Kỹ Thuật Lập Trình

Tài liệu qua vòng giữ xe trước khi vào BTL

Lý Thuyết Bài Tập Lớn

SHERLOCK A STUDY IN PINK - Phần 2

Thảo luận BTL môn KTLT, DSA, NMLT, PPL ....
https://www.facebook.com/groups/211867931379013

Tp. Hồ Chí Minh, Tháng 3/2024

Mục lục

1 Các kiên thức cần có

1. **Operations** Phần này các bạn tự tìm hiểu nha nằm ở phần BTL trước rồi
2. **Control Flow - If** Phần này các bạn tự tìm hiểu nha nằm ở phần BTL trước rồi
3. **Control Flow - Loop** Phần này các bạn tự tìm hiểu nha nằm ở phần BTL trước rồi
4. **String** Phần này các bạn tự tìm hiểu nha nằm ở phần BTL trước rồi
5. **Enum**
6. **Array**
7. **Pointer**
8. **Structure** Phần này không dùng trong BTL này các bạn cũng có thể tìm hiểu và ứng dụng nó trong BTL này
9. **Function**
10. **xử lí file với thư viện ifstream**
11. **Class**
12. **Linked list**

2 Enum trong c++

KTLT2: Enum

Enum trong C++ là một kiểu dữ liệu do **người dùng định nghĩa**, cho phép bạn tạo ra một **tập hợp** các giá trị hằng số nguyên được đặt tên. Điều này giúp cho việc lập trình trở nên rõ ràng và dễ quản lý hơn khi bạn cần một tập hợp các giá trị cố định, như các ngày trong tuần, các mùa trong năm, hoặc các trạng thái của một máy trạng thái.

```
enum Color {
    RED,
    GREEN,
    BLUE
};
```

Trong ví dụ trên, Color là tên của kiểu dữ liệu enum và RED, GREEN, BLUE là các giá trị của nó. Mặc định, giá trị đầu tiên RED sẽ có giá trị là 0, GREEN sẽ là 1, và BLUE sẽ là 2. Bạn có thể gán giá trị cụ thể cho các giá trị enum nếu muốn:

```
enum Color {
    RED = 1,
    GREEN = 2,
    BLUE = 4
};
```

Cách sử dụng **enum**:

```
Color color = Color::RED;
Color color = RED;
Color color = Color(0)
```

Bài tập tự tìm hiểu:

Đọc nội dung code sau để trả lời các câu hỏi 1,2

```
enum Color {
    RED,
    GREEN,
    BLUE
};
```

1. Trong khai báo **enum** sau, giá trị của **RED** là bao nhiêu?

   a) 0 b) 1 c) -1 d) Không xác định

   solution : TODO

2. Trong khai báo **enum** sau, giá trị của **GREEN** là bao nhiêu?

   a) 1 b) 2 c) 3 d) 4

   solution : TODO

3. Trong khai báo **enum** sau, giá trị của **BLUE** là bao nhiêu?

```
enum Color {
    RED = 3,
    GREEN,
    BLUE
};
```

a) 1 b) 5 c) 3 d) 4

solution : TODO

4. Trong khai báo **enum** sau, kết quả đoạn code sau

```
enum ElementType {PATH = 2, WALL, FAKE_WALL = 6};
int main()
{
    std::cout << int(PATH) << " "
              << int(ElementType::FAKE_WALL - ElementType::WALL);
}
```

a) 2 3 b) 2 2 c) 1 2 d) 1 3

solution : TODO

5. Trong khai báo **enum** sau, kết quả *result*

```
enum ElementType {PATH = 2, WALL, FAKE_WALL = 6};
int main()
{
    ElementType result = ElementType(3);
}
```

a) WALL b) PATH c) FAKE_WALL d) UNKNOWN

solution : TODO

6. Trong khai báo **enum** sau, kết quả

```
enum ElementType {PATH = 2, WALL, FAKE_WALL = 6};

std::string elementTypeName(ElementType element) {
    switch (element) {
        case PATH: return "PATH";
        case WALL: return "WALL";
        case FAKE_WALL: return "FAKE_WALL";
        default: return "UNKNOWN";
    }
}

int main()
{
    ElementType result = ElementType(4); // Không hợp lệ, sẽ trả về "UNKNOWN"

    std::cout << elementTypeName(result);
    return 0;
}
```

a) WALL b) PATH c) FAKE_WALL d) UNKNOWN

solution : TODO

7. Trong khai báo **enum** sau, kết quả

```
enum RobotType { C=0, S, W, SW };
int main()
{
    int RobotHP[] = {2, 4, 5, 7};


    std :: cout << RobotHP[C] << " " << RobotHP[SW];
    return 0;
}

```

a) 2 7 b) 7 2 c) Lỗi biên dịch d) Lỗi thực thi

solution : TODO

8. Trong khai báo **enum** sau, kết quả

```
#include <iostream>

enum RobotType { C=0, S, W, SW };

int main() {
   RobotType Type = C;


    std::cout << (Type == RobotType(C));

    return 0;
}


```

a) 1 b) 2 c) -1 d) 0

solution : TODO

9. Trong khai báo **enum** sau, kết quả

```
#include <iostream>

enum RobotType { C=3, S, W, SW };

int main() {
   RobotType Type = C;


    std::cout << (Type == RobotType(C));

    return 0;
}


```

a) 1 b) 2 c) -1 d) 0

solution : TODO

10. Trong khai báo **enum** sai ở dòng nào

```
1  #include <iostream>
2
3  enum RobotType { C = 0, S , W , SW };
4  enum ItemType { MAGIC_BOOK = 0, ENERGY_DRINK, FIRST_AID,
   ↪  EXCEMPTION_CARD,PASSING_CARD };
5  int main() {
6
7
8
9      std::cout << ((RobotType::C) == (ItemType::MAGIC_BOOK));
10
11     return 0;
12 }
13
14
```

a) 2 b) 3 c) 9 d) 4

solution : TODO

11. Trong khai báo **enum** sai hàng nào

```
1  #include <iostream>
2
3  enum RobotType { C = 0, S, W, SW };
4
5  void printRobotType(RobotType type) {
6      switch(type) {
7          case C:
8              std::cout << "C";
9              break;
10         case S:
11             std::cout << "S";
12             break;
13         case W:
14             std::cout << "W";
15             break;
16         case SW:
17             std::cout << "SW";
18             break;
19         default:
20             std::cout << "!!!";
21     }
22 }
23
24 int main() {
25     RobotType robot = SW;
26     printRobotType(10);
27
28     return 0;
29 }
```

```
30
31
```

a) 3 b) 25 c) 26 d) không có lỗi

solution : TODO

12. Trong khai báo **enum** in ra gì

```
1  #include <iostream>
2
3  enum RobotType { C = 0, S, W, SW };
4
5  void printRobotType(RobotType type) {
6      switch(type) {
7          case C:
8              std::cout << "C";
9              break;
10         case S:
11             std::cout << "S";
12             break;
13         case W:
14             std::cout << "W";
15             break;
16         case SW:
17             std::cout << "SW";
18             break;
19         default:
20             std::cout << "!!!";
21     }
22 }
23
24 int main() {
25     RobotType robot = SW;
26     printRobotType(RobotType(10));
27
28     return 0;
29 }
30
31
```

a) !!
b) SW c) w d) S

solution : TODO

3 Array

KTLT2: Array

Array là một cấu trúc dữ liệu cơ bản cho phép lưu trữ một tập hợp cố định **các phần tử có cùng kiểu dữ liệu**. Mảng được sử dụng để lưu trữ dữ liệu một cách **liên tục trong bộ nhớ**, và mỗi phần tử trong mảng có thể được truy cập thông qua chỉ số của nó.

- **Khai báo mảng:** Để khai báo một mảng, bạn cần xác định kiểu dữ liệu của các phần tử và số lượng phần tử mà mảng có thể chứa.

```
int myArray[10]; // Mảng của 10 số nguyên
int myArray[N]; // N không phải const thì không cho phép sai
```

- **Khởi tạo mảng:** Bạn có thể khởi tạo mảng với một danh sách các giá trị cụ thể hoặc để trình biên dịch tự động gán giá trị.

```
int myArray[5] = {1, 2, 3, 4, 5}; // Khởi tạo mảng với giá trị cụ thể
int myArray[] = {1, 2, 3, 4, 5}; // Kích thước mảng được xác định bởi số
↪ lượng giá trị
```

- **Truy cập phần tử mảng:** Bạn có thể truy cập hoặc thay đổi giá trị của một phần tử trong mảng thông qua chỉ số của nó.

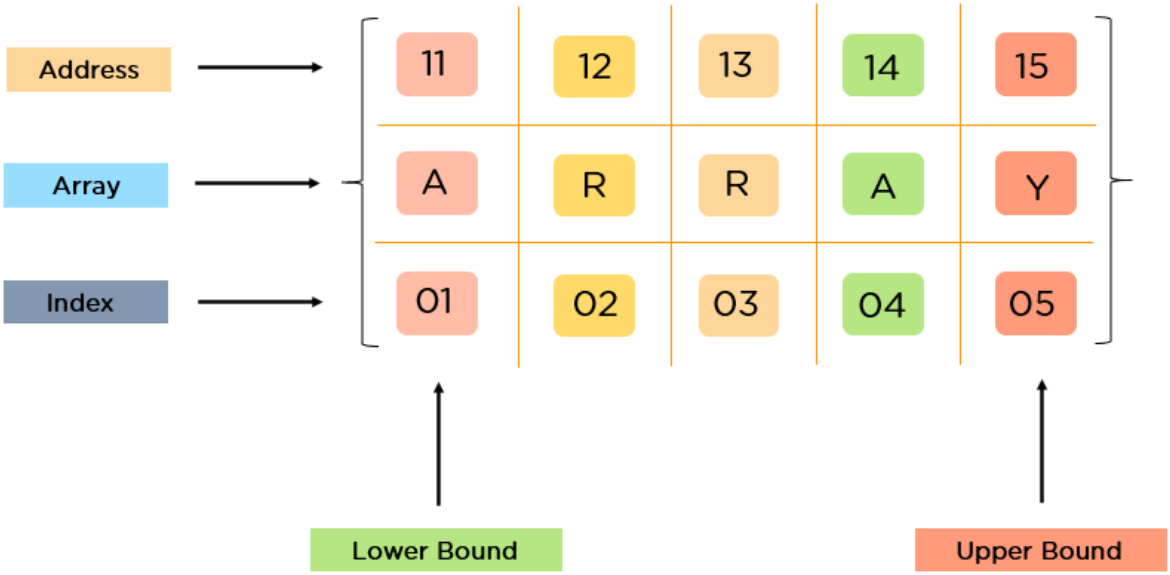
```
int value = myArray[2]; // Truy cập phần tử thứ ba trong mảng
myArray[0] = 9; // Thay đổi giá trị của phần tử đầu tiên trong mảng
```

- **Kích thước mảng:** Kích thước của mảng không thể thay đổi sau khi nó đã được khai báo. Để xác định kích thước của mảng, bạn có thể sử dụng toán tử sizeof.

```
// Tính số lượng phần tử trong mảng
int size = sizeof(myArray) / sizeof(myArray[0]);
```

- **Mảng đa chiều:** C++ cũng hỗ trợ mảng đa chiều, thường được sử dụng để biểu diễn bảng, ma trận, hoặc thông tin không gian nhiều hơn.

```
int matrix[3][4]; // Mảng hai chiều với 3 hàng và 4 cột
```



- Mảng có 5 phần tử, mỗi phần tử chứa một ký tự từ chữ "ARRAY".
- Mỗi phần tử được biểu diễn với ba thuộc tính: Địa chỉ, Giá trị Mảng và Chỉ số.
- Các địa chỉ được ghi từ 11 đến 15.
- Giá trị mảng được ghi với các chữ cái A, R, R, A, Y tương ứng.
- Các chỉ số được ghi từ 01 đến 05.
- Có chỉ dẫn về Giới hạn Dưới tại chỉ số 01 và Giới hạn Trên tại chỉ số 05. Này không giống trong c++ là bắt đầu từ 00

Một số cách truy cập phần tử array xem ở phần pointer

Bài tập tự tìm hiểu:

