Ficha 5

Programação (L.EIC009)

Objectivos

- Transição de C para C++.

Recursos

Slides das aulas teóricas: HTML PDF

0. Validação do ambiente

CLion e CMake

Na imagem Linux dos laboratórios o CLion está instalado no directório `/opt/clion-2021.2.1`.

Será conveniente configurar a variável de ambiente `PATH` da seguinte forma:

```
export PATH=/opt/clion-2021.2.1/bin:/opt/clion-2021.2.1/bin/cmake/linux/bin
```

(pode adicionar a linha acima ao final do ficheiro `.bashrc` para tornar a configuração activa cada vez que inicia uma linha de comando).

Depois de configurar `PATH` execute `cmake --version` na linha de comandos para verificar se o CMake está instalado. O CLion por sua vez pode ser lançado com `clion.sh`.

Ficheiro `CMakeLists.txt`

Considere a seguinte definição inicial para o ficheiro `CMakeLists.txt`.

```
cmake_minimum_required(VERSION 3.10)
project(progp5)

set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -Wall -g -fsanitize=address -fsanit
```

Por cada programa `prog` que queira compilar bastará adicionar uma linha ao ficheiro:

```
add_executable(prog prog.cpp)
```

1

Considere o código C++ abaixo para o um programa que imprime `"Hello World! ++"`.

```
/*
  A simple program that prints "Hello world! ++"
*/
#include <iostream>

int main() {
  // Print "Hello world! ++" ...
  std::cout << "Hello world! ++" << std::endl;
  return 0;
}
```

1.1

Compile o programa na linha de comando e execute-o de seguida:

```
$ g++ -Wall hello.cpp -o hello
$ ./hello
Hello world! ++
```

1.2

Agora compile o programa no contexto do projecto CMake. Como indicado acima, precisa de adicionar a `CMakeLists.txt` a seguinte configuração:

```
add_executable(hello hello.cpp)
```

2

Escreve um programa C++ que peça ao utilizador um conjunto de números inteiros e

imprima o máximo, mínimo e soma dos valores lidos. Use `std::cin` e `std::cout`, completando o seguinte esqueleto:

```
#include <iostream>
#include <climits>

int main(void) {
  int n;
  std::cout << "How many numbers? ";
  std::cin >> n;
  int min = INT_MAX;
  int max = INT_MIN;
  int sum = 0;
  ...
}
```

Exemplo de execução:

```
How many numbers? 3
Enter value: 7
Enter value: -5
Enter value: 20
Min: -5
Max: 20
Sum: 22
```

3

Escreve um programa que, tal como o anterior, leia uma sequência de de inteiros e calcule a mediana dos valores. Relembrando (de um exercício da ficha 3), para um array `a` com `n` elementos:

- podemos começar por ordenar `a`;
- se `n` é impar então a mediana é dada por `a[n / 2]` (após a ordenação);
- se `n` é par, a mediana é dada pela média dos valores `a[n / 2 - 1]` e `a[n / 2]`.

Para alocar e libertar o array, empregue: - os operadores `new` e `delete` respectivamente; - a função `std::sort` (inclúa o header `algorithm`) para ordenar o array como ilustrado abaixo.

Esqueleto:

```cpp
#include <algorithm>
#include <iostream>

int main(void) {
   int n;
   std::cout << "How many numbers? ";
   std::cin >> n;
   std::cout << "Enter values: ";
   // Allocate array
   int* a = new int[n];
   ... // Read values
   std::sort(a, a + n);
   double median;
   ...
   std::cout << "Median: " << median << std::endl;
   // Release memory
   delete [] a;
   return 0;
}
```

Exemplos de execução:

```
How many numbers: 5
Enter values: 10 11 10 14 16
Median: 11
```

```
How many numbers: 6
Enter values: 10 11 10 14 16 12
Median: 11.5
```

4

Escreva o código das seguintes "templates" para funções `max_value` e `norm_values` tal que:

- `max_value(arr, n)` devolve o valor máximo dos `n` elementos em `arr`;
- `norm_values(arr, n, min, max)` normaliza o valor dos `n` elementos em `arr` mediante a conversão de valores inferiores a `min` em `min`, valores superiores a `max` em `max`, e não alterando outros valores compreendidos entre `min` e `max` - por ex. a normalização de `{ 2, -1, 2, 5, 1, 4}` para valores entre `0` e `3` deverá levar a `{ 2, 0, 2, 3, 1, 3}`).

```
template <typename T>
T max_value(const T arr[], int n) {
  ...
}
template <typename T>
void norm_values(T arr[], int n, T min, T max) {
 ...
}
```

Escreva um programa que teste o código usando um array de valores de tipo `int` e outro array de valores de tipo `double`, ex. algo como:

```
int iarr[6] = { 2, -1, 2, 5, 1, 4};
int imax = max_value(iarr, 6);
norm_values(iarr, 6, 0, 3);
...
double darr[6] = { -1.2, 0.5, 1.3, 3.2, -0.7, 1.1 };
double dmax = max_value(darr, 6);
norm_values(darr, 6, -1.0, 1.0);
...
```

