a solução é possível e determinada.

1.5

Escreva um programa que leia dois tempos, expressos em horas, minutos e segundos e que determine a soma desses tempos. O diálogo deve aparecer no ecrã com o seguinte aspecto:

```
Tempo 1 (horas minutos segundos) ? 10 35 50
Tempo 2 (horas minutos segundos) ? 15 59 30
Soma dos tempos: 1 dia, 2 horas, 35 minutos e 20 segundos
```

1.6

A área de um triângulo pode ser determinada pela fórmula de Heron $área = \sqrt{s(s-a)(s-b)(s-c)}$ em que *s*, *a*, *b* e *c* representam, respectivamente o semi-perímetro e os comprimentos dos 3 lados. Escreva um programa que leia as coordenadas dos vértices de um triângulo e calcule a sua área, usando esta fórmula. Recorda-se que a distância entre dois pontos de coordenadas (*x1*,*y1*) e (*x2*,*y2*) é dada por $distância = \sqrt{(x2-x1)^2 + (y2-y1)^2}$.

Faculdade de Engenharia da Universidade do Porto

Mestrado Integrado em Engenharia Informática e Computação

PROGRAMAÇÃO

INTRODUÇÃO À PROGRAMAÇÃO EM LINGUAGEM C++

2. Estruturas de selecção e de repetição

2.1.

Resolver o problema 1.4 (solução de um sistema de equações lineares a 2 incógnitas), por forma a que quando o sistema for impossível ou indeterminado seja apresentada uma mensagem adequada no ecrã: "sistema impossível" ou "sistema indeterminado".

2.2.

a) Escreva um programa que leia 3 números e determine o maior e o menor desses números.

b) Escreva um programa que leia 3 números e os escreva por ordem decrescente de valor.

c) Escreva um programa que leia três valores positivos do tipo **`double`**, e determine se esses valores poderiam representar as medidas dos lados de um triângulo (nota: não é possível construir um triângulo se a soma do comprimento dos dois lados mais pequenos for inferior ou igual ao comprimento do lado maior).

2.3.

Escreva um programa que simule uma calculadora, isto é, que leia do teclado dois números e um carácter (+,–,* ou /) que indica uma operação aritmética, no formato "número operação número", e que apresente o resultado da operação indicada.

2.4.

O custo do transporte de uma certa mercadoria é determinado, em função do peso da mesma, do seguinte modo: se o peso for inferior a 500 gramas o custo é igual a 5 euros; se o peso estiver compreendido entre 500 gramas e 1000 gramas, inclusivé, o custo é igual a 5 euros mais 1.5 euros por cada adicional de 100 gramas ou fracção, acima de 500 gramas; se o peso for superior a 1000 gramas, o custo é 12.5 euros mais 5 euros por cada adicional de 250 gramas ou fracção, acima de 1000 gramas. Escreva um programa que dado o peso de uma determinada mercadoria determine o custo do seu transporte.

2.5.

Escreva um programa para determinar as raízes da equação quadrática $ax^2+bx+c=0$, sendo os coeficientes *a*, *b* e *c* fornecidos pelo utilizador. O programa deve indicar se a equação tem 2 raízes reais diferentes, 2 raízes reais iguais ou 2 raízes complexas conjugadas, e os respectivos valores (com 3 casas decimais).

Exemplo:

```
Introduza os coeficientes (a b c): 2.5 -1 16
A equação tem 2 raízes complexas conjugadas: 0.200+2.522i e 0.200-2.522i
```

2.6.

Um número **n** é primo se apenas for divisível pela unidade e por si próprio.

a) Escreva um programa que leia um número e determine se é primo ou não. Note que não é necessário testar todos os divisores no intervalo [2..n[, basta testar até ao inteiro inferior à raíz quadrada de **n**.

b) Escreva um programa que escreva os 100 primeiros números primos.

c) Escreva um programa que escreva todos os números primos inferiores a 10000.