1. Kích thước của một mảng trong C++ phải được xác định như thế nào?

   a) Tại thời điểm chạy
   b) Tại thời điểm biên dịch
   c) Bất kỳ lúc nào
   d) Không cần xác định

   solution : TODO

2. Làm thế nào để truy cập phần tử đầu tiên của mảng myArray?

   a) myArray[0] b) myArray[1] c) myArray.first d) myArray.begin()

   solution : TODO

3. Kết quả output?

```
int myNumbers[5] = {2, 4, 6, 8, 10};
cout << myNumbers[3];
```

   a) 2 b) 4 c) 8 d) 10

   solution : TODO

4. Kết quả output?

```
int myNumbers[5] = {2};
cout << myNumbers[2];
```

a) 2 b) 4 c) lỗi d) 0

solution : TODO

5. Điều gì sẽ xảy ra nếu bạn cố gắng truy cập một chỉ số ngoài giới hạn của mảng?

   a) Chương trình sẽ tự động thêm phần tử mới
   b) Chương trình sẽ báo lỗi tại thời điểm biên dịch
   c) Chương trình có thể gây ra lỗi thời gian chạy
   d) Chương trình sẽ trả về giá trị mặc định

   solution : TODO

6. Một mảng hai chiều trong C++ có thể được hiểu như thế nào?

   a) Một mảng của các mảng
   b) Một mảng với hai phần tử
   c) Một mảng lưu trữ các số nguyên và số thực
   d) Một mảng không thể chứa mảng khác

   solution : TODO

7. Làm thế nào để khai báo một mảng hai chiều 3x4 (3 hàng và 4 cột) trong C++?

   a) int myArray[3, 4];
   b) int myArray[3][4];
   c) int[3][4] myArray;
   d) int myArray(3)(4);

   solution : TODO

8. Khi truy cập một phần tử trong mảng hai chiều arr, arr[2][1] sẽ trả về phần tử nào?

   a) Phần tử ở hàng thứ 2 và cột thứ 1
   b) Phần tử ở hàng thứ 1 và cột thứ 2
   c) Phần tử ở hàng thứ 3 và cột thứ 2
   d) Phần tử ở hàng thứ 2 và cột thứ 3

   solution : TODO

9. Làm thế nào để khai báo một mảng hai chiều 3x4 (3 hàng và 4 cột) trong C++?

   a) int myArray[3, 4];
   b) int myArray[3][4];
   c) int[3][4] myArray;
   d) int myArray(3)(4);

   solution : TODO

10. Trong C++, làm thế nào để khai báo một mảng có kích thước cố định?

    a) int arr[]
    b) int arr[10]
    c) int[] arr
    d) int arr()

    solution : TODO

11. Trong C++, làm thế nào để khai báo một mảng có kích thước cố định?

```
    #include <iostream>

int main() {
    int arr[] = {1, 2, 3, 4, 5};
    int size = sizeof(arr) / sizeof(arr[0]);

    std::cout << "Kich thuoc cua mang: " << size << std::endl;

```

```
    return 0;

```

a) 0
b) 2
c) 7
d) 5

solution : TODO

12. int arr[5] = 1,2,3,4,5; câu nào lấy giá trị bị sai

a) arr[0]
b) arr[5]
c) arr[2]
d) arr[3]

solution : TODO

13. int arr[5] = 1,2,3,4,5; câu nào lấy giá trị bị sai

a) arr[0]
b) arr[5]
c) arr[2]
d) arr[3]

solution : TODO

14. màng hình in ra được gì

```
    #include <iostream>

int main() {
    int arr[] = {1, 2, 3, 4, 5};
    int size = sizeof(arr) / sizeof(arr[0]);

    std::cout <<*(arr + 3) << std::endl;

    return 0;
}

```

a) 4
b) 2
c) 0
d) 1

solution : TODO

15. màng hình in ra được gì ?

```
 #include <iostream>

int main() {

    int arr[3][4] = {{1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12}};


    std::cout <<sizeof(arr)<< std::endl;

    return 0;

}

```

a) 48
b) 36
c) 12
d) 4

solution : TODO

16. khai báo nào bị sai?

a) int arr[][];
b) int arr[2][];
c) int arr[3][4];
d) Không có đáp án sai

solution : TODO

17. Trong mảng hai chiều kiểu int arr có kích thước 4x4, giá trị của sizeof(arr) sẽ là bao nhiêu?

a) 16;
b) 32;
c) 48;
d) 64

solution : TODO

4 Pointer

KTLT2: Pointer

Pointer là một biến có giá trị là **địa chỉ (vị trí nhà)** của một biến khác. Con trỏ cho phép chương trình thực hiện các thao tác như **truy cập trực tiếp vào bộ nhớ**, mô phỏng tham chiếu khi gọi hàm, và tạo cũng như quản lý các cấu trúc dữ liệu động

ý tưởng thực tế: Võ Tiến muốn miêu tả về trường Bách Khoa cho học sinh miền núi biết Bách Khoa đẹp như thế nào, bây giờ có 2 cách là xây một trường Bách Khoa giống vậy ngay tại chỗ mấy em học sinh miền núi, cách còn lại là đưa địa chỉ của trường Bách Khoa tại 268 Lý Thường Kiệt **Ta thấy cách 1 khá vô lý vì tiền học lại của sinh viên quá ít không đủ tiền xây giống vậy nên cách 2 là đưa địa chỉ để học sinh miền núi tới trường sẽ biết rõ Bách Khoa đẹp thế nào từ đó chúng ta được lý thuyết về con trỏ**

ý tưởng trong lập trình: có một dữ liệu vô cùng lớn chúng ta không có khả năng sao chép mà muốn đọc nó thì chúng ta cần địa chỉ nó.

```
int var = 20;      // biến thông thường
int *ptr;          // khai báo con trỏ
ptr = &var;        // con trỏ ptr lưu trữ địa chỉ của var
```

4.1 Cách dùng cơ bản

https://skills.microchip.com/fundamentals-of-the-c-programming-language-part-iii/700292

- **Khai báo con trỏ:** khai báo một con trỏ bằng cách sử dụng dấu * sau kiểu dữ liệu.

```
int* ptr; // ptr là con trỏ cho kiểu int
```

- **Gán địa chỉ cho con trỏ:** có thể gán địa chỉ của một biến cho một con trỏ bằng cách sử dụng toán tử &.

```
int var = 5; // ví dụ địa chỉ 0x09 nơi lưu trữ 5
ptr = &var; // ptr = 0x09
```

- **Truy cập giá trị thông qua con trỏ:** có thể truy cập hoặc thay đổi giá trị của biến mà con trỏ trỏ tới bằng cách sử dụng toán tử *.

```
// đến địa chỉ ptr = 0x09 thay đổi giá trị tại đó từ 5 thành 10
*ptr = 10; // Thay đổi giá trị của var thành 10
```

- **Con trỏ trỏ đến con trỏ:** C++ cho phép sử dụng con trỏ trỏ đến con trỏ, tạo ra nhiều cấp độ gián tiếp.

```
int** ptrptr = &ptr; // ptrptr là con trỏ trỏ đến con trỏ địa chỉ của ptr
**ptrptr = 1; // đến địa chỉ của ptr và tiếp tục đến địa chỉ var sau đó thay
↪ đổi giá trị thành 1
```

4.2 con trỏ NULL

- **Giá trị NULL:** Trước C++11, NULL được định nghĩa là 0 hoặc ((void*)0), và sau đó nullptr được giới thiệu như một giá trị null chuẩn cho con trỏ.
- **Khởi tạo:** Con trỏ nên được khởi tạo với giá trị NULL nếu nó không được gán với địa chỉ của một đối tượng hợp lệ ngay lập tức.

```
int* ptr = NULL; // Trước C++11
int* ptr = nullptr; // Từ C++11 trở đi
```

- **Kiểm tra con trỏ NULL:** có thể kiểm tra xem một con trỏ có phải là con trỏ NULL hay không bằng cách so sánh nó với NULL hoặc nullptr.

```
if (ptr == nullptr) {
    // Con trỏ là NULL
}
if (!ptr) {
    // Con trỏ là NULL
}
```

- **Dereferencing:** Việc dereference một con trỏ NULL (truy cập giá trị mà nó trỏ đến) sẽ dẫn đến hành vi không xác định và thường gây ra lỗi chương trình

4.3 Cách Dùng array

- **Khai báo con trỏ:** khai báo một con trỏ bằng cách sử dụng dấu * sau kiểu dữ liệu.

```
int* ptr; // ptr là con trỏ cho kiểu int
```

- **Truy cập phần tử mảng:** có thể sử dụng con trỏ để truy cập các phần tử của mảng. Tên mảng là một con trỏ hằng trỏ đến phần tử đầu tiên của mảng. biến arr cũng là địa chỉ đầu tiên của phần tử trong mảng

```
int arr[5] = {1, 2, 3, 4, 5};
int* ptr = arr; // ptr trỏ đến phần tử đầu tiên của arr = &arr[0]
```

- **Duyệt mảng:** có thể dùng con trỏ để duyệt qua mảng. sử dụng phép cộng trong địa chỉ là (ptr + N) sẽ dịch lên địa chỉ mới (ptr + N * size) là địa chỉ hiện tại

```
for(int i = 0; i < 5; ++i) {
    std::cout << *(ptr + i) << ' '; // In giá trị của mỗi phần tử mảng
}
// kết quả 1 2 3 4 5
```

- **Thay đổi giá trị phần tử:** có thể thay đổi giá trị của một phần tử trong mảng thông qua con trỏ.

```
*(ptr + 2) = 10; // Thay đổi giá trị của phần tử thứ ba thành 10
// kết quả arr[2] = 10
```

- **Arithmetic con trỏ:** C++ cho phép thực hiện các phép toán trên con trỏ, như cộng hoặc trừ một số nguyên, để di chuyển con trỏ qua mảng.

```
ptr++; // Di chuyển con trỏ đến phần tử tiếp theo trong mảng
// kết quả ptr = &arr[1]
ptr--; // lùi lại
// kết quả ptr = &arr[0]
```

4.4 Cấp phát động

KTLT2: Cấp phát động

Cấp phát động (Dynamic memory) trong C++ là một quá trình mà bộ nhớ được cấp phát tại **thời điểm chạy (runtime)** thay vì tại thời điểm biên dịch (compile-time). Điều này cho phép chương trình của bạn **yêu cầu bộ nhớ khi cần thiết**, thay vì phải xác định trước kích thước của cấu trúc dữ liệu như mảng hoặc các đối tượng khác

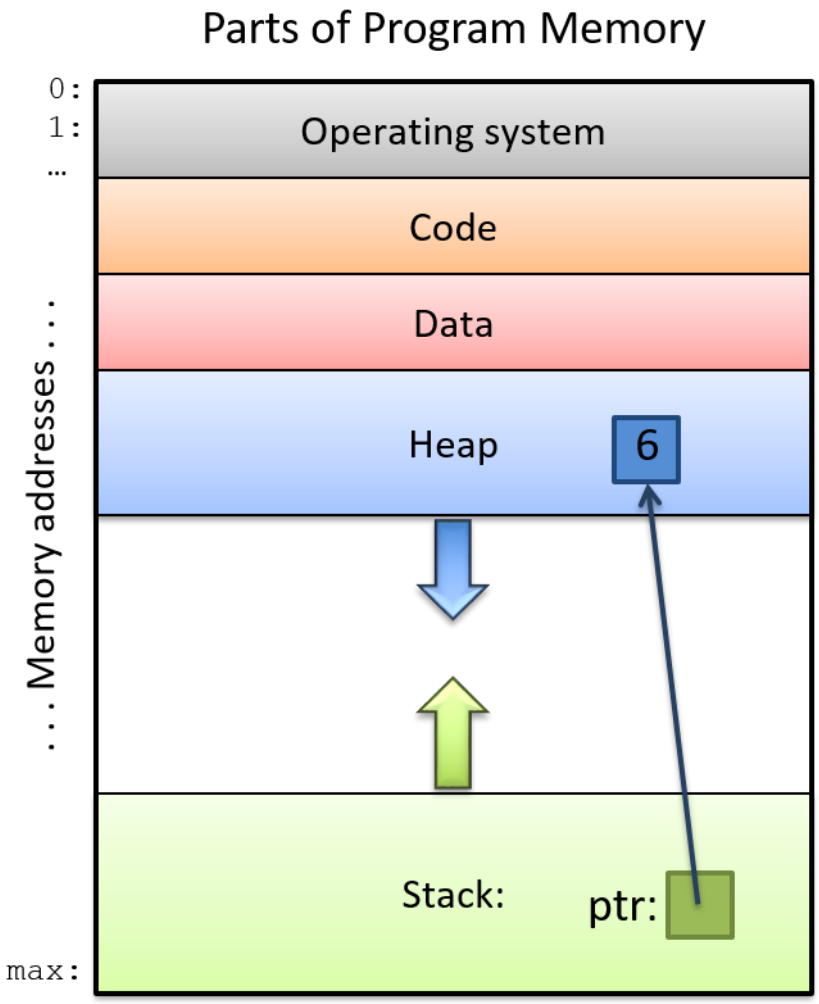
```
int* ptr = new int; // cấp phát bộ nhớ cho một int
delete ptr; // giải phóng bộ nhớ
```