5

Tenha em conta o seguinte esqueleto para a estrutura de dados `coord2d` e a definição de operadores "overloaded" em associação a esse tipo de dados:

```
#include <iostream>

struct coord2d {
    int x;
    int y;
};

std::istream& operator>>(std::istream& is, coord2d& c) {
    is >> c.x >> c.y;
    return is;
}
std::ostream& operator<<(std::ostream& os, const coord2d& c) {
    os << c.x << ' ' << c.y;
    return os;
}
coord2d operator+(const coord2d& a, const coord2d& b) {
    coord2d r;
```

```
    r.x = a.x + b.x;
    r.y = a.y + b.y;
    return r;
}
coord2d operator*(int f, const coord2d& c) {
    coord2d r;
    r.x = f * c.x;
    r.y = f * c.y;
    return r;
}
```

5.1

Experimente ler e escrever coordenadas usando os operadores `>>` e `<<` definidos acima, ex.

```
coord2d a, b;
std::cout << "Enter a and b: ";
std::cin >> a >> b;
coord2d c = 2 * a + b;
std::cout << "2 * a + b = " << c << std::endl;
```

```
Enter a and b: 1 2 3 4
2 * a + b = 6 8
```

5.2

Defina adicionais operadores para a subtração (`-` binário) de coordenadas e a negação dos valores de uma coordenada (`-` unário):

```
coord2d operator-(const coord2d& a, const coord2d& b) { ... }
coord2d operator-(const coord2d& c) { ... }
```

Escreva um programa que calcule `- a + 2 * b - c` para três coordenadas `a` , `b` e `c` introduzidas pelo utilizador.

```
Enter a, b, c : 1 2 3 4 5 6
- a + 2 * b - c = 0 0
```

6

Escreva uma variante do código anterior para definir a template `coord2d<T>`, onde `T` é o tipo das coordenadas `x` e `y`. Teste de seguida um programa que calcule `- a + 2 * b + c` para três itens `coord2d<double>` `a`, `b` e `c` introduzidos pelo utilizador.

Solução parcial:

```
#include <iostream>

template <typename T>
struct coord2d {
    T x;
    T y;
};

template <typename T>
std::istream& operator>>(std::istream& is, coord2d<T>& c) {
    is >> c.x >> c.y;
    return is;
}

template <typename T>
std::ostream& operator<<(std::ostream& os, const coord2d<T>& c) {
    os << c.x << ' ' << c.y;
    return os;
}

template <typename T>
coord2d<T> operator+(const coord2d<T>& a, const coord2d<T>& b) {
    coord2d<T> r;
    r.x = a.x + b.x;
    r.y = a.y + b.y;
    return r;
}

...
```

7

Qual é o valor de `v` no final de cada um dos seguintes fragmentos.

7.1

```
int v = 1;
```

```
int& r = v;
r = 2;
```

7.2

```
int v = 2;
int& r = v;
int* p = &v;
r = *p + r;
r++;
```

7.3

```
int v = 3;
int v2 = 4;
int& r = v;
int& r2 = v2;
r = r2;
r2 = r;
r++;
```

7.4

```
int v = 3;
int a[2] = { 1, 2 };
int& r = v;
int& r2 = a[1];
r2++;
r += a[0] + a[1];
```

8

Considere cada um dos seguintes fragmentos de código inseridos num programa que fazem uso dos operadores `new` e `delete`.

Identifique os problemas que possam existir no uso de memória dinâmica (ex. "leaks", "use-after-free", ou "buffer overflows"). Pode ajudar averiguar o output dos sanitisador ASan em cada caso.

8.1

```
int* p = new int[3] { 0 };
p[1] = p[0] + p[2];
```

8.2

```
int* p = new int[3] { 0 };
p[1] = p[0] + p[3];
delete [] p;
```

8.3

```
int* p = new int[3] { 0 };
delete [] p;
p[1] = p[0] + p[2];
```

8.4

```
int* p = new int[3] { 0 };
int* q = p + 1;
*p = 1;
delete [] p;
*q= 2;
```

8.5

```
int* p = new int[3] { 0 };
int* q = new int;
*q = p[2];
*p = *q;
delete [] p;
```

8.6

```
int* p = new int[3] { 0 };
int* q = new int;
*q = 3;
p[*q] = 1;
delete [] p;
delete q;
```

9

Considere o seguinte programa:

```
#include <iostream>
... // definição de f
int main(void) {
    std::cout << f(10) << std::endl;
    return 0;
}
```

e a definição de `f` de acordo com um dos seguintes dois fragmentos. Que valor é impresso - note que `main` chama `f(10)` ? Interprete o fluxo de execução nos dois casos.

9.1

```
namespace a {
    int f(int x, int y=1) {
        return x * y;
    }
    namespace b {
        int f(int x) {
            return x + a::f(x-1);
        }
    }
}
int f() {
    return 100;
}
int f(int x) {
    return f() - a::f(x, 3) - a::b::f(x) - 9;
}
```

9.2

```
namespace a {
    int f(int x=2, int y=1) {
        return x - y;
    }
    namespace b {
        int f(int x) {
```

```
            return x > 2 ?
              x * f(x - 1) :
              a::f(x) * a::f(x * x, x);
        }
    }
}
int f(int x) {
    return  2 * a::b::f(4) - 6 * a::f();
}
```