2.7.

a) Escreva um programa que apresente no ecrã uma tabela dos senos, cosenos e tangentes dos ângulos compreendidos entre 0 e 90 graus (considera-se que os extremos estão incluídos), com intervalos de 15 graus, como se ilustra a seguir (notar o caso particular da última linha, correspondente ao ângulo de 90 graus):

```
ang      sen         cos         tan
  0   0.000000    1.000000    0.000000
 15   0.258819    0.965926    0.267949
 30   0.500000    0.866025    0.577350
 45   0.707107    0.707107    1.000000
 60   0.866025    0.500000    1.732051
 75   0.965926    0.258819    3.732051
 90   1.000000    0.000000    infinito
```

b) Altere o programa que desenvolveu na alínea anterior por forma a que os limites do intervalo e o incremento do valor dos ângulos constantes da tabela possa ser especificado pelo utilizador (por exemplo, se a gama for [0..1] e o incremento for de 0.1 graus, seria gerada a tabela para os ângulos de 0, 0.1, 0.2, …, e 1 grau).

2.8.

Escreva um programa para determinar e escrever o montante que um depositante poderá levantar do banco, ao fim de **n** anos de depósito de uma quantia **q**, sendo **j%** a taxa de juros anual. Os valores de **n**, **q** e **j** devem ser especificados pelo utilizador. Admita que os juros no final de cada ano são acumulados à quantia depositada.

2.9.

Escreva um programa que leia uma sequência de números inteiros, e determine e escreva a soma, a média, o menor e o maior dos números,

a) sendo o fim da sequência indicado pelo valor 0 (que já não faz parte da sequência); no final, o programa deve indicar o comprimento da sequência.

b) sendo o comprimento da sequência indicado previamente, pelo utilizador.

c) sendo o fim da sequência indicado quando o utilizador teclar *end of input* (CTRL-Z em Windows; CTRL-D em Linux)

2.10.

Uma capicua é um número ou um texto que se lê da mesma maneira de trás para a frente e da frente para trás. Por exemplo, os seguintes números são capicuas: 12321, 555, 45554 e 11611.
a) Escreva um programa que leia um inteiro de 3 dígitos e determine se é ou não uma capicua (sugestão: use os operadores de divisão e módulo para separar o inteiro nos seus dígitos).
b) Generalize o programa da alínea a) por forma a tratar inteiros com um maior número de dígitos.

2.11.

Escreva instruções **`for`** para calcular a soma dos primeiros *n* termos (com *n* inteiro previamente definido) de cada uma das séries seguintes:

a) Série que dá o valor da constante matemática π:

$$4-\frac{4}{3}+\frac{4}{5}-\frac{4}{7}+\frac{4}{9}-\frac{4}{11}+\ldots$$

b) Série que dá o valor da constante matemática *e*:

$$1+\frac{1}{1!}+\frac{1}{2!}+\frac{1}{3!}+\ldots$$

(Sugestão: calcule cada termo a partir do termo imediatamente anterior.)

c) Série que dá o valor de e^{-x} (com *x* real positivo previamente definido):

$$1-\frac{x}{1!}+\frac{x^2}{2!}-\frac{x^3}{3!}+\ldots$$

(Sugestão: calcule cada termo a partir do termo imediatamente anterior.)

2.12.

Repita o problema 2.11 por forma a que o utilizador possa especificar a precisão com que pretende o resultado, isto é, a máxima variação entre o valor da soma da série, entre duas iterações consecutivas. Note que a variação tanto pode ser positiva como negativa.

2-13.

Escreva um programa que leia um número inteiro e o decomponha em factores (exemplo: 20=2x2x5).

2.14.

a) A raíz quadrada de um número **n** pode ser calculada através do seguinte algoritmo, devido a Heron de Alexandria: partindo de uma estimativa inicial do valor da raíz, **rq**, calcular uma nova estimativa, **rqn**, usando a seguinte fórmula, **rqn = (rq + n / rq) / 2** ; repetir este cálculo usando como nova estimativa de **rq** o valor de **rqn**. O quadro seguinte ilustra a evolução do cálculo da raíz quadrada do número **n**=20, partindo de uma estimativa inicial **rq**=1.