Stack Là khu vực bộ nhớ được quản lý theo **cơ chế LIFO (Last In, First Out)**, nơi cấp phát và giải phóng bộ nhớ tự động theo thứ tự ngược lại với thứ tự cấp phát. Stack thường được sử dụng để **lưu trữ các biến cục bộ của hàm và thông tin điều khiển của hàm**. Khi một hàm được gọi, một khối bộ nhớ được cấp phát trên stack để lưu trữ các biến cục bộ và khi hàm kết thúc, bộ nhớ này được giải phóng

Heap Là khu vực bộ nhớ được sử dụng cho việc **cấp phát động**, nơi mà các nhà phát triển có thể yêu cầu cấp phát hoặc giải phóng bộ nhớ một cách rõ ràng thông qua các lệnh lập trình new. **Heap không có cơ chế tự động giải phóng như stack** và thường được sử dụng cho việc cấp phát bộ nhớ có thời gian sống dài hoặc không xác định, như đối tượng hoặc mảng động

```
int* Variable; // Biến cục bộ được cấp phát trên stack
Variable = new int(10); // new int(10) Cấp phát động trên heap
// Sử dụng Variable
delete Variable; // Giải phóng bộ nhớ khi không cần sử dụng nữa
```

https://www.scaler.com/topics/c/dynamic-memory-allocation-in-c/



- **Toán tử new và delete** Được sử dụng để cấp phát và giải phóng bộ nhớ động.

```
int* ptr = new int; // Cấp phát bộ nhớ cho một int
delete ptr; // Giải phóng bộ nhớ
```

- **Cấp phát cho mảng:** cấp phát bộ nhớ cho mảng động và sau đó giải phóng nó

```
int* arr = new int[10]; // Cấp phát bộ nhớ cho mảng 10 int
delete[] arr; // Giải phóng bộ nhớ mảng
```

- **Cấp phát cho mảng 2 chiều:** cấp phát bộ nhớ cho mảng động và sau đó giải phóng nó

```
int m = 3, n = 4; // m là số hàng, n là số cột
int** a = new int*[m]; // Cấp phát mảng của con trỏ

// Cấp phát từng hàng
for (int i = 0; i < m; i++) {
    a[i] = new int[n];
}
```

```
8
9  // Gán giá trị cho mảng
10 for (int i = 0; i < m; i++) {
11     for (int j = 0; j < n; j++) {
12         a[i][j] = giá_trị;
13     }
14 }
15
16 // Giải phóng từng hàng
17 for (int i = 0; i < m; i++) {
18     delete[] a[i];
19 }
20
21 // Giải phóng mảng của con trỏ
22 delete[] a;
```

- **Dangling Reference:** Là tình huống mà một tham chiếu vẫn trỏ đến một vùng nhớ đã được giải phóng hoặc không còn hợp lệ. Điều này thường xảy ra khi bạn trả về một tham chiếu đến một biến cục bộ từ một hàm, và sau đó biến đó ra khỏi phạm vi và bị hủy

```
1  p = new int;
2  q = p;
3  delete p;
4  cout << *q;
5
6  int& func() {
7      int x = 10;
8      return x; // Dangling reference vì x bị hủy khi func kết thúc
9  }
```

- **Memory Leak:** Xảy ra khi bộ nhớ được cấp phát động thông qua new nhưng không được giải phóng bằng delete

```
1  p = new int;
2  p = null;
3
4  void createLeak() {
5      int* ptr = new int(5); // Cấp phát động nhưng không giải phóng
6  }
```

- **Object lifetime:** là thời gian giữa 2 lần tạo ra (creation) và phá hủy (destruction), đối với stack thì Object lifetime nằm trong vùng cục bộ của biến đó, còn đối với khai báo head là từ khi new đến khi delete

4.5 References

KTLT2: References

References là các biến tham chiếu đến một biến khác, cho phép bạn sử dụng chúng như là một tên gọi khác cho biến đó. Khi một biến được khai báo là một reference, nó trở thành một tên gọi thay thế cho một biến đã tồn tại

```
// Cú pháp để khai báo một reference trong C++ là:
data_type &ref = variable;
```

ý tưởng Võ Tiến còn một tên nữa là Tiến đẹp trai bây giờ Tiến đẹp trai được 10. KTLT thì cũng có nghĩa là Võ Tiến cũng 10. KTLT vì chung 1 người, bây giờ các bạn không gọi là Tiến đẹp trai mà gọi tên khác nhưng thực tế thì cũng là Võ Tiến nếu thay đổi trên có bí danh (Alias) đó thì Võ Tiến cũng bị thay đổi

```
int x = 10;
int& ref = x; // ref là một reference cho x
ref = 20; // Thay đổi giá trị của x thông qua ref
cout << "x = " << x << '\n'; // x = 20
```

https://stackoverflow.com/questions/71457868/what-is-an-alias-in-c

Bài tập tự tìm hiểu:

Nhóm câu hỏi truy xuất giá trị và địa chỉ con trỏ

```
int main()
{
    int item = 5;
    int *smallBox = &item;
    int **box = &smallBox;
    int ***bigBox = &box;
    return 0;
}
```

1. Giá trị của ***bigBox**
   a) Giá trị của **item**
   b) Báo lỗi
   c) Địa chỉ của **smallBox**
   d) Giá trị của **smallBox**

   solution : TODO

2. Giá trị của biểu thức ***box == &smallBox**
   a) true
   b) Báo lỗi
   c) false
   d) Đáp án khác

   solution : TODO

3. Biểu thức nào sau đây có thể truy cập giá trị của **item**
   a) ****box**
   b) **smallBox**
   c) **&bigBox**
   d) ***box**

solution : TODO

Nhóm câu hỏi cấp phát và giải phóng bộ nhớ bằng con trỏ

4. Đoạn code bên dưới sai ở hàng nào

```
1  #include <iostream>
2
3  int main() {
4      int * Position = new int(5);
5      delete Position;
6      delete Position;
7      return 0;
8  }
9
10
```

a) Dòng 4
b) Dòng 5
c) Dòng 6
d) Dòng 7

solution : TODO

5. Đoạn code bên dưới sai ở hàng nào

```
1  #include <iostream>
2
3  int main() {
4      int * Position = new int(5);
5      int * temp = Position;
6      *temp = 4;
7      std::cout<<*Position<<" " <<*temp<<std::endl;
8      delete Position;
9      return 0;
10 }
11
12
```

a) Dòng 5 4
b) Dòng 4 5
c) Dòng 5 5
d) Dòng 4 4

solution : TODO

Nhóm câu hỏi vận dụng bài tập lớn

```
1  int main() {
2      // Khai báo số hàng, số cột của bản đồ
3      int soHang = 5;
4      int soCot = 3;
5
6      // Khai báo con trỏ biểu diễn bản đồ (mảng 2 chiều)
7      int*** map;
8
9      // Cấp phát động cho map một mảng các con trỏ dùng để biểu diễn các hàng
       ↪ trong bản đồ
10     map = new int**[soHang];
11
```

```
    // Cấp phát động cho mỗi hàng một mảng các con trỏ dùng để biểu diễn các ô
    ↪ trong hàng
    for (int i = 0; i < soHang; ++i) {
        map[i] = new int*[soCot];
    }

    // Cấp phát động cho mỗi ô một vùng nhớ biểu diễn biến kiểu số nguyên
    for (int i = 0; i < soHang; ++i) {
        for (int j = 0; j < soCot; ++j) {
            map[i][j] = new int;
        }
    }

    return 0;
}
```

6. Biểu thức **map[2][4]** sẽ truy xuất tới?

   a) Giá trị của ô có vị trí [2][4]
   b) Địa chỉ của hàng có index 2
   c) Báo lỗi
   d) Địa chỉ của ô có vị trí [2][4]

   solution : TODO

7. Giá trị của một hàng trong bản đồ chính là?

   a) Giá trị của vùng nhớ kiểu số nguyên
   b) Địa chỉ của vùng nhớ kiểu số nguyên
   c) Địa chỉ của ô đầu tiên trong hàng
   d) Giá trị của ô đầu tiên trong hàng

   solution : TODO

8. Biểu thức **map[1]** sẽ truy xuất tới?

   a) Địa chỉ của hàng thứ 2
   b) Địa chỉ của ô có vị trí [0][2]
   c) Báo lỗi
   d) Giá trị của hàng thứ 2

   solution : TODO

9. Biểu thức ***map[2]** sẽ truy xuất tới?

   a) Giá trị của ô có vị trí [2][0]
   b) Địa chỉ của ô có vị trí [0][2]
   c) Báo lỗi
   d) Địa chỉ của ô có vị trí [2][0]

   solution : TODO

10. Biểu thức lấy giá tri của ô có thứ 4 trong hàng thứ 3 là?

    a) ***map[3][4]**
    b) ***(*(map+3) + 4)**
    c) ***map[2][3]**
    d) ***(map[2] + 4)**

    solution : TODO

Nhóm câu hỏi tổng hợp: gán, cấp phát, vùng nhớ được trỏ bởi nhiều con trỏ, thu hồi

```
int main() {
    int **Sherlock, **Watson;
    int *money, *f88 = new int(-999);

    // Line1
    <Đáp án câu 11>

    // Line2
```

```
    <Đáp án câu 12>

    // Line3
    Watson = Sherlock;

    // Line4
    <Đáp án câu 13>

    // Chuỗi sự kiện khiến Watson bị robot đấm và hi sinh ...

    // Line5
    <Đáp án câu 14>

}
```

11. Hãy chọn câu lệnh phù hợp vào vị trí **Line1** giúp Sherlock truy xuất đến **money**

    a) Sherlock = &money;
    b) *Sherlock = money;
    c) Sherlock = money;
    d) Sherlock = *money;

12. Cơm áo gạo tiền! Hãy lựa chọn câu lệnh phù hợp vào vị trí **Line2** để cho Sherlock 100 đồng đi lập nghiệp

    a) Sherlock = new int(100);
    b) *money = 100;
    c) *Sherlock = new int*(new int(100));
    d) *Sherlock = new int(100);

13. Watson láo vô cùng khi muốn dùng ké tiền Sherlock **(Line3)**. Hãy lựa chọn câu lệnh phù hợp vào vị trí **Line4** để chỉ đưa Watson tới "chắp cánh ước mơ"

    a) *Watson = f88;
    b) Watson = *f88;
    c) Watson = &f88;
    d) **Watson = *f88;

14. Sau khi Watson hi sinh, hãy chọn câu lệnh phừ hợp vào vị trí **Line5** để thu hồi tiền của Watson

    a) delete Watson;
    b) delete *Watson;
    c) delete **Watson;
    d) delete *f88;

5 Function

KTLT2: Function

Tại sao phải sử dụng hàm?

