Ficha 1

Programação (L.EIC009)

Objectivos

- Ambientação aos laboratórios.
- Uso do compilador gcc na linha de comandos.
- Introdução à linguagem C.
  - Aspectos básicos.
  - Funções e fluxo de controlo usando `if-else`, `for` e `while`.
  - I/O com `scanf` e `printf`.
  - Uso de apontadores para passagem por referência de argumentos a funções.

Recursos

Slides das aulas teóricas: HTML PDF

1

Considere o código C para o um programa que imprime "Hello World!" dado em `hello.c`:

```c
/*
  A simple program that prints "Hello world!"
*/
#include <stdio.h>

int main() {
  // Print "Hello world!" ...
  printf("Hello world!\n");
  return 0;
}
```

1.1

Usando a linha de comandos, invoque o gcc para compilar o programa e de seguida execute-o:

```
$ gcc hello.c -Wall -o hello
$ ./hello
Hello world!
```

1.2

Considere agora a seguinte variação do programa anterior:

```
  /*
    A simple program that prints "Hello world!"
  */
#include <stdiox.h>

int main() {
  return 0;
}

int main() {
  // Print "Hello world!" ...
  printf("Hello world!\n);
  return 0
  /*
   end ...
```

Edite um ficheiro C `hello2.c` com o conteúdo acima. Acerte os erros progressivamente até o programa compilar correctamente.

```
$ gcc hello2.c -Wall -o hello2
```

2

Em cada um dos seguintes casos identifique o valor final das variáveis `a` a `e`. Pode escrever um programa para confirmar a sua análise imprimindo os valores em causa com uma chamada a `printf`:

```
printf("a=%d b=%d c=%d d=%d e=%d\n", a, b, c, d, e);
```

(a)

```
int a = 3;
```

```c
int b = 2;
int c = (a + b) * b;
int d = a + b * b;
int e = c / d + c % d;
```

(b)

```c
int a = 1;
int b = a++;
int c = ++a;
int d = --a;
int e = a--;
```

(c)

```c
int a = 10, b = -20, c = 0x1E, d = -40, e = 50;
b += a;
c -= a;
d *= a;
e /= a;
```

(d)

```c
int a = 1;
int b = a % 2 == 1;
int c = a > 0 && b < 0;
int d = c == 1 || b > 0;
int e = !d ? 0 : 1;
```

3

3.1

Escreve um programa em C que leia 2 números de tipo `int` e imprima os valores mínimo, máximo, e soma dos valores lidos. Empregue `scanf` para leitura e `printf` para impressão como ilustrado pelo esqueleto a seguir:

```c
#include <stdio.h>

int main(void) {
  // Read values
  int a, b;
```

```
  printf("Enter values: ");
  scanf("%d %d", &a, &b);

  // Calculate min, max, and sum
  int min, max, sum;
  ... // TODO
  printf("min: %d, max: %d, sum: %d\n", min, max, sum);
  return 0;
}
```

Exemplo de execução:

```
Enter values: 7 -3
min: -3, max: 7, sum: 4
```

3.2

Generalize o programa anterior para `n` números de tipo `int` a partir do esqueleto abaixo. São usadas as definições definições de `INT_MIN` e `INT_MAX` definidos no "header" `limits.h`, constantes para os valores mínimos e máximos representáveis usando o tipo `int`. O propósito é facilitar a inicialisação de `min` e `max`.

```
#include <stdio.h>
#include <limits.h>
int main(void) {
  int n;
  printf("How many numbers? ");
  scanf("%d", &n);

  int min = INT_MAX;
  int max = INT_MIN;
  int sum = 0;

  for (int i = 0; i < n; i++) {
    int v;
    printf("Enter value: ");
    scanf("%d", &v);
    ... // TODO
  }
  printf("min: %d, max=%d, sum=%d\n", min, max, sum);
  return 0;
}
```

Exemplo de execução:

```
How many numbers? 3
Enter value: 7
Enter value: -5
Enter value: 20
min: -5, max=20, sum=22
```

4

Em C não existe um operador para calcular a potência de um número (ao contrário de por exemplo Python em que existe o operador `**` ou de Matlab em que existe o operador `^`), i.e., calcular x^n dados x e $n \geq 0$.

Escreva uma função `power` que calcule a potência de um número inteiro, usando o esqueleto abaixo. Em `main` é chamada `power` para imprimir as potências de 2 e de 3 até $n = 10$.

Sugestão: Podemos obviamente entanto calcular x^n efectuando n multiplicações sucessivas. Use um ciclo `for` no corpo de `power`. E não se esqueça que $x^0 = 1$ ...

```c
#include <stdio.h>
int power(int x, int n) {
  ...
  for ...
  ...
}

int main(void) {
  for (int n = 0; n <= 10; n++) {
    printf("2^%d: %d\n", n, power(2,n));
    printf("3^%d: %d\n", n, power(3,n));
  }
  return 0;
}
```

Output esperado do programa:

```
2^0: 1
3^0: 1
2^1: 2
3^1: 3
2^2: 4
3^2: 9
```

```
2^3: 8
3^3: 27
2^4: 16
3^4: 81
2^5: 32
3^5: 243
2^6: 64
3^6: 729
2^7: 128
3^7: 2187
2^8: 256
3^8: 6561
2^9: 512
3^9: 19683
2^10: 1024
3^10: 59049
```

5

Considere o cálculo do máximo divisor comum de dois números inteiros segundo o algoritmo de Euclides, que é expresso pela recorrência:

$$\text{mdc}(a, b) = \begin{cases} a & , \text{ se } b = 0 \\ \text{mdc}(b, a\%b) & , \text{ se } b \neq 0 \end{cases}$$

e o seguinte esqueleto para um programa que contém a função `mdc` e um programa expresso em `main` para teste.

```c
#include <stdio.h>

int mdc(int a, int b) {
  ...
}

int main(void) {
  int a, b;
  printf("Enter values: ");
  scanf("%d %d", &a, &b);
  printf("mdc(%d,%d): %d\n", a, b, mdc(a, b));
  return 0;
}
```

5.1

Codifique `mdc` recorrendo a um ciclo `while`.