rq	rqn	rqn^2	dif = n - rqn^2
1	10.500000	110.250000	-90.250000
10.50000	6.202381	38.469532	-18.469532
6.202381	4.713475	22.216844	-2.216844
4.713475	4.478314	20.055300	-0.055300
4.478314	4.472140	20.000001	-0.000039

Em geral, este algoritmo converge rapidamente para uma solução correcta, mesmo quando a estimativa inicial é fraca. Notar que a diferença entre o número **n** e o quadrado da raíz de **n** calculada, **dif**, evolui rapidamente para zero, pelo que se poderia usar o valor de **dif** como critério de paragem, isto é, terminar as iterações quando **dif** fosse inferior a um valor suficientemente pequeno, **delta** (por exemplo, **delta**=0.00001). No entanto, por uma questão de "segurança", é conveniente limitar o número de iterações, repetindo, nesse caso, os cálculos até que seja atingido um valor de **dif** inferior a **delta** ou que seja atingido um número máximo de iterações, **nMaxIter**.
Considerando o exemplo apresentado, se **delta**=0.001 e **nMaxIter**=3 o resultado final (**rqn**) seria o calculado após a 3ª iteração pois foi atingido o número máximo de iterações especificado, apesar de o valor de **delta** ainda ser superior a 0.001; mas se **delta**=0.001 e **nMaxIter**=10, o cálculo terminaria após a 5ª iteração pois nessa iteração o valor absoluto de **dif** já é inferior a 0.001.
Escreva um programa que leia os valores de **delta** e **nMaxIter** e que calcule a raíz quadrada de um número usando o algoritmo descrito anteriormente. Use sempre 1 como estimativa inicial de **rq**.
b) Modifique o programa da alínea anterior por forma a apresentar o resultado com o mesmo número de casas decimais usadas na especificação do valor de **delta** (*NOVO* - exemplo: se delta tiver o valor 0.001, o resultado deve ser apresentado com 3 casas decimais e se tiver o valor 0.000001, o resultado deve ser apresentado com 6 casas decimais); para isso deverá encontrar um algoritmo para determinar esse número de casas decimais. Mostre também o valor retornado pela função **sqrt()** da biblioteca de C e compare os valores retornados por esta função e pelo algoritmo de Heron.

2.15.

Escreva um programa que permita testar se o utilizador conhece as tabuadas da multiplicação. O programa deve gerar 2 números aleatórios (entre 2 e 9), apresentá-los ao utilizador, e perguntar o resultado da multiplicação desses números. Após ler a resposta do utilizador deve apresentar-lhe uma mensagem adequada tendo em conta a correcção da resposta e o tempo que o utilizador demorou a dá-la: se a resposta estiver errada, o utilizador deve receber a mensagem "Muito Mau", se responder certo e demorar menos de 5 segundos deve receber a mensagem "Bom", se responder certo mas demorar entre 5 segundos e 10 segundos (inclusivé) deve receber a mensagem "Satisfaz", se não deve receber a mensagem "Insuficiente".
Nota: A função **rand()** gera um número inteiro pseudo-aleatório no intervalo **[0..RAND_MAX]** em que **RAND_MAX** é uma constante simbólica definida na *header file* **<cstdlib>** (o standard ANSI especifica que **RAND_MAX** deve ter o valor de pelo menos **32767**). Para evitar que o gerador de números pseudo-aleatórios seja inicializado sempre com o mesmo valor, de cada vez que o programa for executado, gerando sempre a mesma sequência de números, inclua a instrução **srand(time(NULL))** no início da função **main()**; a função **srand()**, que permite especificar a "semente" do gerador de números aleatórios, está declarada em **<cstdlib>** e a função **time()**, que retorna o número de segundos que decorreram desde as zero horas do dia 1/Jan/1970, está declarada em **<ctime>**.

3. Funções

3.1.

Reescreva o programa do problema 1.6 (cálculo da área de um triângulo pela fórmula de Heron) por forma a usar as seguintes funções:

- **`area(double x1, double y1, double x2, double y2, double x3, double y3)`** para calcular a área de um triângulo cujos vértices têm coordenadas (x1,y1), (x2,y2) e (x3,y3);
- **`distance(double x1, double y1, double x2, double y2)`** para calcular a distância entre dois pontos cujas coordenadas são (x1,y1) e (x2,y2).