- Không nên viết code theo style 1 block for all:
  - Sai sót giữa các phần code
  - Lùm xùm, khó tìm lỗi, khó fix
  - Khi thêm, điều chỉnh chức năng phải chỉnh tất cả các phần code
- Dễ kiểm tra, xem lại
- Sử dụng lại phần code thuận tiện

Truyền tham số:

- Có 2 kiểu truyền tham số
  - Tham trị: giá trị của đối số sẽ được copy vào tham số của hàm
  - Tham chiếu: tham số sẽ liên hệ với đối số
    - Truyền tham chiếu nghĩa là truyền địa chỉ của đối số
    - Các thay đổi với tham số sẽ thay đổi đối số

Truyền tham số là mảng:

- Có 3 cách truyền tham chiếu với mảng

```
<kiểu dữ liệu trả về> <tên hàm> (<kiểu dữ liệu> *<tên mảng>)

<kiểu dữ liệu trả về> <tên hàm> (<kiểu dữ liệu> <tên mảng>[<kích thước
↪ mảng>])

<kiểu dữ liệu trả về> <tên hàm> (<kiểu dữ liệu> <tên mảng>[])
```

Tham số mặc định:

- Tham số mặc định dùng để thay thế cho các đối số bỏ trống khi gọi hàm
- Các tham số mặc định phải là các tham số ngoài cùng từ phải sang trái trong tập hợp các tham số

```
// Khai báo tham số mặc định sai
void setInfo(string name="", int age)
// Khai báo tham số mặc định đúng
void setInfo(int age, string name="")
```

Nạp chồng hàm:

- Các hàm có thể có cùng tên gọi hàm
- C++ compiler sẽ chọn hàm thực thi dựa trên số lượng, kiểu dữ liệu và thứ tự của đối số trong lời gọi

```cpp
float add(float a, float b) {
    return a + b;
}
int add(int a, int b) {
    return a + b;
}
double add(int a, double b) {
    return (double)a + b;
}
int add(int a, int b, int c) {
    return a+b+c;
}
```

Bài tập tự tìm hiểu:

1. Tham số mặc định nên được đặt ở đâu trong khai báo hàm?
   a) Ngoài cùng bên phải tập hợp các tham số
   b) Ngoài cùng bên trái tập hợp các tham số
   c) Bất cứ đâu trong tập hợp các tham số
   d) Ở giữa tập hợp các tham số
2. Nếu một đối số trong tập hợp các tham số được định nghĩa hằng số thì?
   a) Nó có thể được thay đổi bên trong hàm
   b) Nó không thể bị thay đổi bên trong hàm
   c) Báo lỗi
   d) Tùy thuộc vào định nghĩa của tham số được truyền vào
3. Kết quả của đoạn code sau

```cpp
void square (int *x, int *y)
{
    *x = (*x) * --(*y);
}
int main ( )
{
        int number = 30;
        square(&number, &number);
        cout << number;
        return 0;
}
```

   a) 870
   b) 30
   c) 900
   d) 841
4. Có bao nhiêu kiểu trả về dữ liệu của hàm trong c++?
   a) 1
   b) 2
   c) 3
   d) 4
5. Kết quả của đoạn code sau

```cpp
int &pass(int &x){
    x=0;
    return x;
}

```

```
6  int main()
7  {
8      int a=5;
9      pass(a)+=10;
10     cout<<a<<endl;
11 }
```

a) 5
b) 10
c) 15
d) Báo lỗi

6. Khi nào thì sử dụng nạp chồng hàm?

a) Cùng tên hàm nhưng khác số lượng đối số
b) Khác tên hàm nhưng cùng số lượng đối số
c) Cùng tên hàm và cùng số lượng đối số
d) Khác tên hàm và khác số lượng đối số

6 Ifstream và string

KTLT2: Ifstream

ifstream là một lớp được sử dụng để đọc dữ liệu từ file. ifstream viết tắt của "input file stream", nghĩa là luồng nhập file. Đây là một lớp con của istream và cho phép bạn tạo một luồng đầu vào liên kết với một file cụ thể, **Các Bước xử lí file**

1. Tạo một đối tượng *ifstream*.
2. Mở file bằng phương thức *open()* hoặc thông qua constructor.
3. Đọc dữ liệu từ file sử dụng các phương thức >>, nếu trong file đã tới cuối *eof*(End of file) thì >> sẽ trả về false
4. Đóng file bằng phương thức *close()*.

```
std::string str;
// bước 1 và bước 2
std::ifstream inputFile("example.txt");
// bước 3
while (inputFile >> str) {
    std::cout << str << std::endl;
}
// bước 4
inputFile.close();
```

KTLT2: String

string là một lớp được sử dụng để biểu diễn chuỗi ký tự. Đây là một phần quan trọng của thư viện chuẩn C++ và nó cung cấp nhiều chức năng hữu ích để làm việc với chuỗi ký tự
Các hàm cơ bản

- **length() hoặc size()** Trả về độ dài của chuỗi.
- **find()** Tìm kiếm vị trí xuất hiện đầu tiên của một chuỗi con.
- **substr()** Lấy một chuỗi con từ chuỗi hiện tại.
- **stoi()** (string to integer) chuyển đổi một chuỗi thành một số nguyên.
- **to_string()** chuyển đổi một số thành chuỗi

https://cplusplus.com/reference/string/string/

7 Lập trình hướng đối tượng (OOP)

7.1 giới thiệu

KTLT2: Lập trình cấu trúc (Structured Programming) và Lập trình hướng đối tượng (OOP)

Lập trình cấu trúc:

- Là một kỹ thuật được xem là tiền thân của OOP.
- Chia chương trình thành các mô-đun hoặc hàm rõ ràng và tách biệt.
- Tập trung vào việc tạo ra các chương trình có mã nguồn dễ đọc và các thành phần có thể tái sử dụng.
- Theo dõi logic của chương trình dễ dàng hơn thông qua việc giải quyết vấn đề liên quan đến các chuyển tiếp không điều kiện.
- Thường theo hướng tiếp cận "Top-Down".

Lập trình hướng đối tượng:

- Là một phương pháp khác biệt, kết hợp dữ liệu và các hàm thực thi trên chúng.
- Hỗ trợ đóng gói (encapsulation), trừu tượng hóa (abstraction), kế thừa (inheritance), đa hình (polymorphism).
- Tập trung vào việc mô phỏng các thực thể trong thế giới thực thành các đối tượng.
- Các đối tượng chứa cả dữ liệu và hàm, tương tác với nhau thông qua việc truyền thông điệp.
- Theo hướng tiếp cận "Bottom-Up".

So Sánh:

- Trong lập trình cấu trúc, các chương trình được chia thành các hàm nhỏ, trong khi OOP chia chương trình thành các đối tượng hoặc thực thể.
- Lập trình cấu trúc tập trung vào việc tạo ra các chương trình với mã nguồn dễ đọc, trong khi OOP tập trung vào việc tạo ra các đối tượng chứa cả hàm và dữ liệu.
- Lập trình cấu trúc thường theo dõi một cách tuần tự, còn OOP làm việc một cách động, gọi các phương thức theo nhu cầu của mã nguồn.
- Lập trình cấu trúc cung cấp ít tính linh hoạt và trừu tượng hơn so với OOP. Việc sửa đổi và tái sử dụng mã nguồn trong lập trình cấu trúc khó khăn hơn so với OOP

Limitations of Structure Programming :

- Thiếu đóng gói: Lập trình cấu trúc không hỗ trợ đóng gói, điều này làm giảm khả năng bảo mật và quản lý dữ liệu1.
- Thiếu ẩn thông tin: Không có khả năng ẩn đi các chi tiết triển khai bên trong, làm cho việc quản lý mã nguồn trở nên phức tạp hơn.
- Lặp lại mã nguồn: Có thể xuất hiện mã nguồn trùng lặp, làm tăng kích thước tổng thể của chương trình.
- Chương trình dài hơn cần thiết: Do cách tiếp cận tuyến tính, chương trình thường dài và phức tạp hơn so với các phương pháp lập trình khác.
- Sử dụng nhiều bộ nhớ hơn: Các chương trình cấu trúc có thể sử dụng nhiều bộ nhớ hơn và thực thi chậm hơn so với các chương trình không cấu trúc.
- Hạn chế về cấu trúc: Giới hạn các cấu trúc cơ bản đến ba hoặc bốn dạng, làm cho một số nhiệm vụ trở nên rối rắm khi thực hiện.

https://www.geeksforgeeks.org/difference-between-structured-programming-and-object-oriented-pro

7.2 Định nghĩa Class

KTLT2: Class

Class là một khái niệm cơ bản của lập trình hướng đối tượng. Một class là một kiểu dữ liệu **do người dùng định nghĩa**, nó chứa **các thành viên dữ liệu (data, fields, members, attributes, or properties)** và **các thành viên hàm (Functions or methods)** mà có thể được truy cập và sử dụng bằng cách tạo ra một thực thể (instance) của class

- **Attributes (thuộc tính)** trong một class đề cập đến các biến thành viên, nghĩa là các biến được định nghĩa bên trong một class. Các attributes này đại diện cho trạng thái hoặc đặc tính của đối tượng mà class đó mô tả

```
class Person {
public:
    string name; // Attribute
    int age;     // Attribute
private:
    double salary; // Attribute
};
```

- **methods (phương thức)** của một class là các hàm thành viên được định nghĩa bên trong class đó và được sử dụng để xác định hành vi của các đối tượng thuộc class. Các method có thể truy cập các attributes (thành viên dữ liệu) và các method khác của class

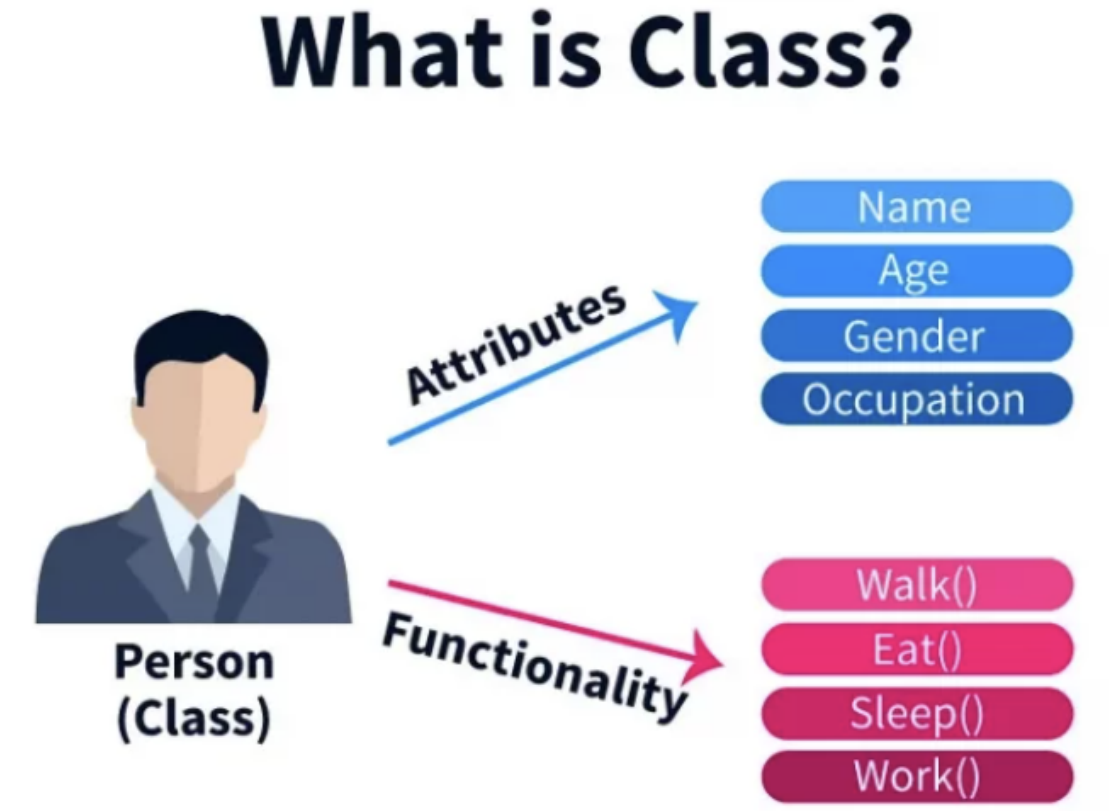
1. **Bên trong định nghĩa class:** method được định nghĩa ngay bên trong *class*.

```
class Dog {
public:
    void run() {
        cout << "run!";
    }
};
```

2. **Bên ngoài định nghĩa class:** method được khai báo bên trong *class* nhưng định nghĩa bên ngoài *class* sử dụng toán tử phân giải phạm vi ::

```
class Dog {
public:
    void run(); // Khai báo method
};

// Định nghĩa method bên ngoài class
void Dog::run() {
    cout << "run!";
}
```




```cpp
class Person {
public:
    std::string name;
    int age;
    std::string gender;
    std::string occupation;
    // Phương thức để mô phỏng hành động đi bộ
    void walk() {
        std::cout << name << " is walking." << std::endl;
    }

    // Phương thức để mô phỏng hành động ăn
    void eat() {
        std::cout << name << " is eating." << std::endl;
    }

    // Phương thức để mô phỏng hành động ngủ
    void sleep() {
        std::cout << name << " is sleeping." << std::endl;
    }

    // Phương thức để mô phỏng hành động làm việc
    void work() {
        std::cout << name << " is working." << std::endl;
    }
};
```