Exemplos de execução:

```
Enter values: 1 22
mdc(1,22): 1
```

```
Enter values: 35 37
mdc(35,37): 1
```

```
Enter values: 12 4
mdc(12,4): 4
```

```
Enter values: 81 45
mdc(81,45): 9
```

```
Enter values: 45 81
mdc(45,81): 9
```

```
Enter values: 5000 1350
mdc(5000,1350): 50
```

5.2

Codifique `mdc` de forma recursiva.

6

Podemos converter uma duração absoluta em segundos no número de horas, minutos e segundos restantes (descontando as horas e minutos). Por exemplo 7636 segundos correspondem a 2 horas ($2x3600 = 7200$), 7 minutos ($7 \times 60 = 420$), e 16 segundos ($7200 + 420 + 16 = 7636$).

Escreva uma função em C `convert` para uma conversão deste tipo de acordo com o esqueleto abaixo. O argumento `d` expressa a duração e os restantes argumentos `h`, `m` e `s` são **apontadores** de tipo `int`.

Recorde-se o significado dos operadores `&` e `*`:

- `&a` dá-nos o apontador de memória relativo a `a`, i.e., se `a` tem tipo `int`

como no exemplo então `&a` tem tipo `int*`;

- `*p` onde `p` é um apontador permite-nos atribuir ou ler (conforme o uso) um valor da posição referenciado por `p`, i.e., se `p` tem tipo `int*` como no exemplo então `*p` tem tipo `int`.

```c
#include <stdio.h>

void convert(int d, int* ph, int* pm, int* ps) {
  *ph = ...; // TODO
  *pm = ...;
  *ps = ...;
}

int main(void) {
  int d, h, m, s;
  printf("Enter duration: ");
  scanf("%d", &d);
  convert(d, &h, &m, &s);
  printf("h: %d, m: %d, s: %d\n", h, m, s);
  return 0;
}
```

Exemplos de execução:

```
Enter duration: 7636
h: 2, m: 7, s: 16
```

```
Enter duration: 69
h: 0, m: 1, s: 9
```

```
Enter duration: 699
h: 0, m: 11, s: 39
```

```
Enter duration: 6999
h: 1, m: 56, s: 39
```

7

A equação quadrática

$$ax^2 + bx + c = 0$$

de números reais com $a \neq 0$ tem 0, 1, ou 2 soluções conforme o valor de $\Delta = b^2 - 4ac$:

- 0 soluções se $\Delta < 0$
- 1 solução se $\Delta = 0$ dada por

$$x_1 = \frac{-b}{2a}$$

- 2 soluções se $\Delta > 0$ dadas por

$$x_1 = \frac{-b - \sqrt{\Delta}}{2a} \quad \text{e} \quad x_2 = \frac{-b + \sqrt{\Delta}}{2a}$$

7.1

Escreva uma função `solve_eq` de acordo com o esqueleto abaixo. A função toma como argumentos coeficientes de tipo `int` para a equação (`a` , `b` e `c`) e apontadores (`x1` , `x2`) de tipo `float` para a escrita dos valores solução, e deve retornar o número de soluções (0 a 2). A função deve também retornar o valor de erro `-1` caso `a==0` .

No caso em que `delta > 0` deve usar a função `sqrt` para o cálculo da raíz quadrada, definida no "header" `math.h` . Precisa de compilar o programa com opção `-lm` para o "linker" incluir esta função no programa gerado, i.e., `gcc prog.c -Wall -o prog -lm` se `prog.c` é o nome do seu ficheiro.

```c
#include <stdio.h>
#include <math.h>

int solve_eq(int a, int b, int c, double* x1, double* x2) {
  int delta = b * b - 4 * a * c;
  int sols;
  ...
  return sols;
}

int main(void) {
  int a, b, c, sols;
  double x1, x2;

  printf("Enter values for a, b, and c: ");
  scanf("%d %d %d", &a, &b, &c);

  sols = solve_eq(a, b, c, &x1, &x2);

  if (sols == 0) {
```

```
    printf("No solutions!\n");
  } else if (sols == 1) {
    printf("One solution: %f\n", x1);
  } else if (sols == 2) {
    printf("Two solutions: %f and %f\n", x1, x2);
  } else {
    printf("Error: a is 0!\n");
  }
  return 0;
}
```

Exemplos de execução:

```
Enter values for a, b, and c: 0 2 1
Error - a is 0!

Enter values for a, b, and c: 1 0 5
No solutions!

Enter values for a, b, and c: 2 -2 3
No solutions!

Enter values for a, b, and c: 1 0 0
One solution: 0.000000

Enter values for a, b, and c: 1 -2 1
One solution: 1.000000

Enter values for a, b, and c: 16 -8 1
One solution: 0.250000

Enter values for a, b, and c: 1 -1 0
Two solutions: 0.000000 and 1.000000

Enter values for a, b, and c: 16 0 -1
Two solutions: -0.250000 and 0.250000

Enter values for a, b, and c: 1 0 -1
Two solutions: -1.000000 and 1.000000

Enter values for a, b, and c: 3 4 -2
Two solutions: -1.720759 and 0.387426
```

7.2

Converta o bloco `if-else` em `main` num bloco `switch-case`.