3.2.

Reescreva o programa do problema 2.6 (determinação de números primos) por forma a usar uma função **`isPrime()`** que determina se o número que lhe é passado como argumento é ou não primo. Escolha tipos adequados para o parâmetro e para o valor de retorno da função.

3.3.

a) Reescreva o programa do problema 2.14 por forma a usar uma função para calcular a raíz quadrada pelo método indicado nesse problema. Note que esta função deve ter 2 parâmetros: a precisão e o número máximo de iterações.
b) Escreva um programa de teste da função desenvolvida que permita comparar o resultado por ela devolvido com o que se obtém usando a função **`sqrt()`** da biblioteca de C/C++.

3.4.

Na linguagem C/C++, não existe nenhuma função que permita arredondar um número decimal para um número de casas decimais especificado. No entanto, é possível consegui-lo recorrendo à função **`floor`**. Por exemplo, para arredondar o valor de **`x`** para as centésimas (isto é, para 2 casas decimais), pode-se executar a instrução:

```
y = floor(x * 100 + 0.5) / 100;
```

Escreva uma função cujo protótipo seja

```
double round(double x, unsigned n)
```

que arredonde um número em vírgula flutuante, **`x`**, para um dado número de casas decimais, **`n`**, retornando o valor arredondado.
Desenvolva um programa de teste dessa função que pergunte ao utilizador os valores de **`x`** e **`n`** e imprima o valor de **`round(x,n)`**.

3.5.

A linguagem C++ não dispõe de variáveis do tipo "fracção" nem, obviamente, de operadores ou funções para manipular fracções. Pretende-se desenvolver um conjunto de funções que permitam manipular fracções, representadas, por enquanto, por variáveis independentes que representam o numerador e o denominador de cada fracção (adiante será proposto um exercício em que o numerador e o denominador de uma fracção serão representados numa estrutura comum).

a) Escreva uma função cujo protótipo é

```
bool readFracc(int &numerator, int &denominator)
```

que lê do teclado uma fração inteira (fração cujo numerador e denominador são números inteiros), no formato N/D (exemplo: 2/3) e actualiza as variáveis indicadas por **`numerator`** e **`denominator`** por forma a conterem esses valores. O valor de retorno da função deve ser **`true`** caso os valores introduzidos para o numerador e para o denominador sejam válidos e o separador também seja válido, isto é **`'/'`**, e **`false`** no caso contrário. Neste último caso os valores devolvidos, do numerador e do denominador, devem ser zero. A função não deve escrever nada no ecrã; qualquer mensagem indicativa dos valores a ler deve ser apresentada ao utilizador antes de invocar a função.

b) Escreva uma função cujo protótipo é

```
void writeFracc(int numerator, int denominator)
```

que mostra no ecrã a fração cujo numerador e denominador lhe forem passados como arugumentos, no formato "numerador/denominador" (ex: **`2/3`**).

c) Escreva uma função cujo protótipo é

```
void reduceFracc(int &numerator, int &denominator)
```

que reduz a fracção cujo numerador e denominador lhe são passados como parâmetros, dividindo-os pelo seu máximo divisor comum. Sugestão: para determinar o máximo divisor comum do numerador e do denominador, procure a versão não recursiva do algoritmo de Euclides e implemente-a.

d) Escreva funções para realizar as operações básicas (soma, subtracção, multiplicação e divisão) sobre fracções, apresentando o resultado na forma reduzida. Escolha protótipos adequados para essas funções.

e) Escreva um programa de teste das funções desenvolvidas.

3.6.