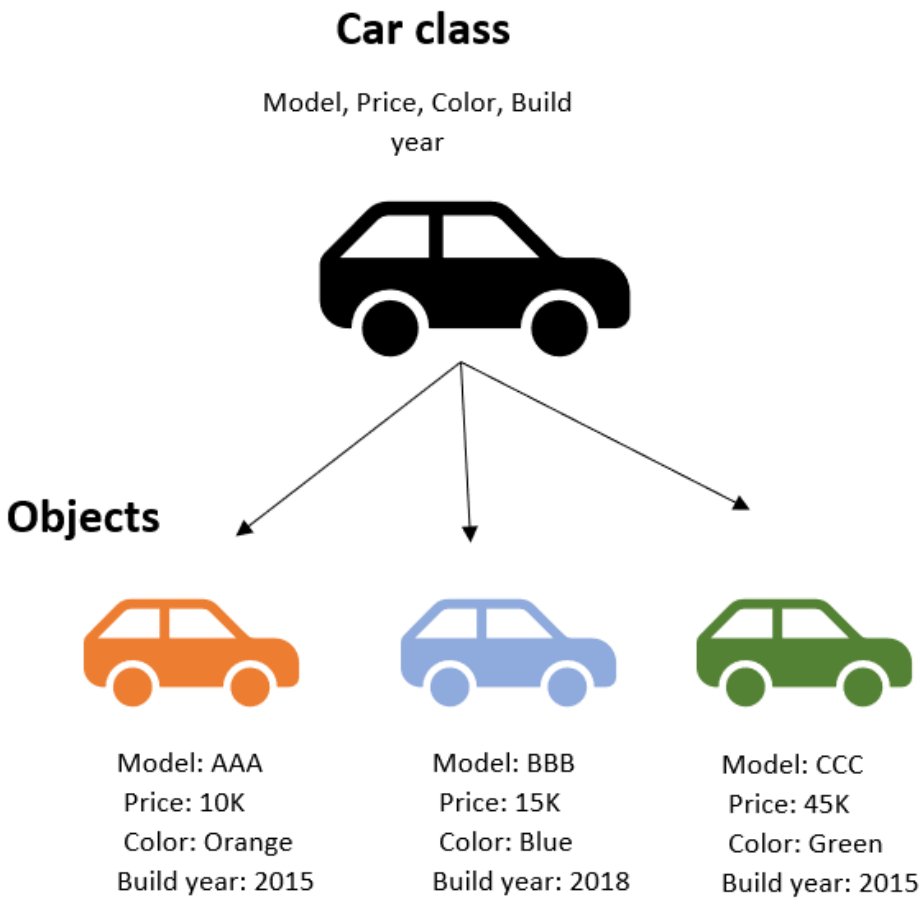
7.3 Objects

KTLT2: Objects

đối tượng(Objects) là một thực thể có thể được xác định với các đặc tính và hành vi. Đối tượng là một thể hiện cụ thể của một class, nơi class đóng vai trò như một bản thiết kế mô tả các thuộc tính (attributes) và phương thức (methods) mà đối tượng sẽ có. **Class giống như khung thiết kế còn Objects là sản phẩm khi dựa trên bản thiết kế đó nên 1 class có nhiều đối tượng**

```
class Car { // Định nghĩa class Car
public:
    std::string brand;
    std::string model;
    int year;

    void run();  // Phương thức chạy Car
};
Car car1;   // Tạo đối tượng car1 từ class Car
```



7.4 Contructor

KTLT2: Contructor

Constructor là một phương thức đặc biệt của class được gọi tự động khi một **đối tượng được tạo ra**. Mục đích chính của Constructor là **khởi tạo các thuộc tính (attributes)** của đối tượng với giá trị cụ thể

Cấu trúc <Tên class>(List Parameterized) { Body }

- **Default Constructor:** Là Constructor không có tham số. Nếu không có Constructor nào được định nghĩa trong class, C++ sẽ tự động tạo ra một Default Constructor
- **Parameterized Constructor:** Là Constructor có tham số, cho phép truyền giá trị cụ thể khi tạo đối tượng, Tham số mặc định
- **Copy Constructor:** Là Constructor dùng để tạo một đối tượng mới bằng cách sao chép dữ liệu từ một đối tượng khác
- Có tính overLoad giống bên hàm

```
1  class Rectangle {
2  private:
3      int width, height;
4  public:
5      // Default Constructor
6      Rectangle() : width(5), height(5){}
7      // Parameterized Constructor
8      Rectangle(int w, int h) {
9          width = w;
10         height = h;
11     }
12     // Parameterized Constructor
13     Rectangle(int w = 5, int h) {
14         width = w;
15         height = h;
16     }
17     // Copy Constructor
18     Rectangle(const Rectangle& other) {
19         width = other.w;
20         height = other.h;
21     }
22 };
```

7.5 Destructor

KTLT2: Destructor

Destructor là một phương thức đặc biệt của class được gọi tự động khi một đối tượng của class đó **không còn được sử dụng nữa và chuẩn bị bị hủy**. Mục đích của Destructor là để **giải phóng bất kỳ tài nguyên nào mà đối tượng có thể đã chiếm giữ trong suốt thời gian tồn tại của nó**, như bộ nhớ hoặc các tài nguyên hệ thống
Cấu trúc ngã<Tên class>() { Body }

- Có tên giống như tên class nhưng được tiền tố bởi một dấu ngã (~).
- Không thể được nạp chồng (overload) và mỗi class chỉ có thể có một Destructor.
- Không nhận tham số và không trả về giá trị.
- Được gọi tự động khi đối tượng ra khỏi phạm vi hoạt động hoặc khi delete được sử dụng để giải phóng đối tượng được cấp phát động

```
class Matrix {
private:
    int** data; // Con trỏ đến con trỏ để lưu trữ ma trận
    int rows;
    int cols;

public:
    // Constructor khởi tạo ma trận với kích thước cho trước
    Matrix(int r, int c) : rows(r), cols(c) {
        data = new int*[rows]; // Cấp phát động cho hàng
        for (int i = 0; i < rows; ++i) {
            data[i] = new int[cols]; // Cấp phát động cho cột của mỗi hàng
        }
    }

    // Destructor giải phóng bộ nhớ của ma trận
    ~Matrix() {
        for (int i = 0; i < rows; ++i) {
            delete[] data[i]; // Giải phóng mỗi hàng của ma trận
        }
        delete[] data; // Giải phóng mảng của con trỏ hàng
    }
};

```

7.6 Encapsulation

KTLT2: Encapsulation

Tính đóng gói (Encapsulation) là một trong bốn tính chất cơ bản của lập trình hướng đối tượng (OOP), bên cạnh kế thừa (Inheritance), đa hình (Polymorphism), và trừu tượng (Abstraction). Trong C++, Encapsulation được sử dụng để **ngăn chặn truy cập trực tiếp vào dữ liệu nội bộ của đối tượng từ bên ngoài**, đồng thời cung cấp một giao diện thông qua các hàm công khai (public methods) để tương tác với dữ liệu đó.
Lợi ích của Encapsulation:

- **Bảo mật:** Che giấu dữ liệu nội bộ, chỉ cho phép truy cập thông qua các phương thức được xác định rõ.
- **Kiểm soát:** Kiểm soát cách dữ liệu được sửa đổi hoặc truy cập, giúp ngăn chặn việc sử dụng sai.
- **Dễ sửa đổi:** Dễ dàng thay đổi hoặc nâng cấp một phần của hệ thống mà không ảnh hưởng đến các phần khác.
- **Giảm phức tạp:** Giúp giảm thiểu độ phức tạp của mã nguồn và làm cho nó dễ đọc, dễ bảo trì hơn.

Tầm nhìn (visibility) khả năng nhìn thấy và được phép truy cập thuộc tính hay gọi hàm của class đó. Có ba mức độ tầm nhìn chính trong OOP c++:

- **Public:** Thành viên được khai báo là public có thể được truy cập từ bất kỳ đâu ngoài lớp.
- **Private:** Thành viên private chỉ có thể được truy cập bên trong lớp đó.
- **Protected:** Thành viên protected có thể được truy cập bên trong lớp và bởi các lớp con của nó.

```
1  class Rectangle {
2  private:
3      int width, height; // Dữ liệu nội bộ, không thể truy cập trực tiếp từ bên ngoài
4
5      int setWidth(int w);
6  public:
7      void setDimensions(int w, int h);
8      int getArea() const;
9      void assgment(Rectangle other); // đối tượng trong class
10 };
11 Rectangle r; // đối tượng ngoài class
```

- **Tầm nhìn của Hàm trong Class** là tất cả trong hàm và thuộc tính trong Class đó kể cả khu vực public và private, ví dụ *getArea* có thể nhìn thấy **width, height, setDimensions, setWidth, getArea, assgment**
- **Tầm nhìn của đối tượng bên ngoài Class** thì khả năng nhìn thấy của đối tượng đó khi ở bên ngoài có thể nhìn thấy trong khu vực public **setDimensions, getArea, assgment**
- **Tầm nhìn của đối tượng được khai báo trong Class** là tất cả trong hàm và thuộc tính trong Class đó kể cả khu vực public và private, ví dụ *assgment* có thể nhìn thấy **width, height, setDimensions, setWidth, getArea, assgment**

7.7 Methods

7.7.1 Get và Set

KTLT2: Get và Set

Hàm get và set là phần của nguyên lý đóng gói, giúp bảo vệ dữ liệu bên trong một đối tượng. Chúng cho phép bạn kiểm soát cách truy cập và cập nhật giá trị của các thuộc tính riêng tư.
Vì biến ở dạng private nên dùng get và set để lấy và gán nhằm đảm bảo tính bảo mật và tránh chỉnh sửa không hợp lí

- **Hàm Get (Accessor):**
  - Hàm get được sử dụng để truy cập giá trị của một thuộc tính riêng tư từ bên ngoài class.
  - Nó thường không thay đổi dữ liệu và chỉ trả về giá trị của thuộc tính.
  - thường *const* có ở hàm nhằm đảm bảo chỉ thực hiện get mà không thay đổi thuộc tính nào nữa trong class
- **Hàm Set (Mutator):**
  - Hàm set cho phép bạn cập nhật giá trị của một thuộc tính riêng tư.
  - Nó có thể bao gồm logic kiểm tra để đảm bảo rằng dữ liệu mới là hợp lệ trước khi cập nhật thuộc tính.

```
class Student {
private:
    int age;

public:
    // Hàm get để truy cập giá trị age
    int getAge() const{
        return age;
    }

    // Hàm set để cập nhật giá trị age
    void setAge(const int& a) {
        if (a > 0) { // Kiểm tra giá trị hợp lệ
            age = a;
        }
    }
};
```

7.7.2 const

KTLT2: Const

Hàm thành viên const là một hàm không thay đổi trạng thái của đối tượng mà nó thuộc về. Khi bạn khai báo một hàm thành viên với từ khóa const sau danh sách tham số, bạn đang bảo đảm rằng hàm đó không thể sửa đổi bất kỳ thuộc tính (biến thành viên) nào của lớp, **Không được phép thay đổi thuộc tính và chỉ được phép gọi các hàm const, đối với đối tượng khai báo const chỉ được gọi đến các hàm const**

```
class Dog {
private:
    int age;
public:
    // Hàm thành viên const
    void test1() const {}
    // Hàm thành viên không const
    void test2() {}

    void get() const{
        age = 1; // không được phép thì thành viên không const

        test1(); // được phép vì thành viên const
        test2(); // không được phép thì thành viên không const
    }
};

const Dog a; // biến const
a.test1(); // được phép vì thành viên const
a.test2(); // không được phép thì thành viên không const
```

7.8 Static data members

KTLT2: Static data members

thành viên dữ liệu static là những thành viên của lớp được khai báo với từ khóa static. Các đặc điểm chính của thành viên dữ liệu static bao gồm:

- **Chia sẻ:** Chỉ có một **bản sao của thành viên static được tạo ra cho toàn bộ lớp** và được **chia sẻ bởi tất cả các đối tượng của lớp đó**, không phụ thuộc vào số lượng đối tượng được tạo.
- **Khởi tạo:** Thành viên static được khởi tạo trước khi bất kỳ đối tượng nào của lớp được tạo ra, thậm chí trước khi hàm main bắt đầu.
- **Phạm vi truy cập:** Mặc dù thành viên static chỉ có thể nhìn thấy bên trong lớp, nhưng thời gian tồn tại của nó kéo dài suốt chương trình.
- **Truy cập:** Thành viên static có thể được truy cập mà không cần thông qua đối tượng, bằng cách sử dụng toán tử phân giải phạm vi :: cùng với tên lớp.
- **Định nghĩa:** Thành viên static phải được định nghĩa một cách rõ ràng bên ngoài lớp sử dụng toán tử phân giải phạm vi.

```
1  class Example {
2  public:
3      static int staticValue; // Khai báo thành viên dữ liệu static
4  };
5
6  int Example::staticValue = 0; // Định nghĩa và khởi tạo thành viên dữ liệu static
7
8  // Cách truy cập 1 hay dùng để dễ kiểm sót hơn
9  Example::staticValue = 1;
10
11 // Cách truy cập 2 ít khi dùng
12 Example e;
13 e.staticValue = 2;
```

ý nghĩa Vì biến static được khởi tạo và dùng chung cho mọi đối tượng class đó chúng ta có thể tạo biến static kiểm sót số lương hay các biến cố định.