Pretende-se desenvolver um programa que mostre, no ecrã, o calendário de um dado ano. O desenvolvimento deste programa será feito modularmente, recorrendo a diversas funções que "resolvem" alguns dos sub-problemas do problema proposto. Além destas funções, descritas nas alíneas seguintes, outras poderão revelar-se úteis / necessárias.

a) Um ano bissexto é um ano que satisfaz as seguintes condições: é divisível por 4 mas não divisível por 100; no entanto, os anos divisíveis por 400, apesar de serem divisíveis por 100, são considerados bissextos (ex: o ano 2000 foi bissexto mas o ano 2100 não será). Escreva uma função que tenha como parâmetro um número inteiro representando um ano e retorne um valor booleano, indicando se o ano é ou não bissexto (**`true`** se for e **`false`** se não for). Escreva um programa que teste essa função.

b) Escreva uma função que tenha como parâmetros dois inteiros, representando um mês e um ano, e devolva o número de dias desse mês, nesse ano. Note que apenas o mês de Fevereiro tem um número de dias variável, consoante o ano seja ou não bissexto.

c) Em Novembro de 2004, Sohael Babwani publicou um artigo na Mathematical Gazette em que descreve uma fórmula para calcular o dia da semana (domingo, segunda-feira, …) correspondente a uma determinada data do calendário Gregoriano. A fórmula é a seguinte:

$$ds = \left(\left\lfloor \frac{5 \cdot a}{4} \right\rfloor + c + d - 2 \cdot (s\%4) + 7 \right)\%7$$

em que

- $\lfloor\ \rfloor$ – operador que calcula o "inteiro contido em"
- % – operador que calcula o "resto da divisão inteira"
- *ds* – dia da semana
- *d* – dia do mês
- *m* – número do mês (1-Janeiro, 2-Fevereiro, …)
- *s* – dois primeiros dígitos do ano (ex: se o ano for 2010, *s* tomará o valor 20)
- *a* – dois últimos dígitos do ano (ex: se o ano for 2010, *a* tomará o valor 10)
- *c* – código do mês, dado pela seguinte tabela, em que *m* representa o número do mês (1-Janeiro, 2-Fevereiro, …); notar que o código depende de o ano ser ou não bissexto

		c	
mês	***m***	**bissexto**	**não-bissexto**
Janeiro	1	6	0
Fevereiro	2	2	3
Março	3	3	3
Abril	4	6	6
Maio	5	1	1
Junho	6	4	4

		c	
mês	***m***	**bissexto**	**não-bissexto**
Julho	7	6	6
Agosto	8	2	2
Setembro	9	5	5
Outubro	10	0	0
Novembro	11	3	3
Dezembro	12	5	5

O resultado, *ds*, deve ser interpretado da seguinte forma: 0 = Sábado, 1 = Domingo, 2 = Segunda-feira, etc.
Por exemplo, aplicando a fórmula ao dia 1 de Janeiro de 2011 obtém-se:

$$ds = \left(\left\lfloor \frac{5 \cdot 11}{4} \right\rfloor + 0 + 1 - 2 \cdot (20\%4) + 7 \right)\%7 = 0$$

indicando que o dia da semana correspondente é Sábado.

Escreva uma função que tenha como parâmetros 3 números inteiros, representando uma data (ano, mês, dia), e que devolva um inteiro indicando o dia da semana correspondente. Escreva um programa que leia uma data e, usando essa função, escreva o nome do dia da semana correspondente (**`Domingo`**, **`Segunda-feira`**, …).

d) Escreva uma função que, usando a função desenvolvida na alínea anterior, mostre no ecrã o calendário de um mês/ano especificado pelo utilizador, com um formato semelhante ao seguinte:

```
Janeiro/2011
Dom  Seg  Ter  Qua  Qui  Sex  Sáb
                                1
  2    3    4    5    6    7    8
  9   10   11   12   13   14   15
 16   17   18   19   20   21   22
 23   24   25   26   27   28   29
 30   31
```

e) Finalmente, escreva um programa que, recorrendo às funções anteriormente desenvolvidas e a outras que considere necessárias, mostre no ecrã o calendário de todos os meses de um ano indicado pelo utilizador.

3.7.

a) Escreva uma função, **`factorial_ite(int n)`**, que determine, de forma iterativa, o factorial de um número. Declare a variável que vai conter o resultado como sendo do tipo **`long`**. Comece por determinar qual o maior inteiro que pode ser representado numa variável desse tipo e qual o maior número cujo factorial é inferior esse inteiro. Nota: existe uma constante, **`LONG_MAX`**, definida em **`<climits>`** que contém o valor do maior inteiro representável numa variável do tipo **`long`**.

b) Escreva uma função, **`factorial_rec(int n)`**, que determine, de forma recursiva, o factorial de um número.

3.8.

O algoritmo de Euclides para determinar o máximo divisor comum (*mdc*) entre 2 números, *m* e *n*, pode ser definido recursivamente da seguinte forma:

- se *n* for divisor de *m*, o *mdc*(*m*,*n*) é *n*
- caso contrário, o *mdc*(*m*,*n*) é o *mdc*(*n*, resto da divisão de *m* por *n*)

Escreva uma função recursiva que determine o máximo divisor comum de 2 números que lhe são passados como parâmetros. Escreva um programa de teste dessa função.

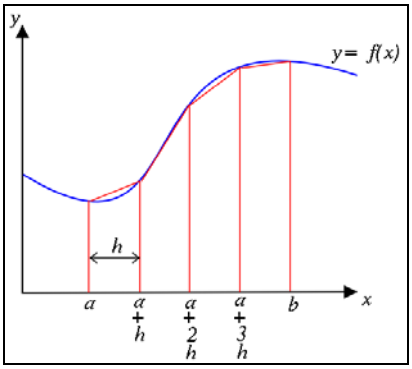
3.9.

A "regra dos trapézios" é um método numérico para calcular um valor aproximado para o integral definido. Para calcular o integral

$$\int_a^b f(x)\,dx$$

a área sob a curva *y=f(x)* é dividida em *n* regiões, cada uma com uma largura *h=(b-a)/n*. A área de cada região é aproximada pela área de um trapézio. A soma das áreas de todos os trapézios dá um valor aproximado do integral definido. A área do *i*-ésimo trapézio (i=1,2,…) é dada por

$$\frac{h}{2}\left(f(a+(i-1)h)+f(a+ih)\right)$$



Em geral, a estimativa do integral melhora com o decréscimo de *h*.
Escreva uma função cujo protótipo é

`double integrateTR(double f(double), double a, double b, int n)`,

com parâmetros *f* ,*a*, *b* e *n*, anteriormente definidos, que implemente o cálculo do integral usando este método.
Escreva um programa de teste dessa função, usando valores de *n* iguais a 2,4,8,16,32,64 e 128, sobre as funções

$g(x)=x^2$ com $a=0, b=10$ e $h(x)=\sqrt{4-x^2}$ com $a=-2, b=2$.

A função *h* define um semi-círculo com raio 2. Compare a estimativa obtida com a área real do semi-círculo.

Faculdade de Engenharia da Universidade do Porto

Mestrado Integrado em Engenharia Informática e Computação

PROGRAMAÇÃO

INTRODUÇÃO À PROGRAMAÇÃO EM LINGUAGEM C++

4. *Arrays, Vectors, Strings*, *Structs* e *Files*

4.1.

a) Escreva uma função **`bool isHydroxide(char compound[])`** que tem uma *string* de C como parâmetro e devolve **`true`** ou **`false`** consoante essa *string* terminar ou não com os caracteres OH. Escreva um programa que teste a função com os seguintes dados: KOH, H2O2, NaCl, NaOH, C9H8O4 e MgOH.
b) Idem, usando uma *string* da STL como parâmetro: **`bool isHydroxide(string compound)`**.

4.2.

Escreva uma função **`bool sequenceSearch(string s, int nc, char c)`** que verifica se na *string* **`s`** existe uma sequência de **`nc`** caracteres consecutivos iguais a **`c`**, devolvendo **`true`** ou **`false`** consoante exista ou não. Escreva um programa de teste dessa função.

4.3.

Escreva uma função **`void decompose(string compound)`** que tem com parâmetro uma *string* representando um composto químico e mostra no ecrã os seus elementos constituintes. Por exemplo, o composto H2O (água) tem como elementos constituintes H e O; o composto NaCl tem como elementos constituintes Na e Cl. Teste a função com os seguintes dados: H2O, KOH, H2O2, NaCl, NaOH, C9H8O4 e MgOH.