7.9 Con trỏ This

KTLT2: Polymorphism

Con trỏ this trong C++ là một con trỏ đặc biệt dùng để trỏ đến địa chỉ của đối tượng hiện tại. Khi một hàm thành viên của lớp được gọi, this tự động được truyền vào hàm như một tham số ẩn và nó tham chiếu đến đối tượng đang gọi hàm đó. Đây là một số điểm chính về con trỏ this:

- **Truy cập Thành Viên:** Con trỏ this được sử dụng bên trong các hàm thành viên của lớp để truy cập các thành viên của lớp đó, **đặc biệt là khi tên tham số trùng với tên của thành viên lớp.**
- **Trả Về Đối Tượng Từ Hàm Thành Viên:** Các hàm thành viên có thể sử dụng con trỏ this để trả về đối tượng hiện tại.
- **Không Dùng Cho Hàm Friend:** Các hàm friend không phải là hàm thành viên của lớp nên chúng không có con trỏ this.
- **Không Dùng Cho Hàm Static:** Các hàm static không liên kết với bất kỳ đối tượng cụ thể nào nên chúng không sử dụng con trỏ this.

```cpp
class Employee {
private:
    int id;
    std::string name;
    float salary;
    static int count;
public:

    // Constructor sử dụng con trỏ this để phân biệt giữa tham số và thành viên lớp
    Employee(int id, std::string name, float salary) {
        this->id = id;
        this->name = name;
        this->salary = salary;
        // this->static += 1; // lỗi
    }

    Employee copy(){
        return *this;
    }

    void display() {
        std::cout << id << " " << name << " " << salary << std::endl;
    }
};
```

7.10 Inheritance

KTLT2: Inheritance

Kế thừa (Inheritance) trong lập trình hướng đối tượng C++ là một cơ chế cho phép một lớp (được gọi là lớp dẫn xuất) kế thừa các thuộc tính và phương thức từ một lớp khác (được gọi là lớp cơ sở). Đây là những điểm chính về kế thừa:

- **Tái sử dụng mã nguồn:** Kế thừa cho phép **tái sử dụng mã nguồn của lớp cơ sở, giảm trùng lặp** và tăng tính bảo mật cho dữ liệu.
- **Phân cấp lớp:** Kế thừa tạo ra một cấu trúc phân cấp của các lớp, giúp quản lý và mở rộng mã nguồn dễ dàng hơn.
- **Mở rộng:** Lớp dẫn xuất có thể mở rộng thêm các thuộc tính và phương thức mới mà không ảnh hưởng đến lớp cơ sở.

Cú pháp cơ bản để tạo một lớp dẫn xuất trong C++ là:

```
class DerivedClass : accessSpecifier BaseClass {
    // Thân của lớp dẫn xuất
};
```

đối với BTL này chúng ta dùng accessSpecifier is Public các hàm hay thuộc tính được thừa kế từ class cha sang class con tầm vực public với protected vẫn giữ nguyên

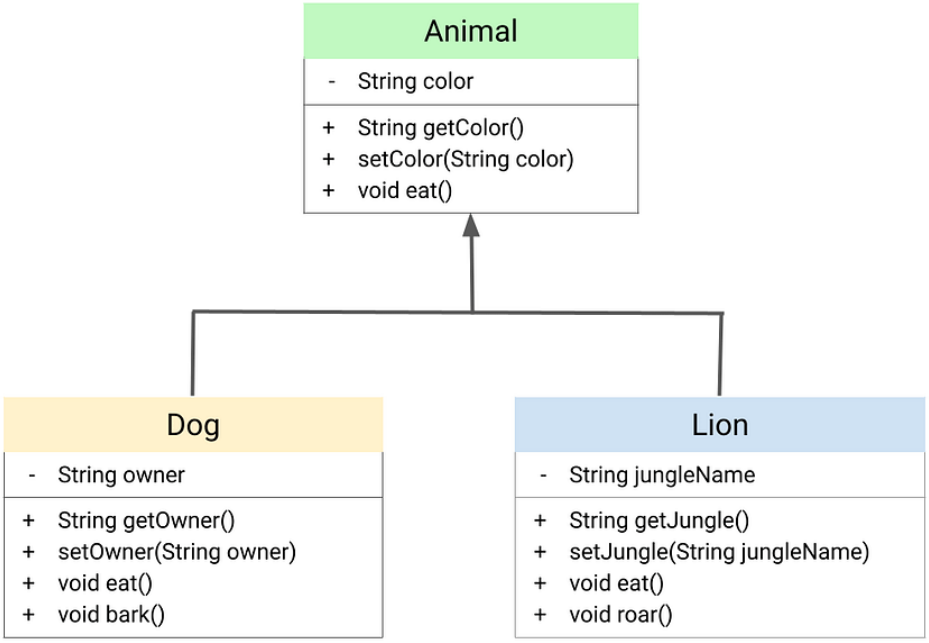
```
class Animal {
private:
    int weight;
protected:
    int age;
public:
    void eat() {
        // Phương thức ăn
    }
};

class Dog : public Animal {
    // Dog kế thừa phương thức eat() và age có thể nhìn thấy
    // nhưng weight sẽ không được nhìn thấy
};
```

Tương Tự đoạn code sau với accessSpecifier is Public

```
class Dog{
protected:
    int age;
public:
    void eat() {
        // Phương thức ăn
    }
};
```

Contructor và destructor trong thừa kế
https://www.geeksforgeeks.org/order-constructor-destructor-call-c/



Phần tích ý tưởng

- ý tưởng của thừa kế đơn giản là tận dụng các code có sẵn và triển khai code không ảnh hưởng đến code cũ
- Tìm nhưng đặt điểm chung của 2 class *dog* và *lion* ở đây là các thuộc tính và phương thức trong *Animal*
- Chon accessSpecifier khi thừa kế phù hợp
- Tạo thêm các thuộc tính và phương thức cho các lớp con
- Chú ý nếu Contructor của lớp cha có tham số thì bắt buộc class con phải gọi đến Contructor của lớp cha để khởi tạo với toán tử :
- Hàm hủy sẽ được gọi khi xóa đối tượng kể cả hàm huy lớp cha và lớp con
- Override các hàm nếu cần ở lớp con
- khi Override muốn gọi tên hàm đã Override ở class cha thì ta dùng toán tử ::
- Nếu quá trình mở rộng thêm một class là *cat* hay thiết kế sao cho không ảnh hưởng các *class* còn lại khi thêm một *class* mới vào, ở đây ta chỉ cần đưa class *cat* thừa kế từ *Animal* và hiện thực nó thôi.
- Nếu ở đây chúng ta có 2 loại *Dog* là Shiba và Husky ta sẽ hiện thực 2 class là Shiba và Husky được thừa kế từ *Dog* và là con cháu của *Animal*
- Trong c++ còn có tính chất đa thừa kế và trong các ngôn ngữ khác thì có interface các bạn tự tìm hiểu

7.11 Friend Class

KTLT2: Friend Class

Friend class là một lớp được cho phép truy cập vào các thành viên private và protected của lớp khác. Điều này có thể hữu ích khi **hai lớp cần chia sẻ dữ liệu hoặc hợp tác chặt chẽ với nhau**, nhưng bạn không muốn cung cấp quyền truy cập rộng rãi đến các thành viên này cho tất cả các lớp

- Mang tính chất không được thừa kế nếu class là bạn của lớp cha thì lớp con không phải là bạn
- Chỉ một chiều nếu A là bạn class B thì B có thể không phải là bạn của A.

```cpp
class Base {
    // Friend có thể truy cập thành viên private và protected của Base
    friend class Friend;
protected:
    int protectedData;
};

class Derived : public Base {
    // Friend KHÔNG thể truy cập trực tiếp protectedData từ Derived
};

class Friend {
public:
    void accessBase(Base& b) {
        // Có thể truy cập protectedData từ Base
        int data = b.protectedData;
    }
    void accessDerived(Derived& d) {
        // Không thể truy cập protectedData từ Derived
        // int data = d.protectedData; // Lỗi biên dịch
    }
};
```

ClassB là friend class của ClassA, nghĩa là ClassB có thể truy cập trực tiếp đến các thành viên private của ClassA. Điều này không thay đổi quyền truy cập của các lớp khác đối với ClassA; chỉ ClassB mới có quyền truy cập đặc biệt này

7.12 Polymorphism

KTLT2: Polymorphism

Đa hình (Polymorphism) trong lập trình hướng đối tượng C++ là một tính chất cho phép **các đối tượng thuộc các lớp khác nhau biểu diễn một hành động thông qua các phương thức khác nhau**. Đây là một số điểm chính về đa hình:

- **Đa hình thời gian biên dịch (Compile-time Polymorphism):** Được thực hiện thông qua nạp chồng hàm (function overloading) và nạp chồng toán tử (operator overloading). Trong đó, các hàm hoặc toán tử có cùng tên nhưng khác nhau về số lượng hoặc kiểu dữ liệu của tham số.
- **Đa hình thời gian chạy (Runtime Polymorphism):** Được thực hiện thông qua ghi đè phương thức (method overriding), nơi mà phương thức của lớp cơ sở được viết lại trong lớp dẫn xuất để thực hiện hành động khác nhau tùy thuộc vào đối tượng của lớp nào đang gọi phương thức.
- **Lợi ích của đa hình:** Giúp giảm sự trùng lặp mã nguồn, tăng khả năng tái sử dụng và mở rộng mã nguồn, cũng như hỗ trợ việc xây dựng các hệ thống có khả năng mở rộng và bảo trì dễ dàng hơn

```cpp
class Animal {
public:
    virtual void makeSound() { cout << "Some sound" << endl;}
};

class Dog : public Animal {
public:
    void makeSound() override {cout << "Woof" << endl;}
};

class Cat : public Animal {
public:
    void makeSound() override {cout << "Meow" << endl;}
};

Animal* myDog = new Dog();
Animal* myCat = new Cat();

myDog->makeSound(); // Outputs: Woof
myCat->makeSound(); // Outputs: Meow
```

- **Virtual:** Từ khóa virtual được sử dụng để chỉ định rằng một phương thức có thể được ghi đè trong một lớp dẫn xuất. Khi một phương thức được khai báo là virtual trong lớp cơ sở, nó cho phép lớp dẫn xuất cung cấp một định nghĩa cụ thể cho phương thức đó. Điều này là cơ sở cho đa hình thời gian chạy, nơi mà một con trỏ hoặc tham chiếu đến lớp cơ sở có thể gọi phương thức từ lớp dẫn xuất tương ứng
- **Override:** Từ khóa override được sử dụng trong lớp dẫn xuất để chỉ rõ rằng phương thức đó đang ghi đè một phương thức virtual từ lớp cơ sở. Nó giúp đảm bảo rằng bạn đang ghi đè một phương thức có sẵn, không phải tạo ra một phương thức mới. Nếu không có phương thức tương ứng trong lớp cơ sở, trình biên dịch sẽ báo lỗi, giúp ngăn chặn lỗi do gõ nhầm tên phương thức hoặc không khớp chính xác với chữ ký của phương thức cơ sở

7.13 Abstraction

KTLT2: Abstraction

Tính trừu tượng (Abstraction) trong lập trình hướng đối tượng C++ là một khái niệm quan trọng giúp giảm sự phức tạp của chương trình bằng cách **ẩn đi các chi tiết kỹ thuật không cần thiết**. Dưới đây là những điểm chính về tính trừu tượng:

- **Ẩn đi chi tiết:** Tính trừu tượng cho phép lập trình viên tập trung vào "cái gì" một đối tượng làm mà không cần quan tâm đến "làm thế nào" nó thực hiện.
- **Sử dụng Interface và Lớp trừu tượng:** Trong C++, tính trừu tượng thường được thực hiện thông qua việc sử dụng các interface và lớp trừu tượng (abstract class). Một lớp trừu tượng chứa ít nhất một hàm thuần ảo (pure virtual function) và không thể tạo đối tượng trực tiếp từ nó.
- **Hàm thuần ảo:** Là hàm không có định nghĩa cụ thể trong lớp cơ sở và bắt buộc phải được định nghĩa lại trong lớp dẫn xuất

```
class Shape {
public:
    // Hàm thuần ảo, định nghĩa lại trong lớp dẫn xuất
    virtual void draw() = 0;
};

class Circle : public Shape {
public:
    // Định nghĩa lại hàm thuần ảo draw
    void draw() override {
        // Vẽ hình tròn
    }
};
```

- **Virtual:** Từ khóa virtual cho biết rằng hàm có thể được ghi đè (overridden) trong một lớp dẫn xuất.
- **void draw():** Đây là chữ ký của hàm, nói rằng hàm này tên là draw và không trả về giá trị (void).
- **= 0:** Cú pháp này chỉ ra rằng hàm draw không có định nghĩa cụ thể trong lớp hiện tại và bắt buộc phải được định nghĩa trong lớp dẫn xuất, trừ khi lớp dẫn xuất đó cũng là một lớp trừu tượng

8 Danh sách liên kết

https://drive.google.com/file/d/1p95yrZoLSayUgUzgbjvOOyG5atG5wxkt/view?usp=sharing

Bài tập tự tìm hiểu:

1. Trong danh sách liên kết đôi, mỗi node có thể trỏ đến bao nhiêu node?

   a) 0
   b) 1
   c) 2
   d) Không giới hạn

2. Lỗi gì có thể xảy ra nếu cố gắng truy cập đến một node không tồn tại trong danh sách liên kết?

   a) Lỗi tràn bộ nhớ (Memory Leak)
   b) Lỗi truy cập không hợp lệ (Segmenation Fault)
   c) Lỗi vòng lặp vô hạn (Infinite Loop)
   d) Lỗi logic trong thuật toán

3. Trong lớp Node, thuộc tính nào được sử dụng để lưu trữ phần tử dữ liệu của node?

```
class BaseBag{
protected:
    class Node{
    public:
        BaseItem * item;
        Node * next;
        friend class BaseBag;
    public:
        Node(BaseItem * item, Node* next = nullptr):item(item), next(next){}
    };
protected:
    int size;
    int capacity;
    Node* head;

public:
};
```

   a) item
   b) next
   c) friend
   d) BaseItem

4. Trong một danh sách liên kết, con trỏ head được sử dụng để tham chiếu đến điều gì?

   a) Phần tử đầu tiên của danh sách.
   b) Phần tử cuối cùng của danh sách.
   c) Phần tử trước đó của node đầu tiên.
   d) Phần tử đằng sau node cuối cùng.

5. Trong thao tác thêm một phần tử vào đầu danh sách liên kết, nếu không cập nhật con trỏ head, điều gì có thể xảy ra?

   a) Danh sách sẽ không thay đổi.
   b) Phần tử mới sẽ được thêm vào cuối danh sách.
   c) Phần tử mới sẽ thay thế phần tử đầu tiên.
   d) Lỗi truy cập không hợp lệ có thể xảy ra.

6. khi chèn một phần tử vào danh sách liên kết thì cần kiểm tra index về gì

   a) index >= 0
   b) index <= size
   c) index > 0
   d) a và b

7. Khi xóa 1 Node không phải node đầu tiên thì cần tìm được node nào ?

   a) Node cần xóa
   b) Node trước Node cần xóa
   c) Node sau Node cần xóa
   d) Node cuối danh sách

8. khi lấy phần tử node ở vị trí **i** Của basebag cần chú ý gì ?

a) i > size - 1
b) i >= 0
c) i < size - 1
d) B và C

9. chọn Điều kiện đúng nhất ?

```
    #include <iostream>

using namespace std;

// Định nghĩa lớp BaseItem (giả sử đã được định nghĩa)

class BaseItem {
   Public get(int i){
        return i;
   }
};

// Định nghĩa lớp BaseBag
class BaseBag {
protected:
    class Node {
    public:
        BaseItem* item;
        Node* next;
        friend class BaseBag;

        Node(BaseItem* item, Node* next = nullptr) : item(item), next(next) {}
    };

    int size;
    int capacity;
    Node* head;

public:
    // Hàm in ra thông tin của từng node trong danh sách liên kết
    void printNodes() {
        Node* current = head;
        while (// điền diều kiện) {
            cout << "Node: " << current->item->get(i) << endl;
            current = current->next;
        }
    }
};

```

a) current != nullptr
b) current == nullptr
c) true
d) current->next != nullptr

10. chọn Điều kiện đúng nhất ?

```
class BaseBag {
protected:
    class Node {
    public:
        BaseItem* item;
        Node* next;
        friend class BaseBag;

        Node(BaseItem* item, Node* next = nullptr) : item(item), next(next) {}
    };

    int size;
    int capacity;
    Node* head;

public:
    // Hàm xóa một phần tử khỏi danh sách liên kết
    void remove(BaseItem* itemToRemove) {
        // Kiểm tra xem danh sách có rỗng không
        if (head == nullptr) {

            return;
        }

        Node* current = head;
        Node* prev = nullptr;
        //Tìm phần tử đúng trước  node cần xóa
        while (//dieu kien) {
            prev = current;
            current = current->next;
        }
        // TODO
    }
};


```

a) current != nullptr
b) current->item != itemToRemove
c) current->item != itemToRemove && current != nullptr
d) current != nullptr && current->item != itemToRemove

9 Ứng Dụng BTL:

1. Điểm khác nhau giữa class và struct trong c++?
   a) Tất cả thành viên của struct là public và không có hàm khởi tạo cũng như hàm hủy
   b) Mặc định, các thành viên của class là private trong khi của struct là public
   c) Tất cả thành viên của struct là public và không có hàm ảo
   d) Tất cả các câu trên

2. Chọn câu đúng với hàm ảo (virtual) trong class?
   a) class chứa hàm ảo sẽ trở thành class trừu tượng (abstract class)
   b) Hàm ảo không thể được gọi bởi đối tượng thuộc class cha
   c) Hàm ảo bắt buộc các class con phải hiện thực
   d) Hàm ảo không cho phép các class con ghi đè

3. Chọn câu đúng với hàm thuần ảo (pure virtual) trong class?
   a) class chứa hàm thuần ảo sẽ trở thành class trừu tượng (abstract class)
   b) Hàm thuần ảo không thể được gọi bởi đối tượng thuộc class cha
   c) Hàm thuần ảo bắt buộc các class con phải hiện thực
   d) Hàm thuần ảo cho phép các class con ghi đè

4. class trong C++ có bao nhiêu kiểu kế thừa?
   a) 1 b) 2
   c) 3 d) 4

5. class con có thể truy cập dữ liệu trong phạm vi nào của class cha?
   a) protected b) public, protected
   c) private, public d) private, public, protected

6. Ý nghĩa của việc kế thừa kiểu **public**?
   a) Dữ liệu không đổi phạm vi truy cập
   b) Đổi phạm vi truy cập của toàn bộ dữ liệu kế thừa thành public trong class con
   c) Có thể truy cập vào dữ liệu private của class cha
   d) Để chỉ kế thừa dữ liệu public từ class cha

7. Ý nghĩa của việc kế thừa kiểu **private**?
   a) Dữ liệu không đổi phạm vi truy cập
   b) Đổi phạm vi truy cập của toàn bộ dữ liệu kế thừa thành private trong class con
   c) Có thể truy cập vào dữ liệu private của class cha
   d) Để chỉ kế thừa dữ liệu private từ class cha

8. Định nghĩa hàm **const** có ý nghĩa gì?

```
1 int getReqExp () const ;
```

   a) Hàm không làm thay đổi thuộc tính của class
   b) Không thể thay đổi giá trị các tham số
   c) Chỉ cho phép đối tượng được định nghĩa **const** thực hiện gọi hàm
   d) Tất cả đều đúng

Đọc đoạn code sau và trả lời câu hỏi

```
class MapElement
{
public:
    MapElement()
    {
        cout << "MapElement" << endl;
    }
};

class Wall : public MapElement
{
public:
    Wall()
    {
        cout << "Wall" << endl;
    }
};

int main()
{
    <Câu lệnh thực hiện>
}
```

9. Kết quả đoạn code khi thực hiện câu lệnh **MapElement ele;**

a) Wall
Element

b) MapELement

c) Wall

d) Element
Wall

10. Kết quả đoạn code khi thực hiện câu lệnh **MapElement *ele = new Wall();**

a) Wall
Element

b) MapELement

c) Wall

d) Element
Wall

Đọc đoạn code sau và trả lời câu hỏi

```
class MapElement
{
public:
    virtual ~MapElement()
    {
        cout << "Delete MapElement" << endl;
    }
};

class Wall : public MapElement
{
public:
    ~Wall(){
        cout << "Delete Wall" <<endl;
    }
};

```

```
18 int main()
19 {
20     MapElement *ele = new Wall();
21     delete ele;
22 }
```

11. Kết quả của đoạn code?

    a) Delete MapElement
       Delete Wall
    b) Delete MapElement
    c) Delete Wall
       Delete MapElement
    d) Delete Wall

12. Nếu hàm hủy của MapElement không phải hàm ảo thì kết quả của chương trình sẽ là?

    a) Delete MapElement
       Delete Wall
    b) Delete MapElement
    c) Delete Wall
       Delete MapElement
    d) Delete Wall

13. Cách định nghĩa hàm ngoài class ngoài sau đây là hợp lệ?

    a) void Sherlock::move(){}
    b) void move()::Sherlock{}
    c) Sherlock::void move(){}
    d) void move::Sherlock(){}

14. Vị trí nào là vị trí gán giá trị cho biến **size** hợp lệ?

```
1  class BaseBag{
2  public:
3      static int size;
4      // Line0
5      BaseBag(){
6          //Line1
7          size++;
8      }
9  };
10 // Line2
11 int main(){
12     // Line3
13     return;
14 };
```

    a) Line0
    b) Line1
    c) Line2
    d) Line3

15. Câu lệnh gán giá trị cho **size** hợp lệ?

    a) BaseBag::size = 0;
    b) size = 0;
    c) int BaseBag::size = 0;
    d) BaseBag size = 0;

16. Cách truy xuất thuộc tính/hàm static nào của class sau đây là hợp lệ?

    a) Position.npos
    b) Position:npos
    c) Position::npos
    d) Position->npos

17. Chọn câu trả lời phù hợp vào chỗ trống?

```
class Sherlock{
public:
    int money;
    Sherlock(int money) ...
};
```

a) {this->money = money;}
b) : money(money) {}
c) A và B đúng
d) Báo lỗi

18. Chọn cách truy cập hợp lệ?

```
class BaseBag {
public:
    class Node {
    public:
        static int size;
    };
};
```

a) BaseBag::Node::size
b) BaseBag->Node->size
c) Node::size
d) Node.size

19. Kết quả của đoạn code sau

```
class BaseBag{
public:
    static int size
    BaseBag(){
        size++;
    }
};
int BaseBag::size = 0;
int main(){
    BaseBag *bag = new BaseBag[10]();
    cout<<BaseBag::size<<endl;
}
```

a) 0
b) 1
c) 9
d) 10

10 Chơi Game

1. Hiện thực hàm khởi tạo Configuration() trong file config.cpp
   Link drive
   Mô tả:
   - Hàm nhận vào tham số filename có kiểu dữ liệu string chính là tên của file cần đọc và lấy thông tin.
   - Hàm cần truyền dữ liệu vào các biến toàn cục trong file main.h
   - Dữ liệu cần lấy từ mỗi dòng trong file
     - **MAP_NUM_ROWS=<nr>**
       - Gán giá trị <nr> cho biến map_num_rows
     - **MAP_NUM_COLS=<nc>**
       - Gán giá trị <nc> cho biến map_num_cols
     - **MAX_NUM_MOVING_OBJECTS=<mnmo>**
       - Gán giá trị <mnmo> cho biến max_moving_objects
     - **ARRAY_WALLS=<aw>**
       - <aw> có định dạng : [(<x0>,<y0>);(<x1>,<y1>); ... (<xn>,<yn>)]
       - Số cặp x y cần được lưu vào biến num_walls
       - Cấp phát vùng nhớ cho mảng với kích thước bằng số cặp x y
       - Một cặp x y cần được lưu vào arr_walls[i][0] arr_walls[i][1]
     - **ARRAY_WALLS=<afw>**
       - <afw> có định dạng : [(<x0>,<y0>);(<x1>,<y1>); ... (<xn>,<yn>)]
       - Số cặp x y cần được lưu vào biến num_fake_walls
       - Cấp phát vùng nhớ cho mảng với kích thước bằng số cặp x y
       - Một cặp x y cần được lưu vào arr_fake_walls[i][0] và arr_fake_walls[i][1]
     - **SHERLOCK_MOVING_RULE=<smr>**
       - Gán giá trị <smr> cho biến sherlock_moving_rule
     - **SHERLOCK_INIT_POS=<sip>**
       - <sip> có định dạng : (<x>,<y>)
       - Gán giá trị x y vào biến sherlock_init_pos[0] và sherlock_init_pos[1]
     - **SHERLOCK_INIT_HP=<sih>**
       - Gán giá trị <sih> cho biến sherlock_init_hp
     - **SHERLOCK_INIT_EXP=<sie>**
       - Gán giá trị <sie> cho biến sherlock_init_exp
     - **WATSON_MOVING_RULE=<wmr>**
       - Gán giá trị <wmr> cho biến watson_moving_rule
     - **WATSON_INIT_POS=<wip>**
       - <wip> có định dạng : (<x>,<y>)
       - Gán giá trị x y vào biến watson_init_pos[0] và watson_init_pos[1]
     - **WATSON_INIT_HP=<wih>**