4.4.

Escreva uma função **`string normalizeName(string name)`** para "normalizar" um nome da seguinte forma: espaços no início ou no fim são suprimidos; sequências de vários espaços entre duas palavras são substituídos por um único espaço; as minúsculas são convertidas para maiúsculas; são removidas as partículas "DE", "DO", "DA", "DAS", "DOS" e "E". O valor de retorno da função é o nome "normalizado". Sugestão: use um *array* de *strings* para guardar as partículas a remover.

4.5.

Resolva o problema 3.5 usando uma **`struct`** para representar uma fracção:

```
struct Fraction {
   int numerator;
   int denominator;
};
```

O protótipo das funções **`readFracc()`**e **`reduceFracc()`** passará a ser **`Fraction readFracc()`**e **`Fraction reduceFracc(Fraction f)`**, respectivamente. O protótipo das outras funções desenvolvidas deverá ser modificado de modo semelhante.

4.6.

a) Escreva uma função, cujo protótipo é **`void readIntArray(int a[], int nElem),`** que lê do teclado os elementos de um *array* de inteiros, **`a`**; o número de elementos a ler é **`nElem`**. Antes de ler cada elemento do *array*, deve ser apresentado no ecrã o respectivo índice.

b) Escreva uma função, cujo protótipo é **`int searchValueInIntArray(const int a[], int nElem, int value)`** que pesquisa nos **`nElem`** elementos do *array* **`a`** a ocorrência do valor **`value`**. Caso **`value`** seja encontrado, a função deve devolver o índice do elemento do *array* cujo valor é **`value`**; caso contrário devolve **`-1`**. No caso de múltiplas ocorrências de **`value`** deve ser retornado o índice da primeira ocorrência.

c) Escreva um programa de teste das funções **`readIntArray()`** e **`searchValueInIntArray()`**.

d) Modifique a função **`searchValueInIntArray()`**, por forma a resultar uma nova função **`int searchMultValuesInIntArray(const int a[], int nElem, int value, int index[])`**que, caso existam múltiplas ocorrências de **`value`** em **`a[]`**, devolva no *array* **`index[]`** os índices das ocorrências de **`value`**. O valor retornado pela função deve ser o número de ocorrências, isto é, o número de elementos válidos de **`index[]`**.
Nota: esta função também poderia chamar-se **`searchValueInIntArray`**, como a função anterior; explique porquê.
e) Altere o programa desenvolvido na alínea **c)** por forma a usar a função **`searchMultValuesInIntArray()`**.
Nota: o espaço para o vector **`index[]`** deve ser reservado antes de invocar a função **`searchMultValuesInIntArray()`**, tendo em conta o número máximo de ocorrências esperado (no limite, deve ter **`nElem`** elementos).

4.7.

a) Repita o problema anterior, usando *vectors* da STL para guardar os números, em vez de *arrays* de C. Os protótipos das funções a desenvolver são, neste caso, os seguintes:

- `void readIntVector(vector<int> &v, int nElem);`
  `//OU vector<int> readIntVector(int nElem);`
- `int searchValueInVector(const vector<int> &v, int value);`
- `vector<int> searchMultValuesInIntVector(const vector<int> &v, int value);`

b) Escreva novas versões de **`readIntVector()`**em que o número de elementos não tenha de ser especificado como parâmetro; o utilizador deverá teclar **`CTRL-Z`** para indicar que terminou a entrada dos elementos do vector.

- `void readIntVector(vector<int> &v);`
- `vector<int> readIntVector();`

4.8.

a) O método *bubblesort*, para ordenar os elementos de um vector com *N* elementos, consiste no seguinte:
- começando no 1º elemento do vector, comparar os elementos de índices *i* e *i+1* do vector e trocar a sua posição se eles estiverem fora de ordem;
- repetir o passo anterior para os restantes elementos do vector, isto é, começando no 2º elemento, depois no 3º elemento, etc., até ao elemento de índice *N-1*, o qual será comparado com o elemento de índice *N*; depois destes passos, o maior (ou menor, consoante a ordenação seja feita por ordem crescente ou decrescente) elemento do vector estará na posição correcta (a última posição do vector);
- repetir os passos anteriores, considerando em cada iteração apenas os elementos ainda não ordenados do vector; após a 2ª iteração os últimos 2 elementos estarão na posição correcta, após a 3ª iteração os 3 últimos elementos estarão na posição correcta, e assim sucessivamente.
Escreva uma função **`void bubblesort(vector<string> &v)`** que implemente este método de ordenação para ordenar um vector de nomes de pessoas por ordem crescente. Nota: a STL dispõe de algoritmos de ordenação que serão analisados posteriormente.

b) Escreva um programa de teste dessa função.

4.9.

a) O algoritmo de pesquisa binária que se segue pode ser usado para procurar um valor num vector ordenado por ordem crescente:
- fazer *bottom* igual ao índice do 1º elemento do vector;
- fazer *top* igual ao índice do último elemento do vector;
- fazer *found* igual a *falso*;
- repetir, enquanto *bottom* não for maior do que *top* e *found* for igual a *falso*
  - fazer *middle* igual ao índice do elemento que está a meio caminho entre *bottom* e *top*
  - se o valor que ocupa a posição *middle* for o valor procurado
    - fazer *found* igual a *verdadeiro*

    senão, se o valor que ocupa a posição *middle* for ~~maior~~ menor que o valor procurado
    - fazer *bottom* igual a *middle+1*

    senão
    - fazer *top* igual a *middle-1*

Escreva uma função, **`int binarySearch(const vector<string> &v, ~~int~~ string value)`** que implemente este método de pesquisa aplicado ao vector **`v`**. A função deve retornar o índice do elemento cujo valor é **`value`** caso ele exista em **`v`** ou **`-1`**, no caso de ele não existir.

b) Escreva um programa de teste dessa função.

4.10.

Escreva uma função **`void removeDuplicates(vector<int> &v)`** que elimina os elementos repetidos do vector **`v`**. A ordem original dos elementos do vector deve ser mantida. Escreva um programa de teste dessa função. Sugestão: a eliminação de um elemento pode ser efectuada deslocando todos os elementos que ocupam as posições seguintes do vector para a posição anterior à que ocupam e alterando o tamanho do vector, usando o método **`resize()`** da classe **`vector`**, para diminuir o tamanho do vector. Nota: existem métodos da classe **`vector`** que permitem eliminar elementos de um vector e algoritmos da STL que permitem eliminar elementos repetidos de um vector; estes métodos serão estudados posteriormente.

4.11.

a) Desenvolver duas funções cujos protótipos são

```
vector<int> union(const vector<int> &v1, const vector<int> &v2);
vector<int> intersection(const vector<int> &v1, const vector<int> &v2);
```

que realizem a reunião e a intersecção dos elementos dos vectores de inteiros **`v1`** e **`v2`**, retornando o respectivo resultado. O vector resultante não deve ter elementos repetidos e os elementos devem ficar ordenados por ordem crescente. Sugestão: use as funções **`bubblesort()`** e **`removeDuplicates()`** anteriormente desenvolvidas, antes de fazer a reunião/intersecção.

b) Desenvolver um programa de teste dessas funções.

4.12.

Escreva uma função **`double executeOperation(string op)`** que tem como parâmetro uma *string* representando uma operação aritmética simples (soma, subracção, multiplicação ou divisão) de dois números que podem ter ou não parte decimal e que devolve o resultado da operação (do tipo double). Por exemplo, se for executada a seguinte chamada à função **`executeOperation("12.3 + 5")`**ela deve devolver o valor **`17.3`**. Sugestão: use *stringstreams* para extrair o valor dos operandos e do operador.

4.13.

Desenvolver um programa que ordene os nomes de um conjunto de pessoas contidos num ficheiro de texto. guardando o resultado noutro ficheiro de texto. O nome do ficheiro original, com a extensão **`.txt`**, deve ser indicado pelo utilizador (ex: **`names.txt`**); o nome do ficheiro resultante deve ser igual ao do ficheiro original, acrescentando a string "**`_sorted`**" e mantendo a extensão (ex: **`names_sorted.txt`**).