    * Gán giá trị <wih> cho biến watson_init_hp
  - **WATSON_INIT_EXP=<wie>**
    * Gán giá trị <wie> cho biến watson_init_exp
  - **CRIMINAL_INIT_POS=<cip>**
    * <cip> có định dạng : (<x>,<y>)
    * Gán giá trị x y vào biến crimianl_init_pos[0] và criminal_init_pos[1]
  - **NUM_STEPS=<ns>**
    * Gán giá trị <ns> cho biến num_steps

2. Điền khuyết vào đoạn **code**

```
class Position {
private:
    int r, c;
public:
    static const Position npos;

    Position(int r, int c)
    // Code(1)
    {}
    Position(const Position& other)
    {
        // Code (2)

    }

    int getRow() const
    {
        // Code(3)

    }
    // với r > 0 trả về true, còn lại trả về false
    bool setRow(int r)
    {
        // Code(4)



    }
};
```

3. Điền khuyết vào đoạn **code**

```
// cách 1 dùng biến type
enum ElementType { PATH, WALL, FAKE_WALL };
class MapElement {
protected:
    ElementType type;
public:
    MapElement(ElementType in_type);
    virtual ElementType getType() const
    {
        // Code(1)

    }
};

class Path : public MapElement {
public:
    Path()
    // Code(2)
    {}
};

```

```
22
23 // cách 2 dùng method
24 enum ElementType { PATH, WALL, FAKE_WALL };
25 class MapElement {
26 public:
27     virtual ElementType getType() const = 0
28 };
29
30 class Path : public MapElement {
31 public:
32     ElementType getType() const{
33         // Code(3)
34
35     }
36 };
```

giải thích mấy câu sau

(a) So sánh 2 cách dùng để tìm **getType**

(b) Vì sao biến **type** lại là biến protected mà không phải private

(c) Nếu thay thừa kế **class Path : public MapElement** thành **class Path : protected MapElement** thì điều gì sẽ xảy ra

(d) Khai báo **MapElement* map = new MapElement()** thì cách nào không lỗi vì sao.

(e) Nếu **virtual trong hàng 8** bị mất đi có ảnh hưởng gì không

(f) Nêu ý nghĩa **const trong hàng 8 và 32**

(g) Nếu thêm một **class Wall : public MapElement** thì cách 1 bắt buộc phải có hàm gì và cách 2 bắt buộc phải có hàm gì

4. Điền khuyết vào đoạn **code**

```
1  enum ElementType { PATH, WALL, FAKE_WALL };
2  class MapElement {
3  public:
4      virtual ElementType getType() const;
5  };
6
7  class Path : public MapElement {
8      // Path
9  };
10
11 class Wall : public MapElement {
12     // Wall
13 };
14
15 class FakeWall : public MapElement {
16     // FakeWall
17 };
18
19 class Position {
20 private:
21     int r, c;
22 public:
23     int getRow() const;
```

```
    int getCol() const;
};


class Map {
private:
    int num_rows, num_cols;
    MapElement*** map; // mảng 2 chiều với từng phần tử là con trỏ
public:
    ElementType getElementType(int i, int j) const
    {
        // Code(1)

    }
};
```

(a) Hiện thực hàm khởi tạo sau với tất cả đều là MapElement đều là Path

```
    Map(int num_rows, int num_cols)
    {
        //todo implement
    }
```

(b) Chơi game nào hãy điền vào các ô với các thông số sau

- num_rows = num_cols = 10
- num_walls = 5
- array_walls = {Position(1,1), Position(5,5), Position(3,1), Position(1,9), Position(4, 4)}
- num_fake_walls = 2
- array_fake_walls = {Position(2,2), Position(7,7)}
- Các ô còn lại là Path

Hãy điền vào các ô sau với

- **Path kí hiệu là ô trống**
- **Wall kí hiệu là W**
- **FakeWall kí hiệu là F**

(c) Hiện thực hàm khởi tạo sau với **int num_walls, Position * array_walls** là số lượng tường và vị trí của từng tường đang lưu trong mảng và **int num_fake_walls, Position * array_fake_walls** là số lượng tường ảo và vị trí của từng tường ảo đang lưu trong mảng, các ô còn lại đều là Path

```
Map(int num_rows, int num_cols, int num_walls, Position * array_walls,
 ↪  int num_fake_walls, Position * array_fake_walls)
{
    //todo implement
}
```

(d) Hiện thực hàm hủy

```
~Map();
```

5. hiện thực class *ArrayMovingObject*

```cpp
class MovingObject {
protected:
    Position pos;
public:
    Position getCurrentPosition() const{
        return pos;
    }
    virtual string str() const = 0;
};
class  Character : public MovingObject{
public:
    virtual string str() const = 0;
}
class Sherlock : public Character{
public:
    string str() const{
        return "Sherlock";
    }
}
class Watson : public Character{
public:
    string str() const{
        return "Watson";
    }
}

class ArrayMovingObject {
private:
    MovingObject** arr_mv_objs; // mảng đối tượng
    int count; // số lượng nhân vật đang có
    int capacity; // số lượng nhân vật tối đa
public:
    ArrayMovingObject(int capacity);
    ~ArrayMovingObject() ;
    bool isFull() const;
    bool add(MovingObject * mv_obj);
    void remove(int index);
    MovingObject * get(int index) const;
    int size() const; // return current number of elements in the array
};
```

(a) Hiện thực cách hàm sau

- **ArrayMovingObject(int capacity)** nhận vào kích thước tối đa của mảng đối tượng, khởi tạo mảng đối tượng với kích thước đó
- **ngã ArrayMovingObject()** xóa mảng đối tượng đã được khởi tạo và các nhân vật trong đó
- **isFull()** kiểm tra mảng đã đầy hay chưa
- **add()** thêm đối tượng vào cuối mảng, nếu đầy thì bỏ qua
- **remove(int index)** xóa đối tượng tại vị trí *Index* trong mảng bắt đầu từ 0, nếu không có thì bỏ qua

- **get(int index)** lấy đối tượng tại vị trí *Index* trong mảng bắt đầu từ 0, nếu không có thì bỏ qua
- **size()** kích thước hiện tại của mảng

(b) Chơi game nào hãy điền vào các ô với các thông số sau

- arr_mv_objs = {Sherlock(Position(1,3)),Sherlock(Position(2,3)),Sherlock(Position(5,3)), Sherlock(Position(1,9)),Watson(Position(1,3)),Watson(Position(3,3))}

Hãy điền vào các ô sau với

- **Path kí hiệu là ô trống**
- **Wall kí hiệu là W**
- **FakeWall kí hiệu là F**
- **Sherlock kí hiệu là S**
- **Watson kí hiệu là T**

	F		W						
				W		F		F	
	F	W	F			F			
	F		F	F		W		W	
			W		F		F		
	F			F			F	W	
		W		F			W		
	F				W			F	
			W				W		

6. Hiện thực class sau

```
class MovingObject {
protected:
    Position pos; // vị trí hiện tại của nhân vật
    Map * map; // ma trận đường đi
    int count_move; // số lượng bước đã di chuyển không tính đứng yên
public:
    Position getCurrentPosition() const{return pos;}
    virtual void move() = 0;
};
class  Character : public MovingObject{
public:
    virtual void move() = 0;
}
class Sherlock : public Character{
    string moving_rule; // hướng di chuyển
    int index_moving_rule; // vị trí trong chuỗi của hướng di chuyển
public:
    void move();
}
class Watson : public Character{
    string moving_rule; // hướng di chuyển
    int index_moving_rule; // vị trí trong chuỗi của hướng di chuyển
public:
    void move();
}
class Criminal : public Character{
    Sherlock *sherlock;
    Watson *watson;
public
    void move();
}
```

(a) Hiện thực 3 hàm move của Sherlock, Watson, Criminal

- **moving_rule:** mô tả cách thức mà Sherlock di chuyển. Đây là một chuỗi mà các ký tự chỉ có thể là một trong 4 giá trị: 'L' (Left - đi sang trái), 'R' (Right - đi sang phải), 'U' (Up - đi lên trên), 'D' (Down - đi xuống dưới). Ví dụ với moving_rule = "LR" thì thứ tự các ký tự được sử dụng là: 'L', 'R', 'L', 'R', 'L', 'R',... Nếu Position được trả ra không phải là một vị trí hợp lệ cho đối tượng này di chuyển thì trả về npos thuộc class Position
- **pos:** kiểu Position, vị trí hiện tại của đối tượng di chuyển.
- **map:** kiểu Map *, bản đồ cho đối tượng này di chuyển trong đó
- **count_move:** số bước di chuyển không tính đứng yên
- **index_moving_rule:** vị trí kí tự đang xét nếu vượt quá thì quay lại vị trí ban đầu
- **move Criminal** Tên tội phạm có camera theo dõi cả Sherlock và Watson trong mê cung này. Do đó, khác với cách di chuyển của cặp đôi thám tử, tên tội phạm sẽ lựa chọn vị trí di chuyển tiếp theo là vị trí hợp lệ có tổng khoảng cách đến Sherlock và Watson là lớn nhất Trong BTL này, khi nói đến khoảng cách, ta đang sử dụng khoảng cách Manhattan. Khoảng cách Manhattan giữa 2 điểm P1 có tọa độ (x1, y1) và P2 có tọa độ (x2, y2) là: |x1 x2| + |y1 y2|. Trong trường hợp có nhiều hơn 1 vị trí đều có tổng khoảng cách đến Sherlock và Wáton là lớn nhất thì ưu tiên chọn vị trí theo thứ tự các hướng đi 'U', 'L', 'D', 'R'

(b) Chơi game nào hãy điền vào các ô với các thông số sau

- moving_rule của Sherlock = "RD"
- moving_rule của Watson = "TL"

Hãy điền vào các ô sau với

- **Path kí hiệu là ô trống**
- **Wall kí hiệu là W**
- **FakeWall kí hiệu là F**
- **Sherlock kí hiệu là S với mỗi bước đi lần lược S1, S2, S3 ....**
- **Watson kí hiệu là T với mỗi bước đi lần lược T1, T2, T3 ....**
- **Criminal kí hiệu là C với mỗi bước đi lần lược C1, C2, C3 ....**
- **Thua khi quá 20 bước không bắt được, thắng khi S và T chung ô với C**

S	F		W						
				W		F		F	
	F	W	F			F			
					C				
	F		F	F		W		W	
			W				F		
	F			F			F	W	
		W		F			W		
	F				W			F	
			W						**T**

7. Hiện thực Danh sách liên kết sau

```
class BaseItem{
public:
    virtual bool canUse ( Character * obj , Robot * robot ) = 0;
    virtual string str() const = 0;
};

class BaseBag{
protected:
    class Node{
    public:
        BaseItem * item; // vật phẩm
        Node * next; // vật phẩn tiếp theo
        friend class BaseBag;
    public:
        Node(BaseItem * item, Node* next = nullptr):item(item), next(next){}
    };
protected:
    int size; // kích thước hiện tại
    int capacity; // kích thước tối đa
    Node* head; // con trỏ head
public:
    BaseBag(int capacity):capacity(capacity){};
    virtual ~BaseBag();
    virtual bool insert ( BaseItem * item );
    virtual BaseItem * get ();
};
```

- **insert :** Thêm vật phẩm vào đầu danh sách liên kết
- **get :** Tìm vật phẩm có thể dùng được sau đó hoán đổi với vật phẩm ở đầu rồi lấy ra từ trong túi

Thảo luận BTL môn KTLT, DSA, NMLT, PPL ....
https://www.facebook.com/groups/211867931379013

- Lớp BTL1 + GK + LAB + Lý thuyết + Harmony của môn DSA HK232
- Lớp BTL2 + CK + LAB + Lý thuyết + Harmony của môn DSA HK232
- Lớp BTL1 + Lý thuyết + Harmony của môn KTLT HK232
- Lớp BTL2 + Lý thuyết + Harmony của môn KTLT HK232
- Lớp CK + LAB + Harmony của môn KTLT HK232
- Lớp BTL1 + BTL2 + GK + Harmony của môn PPL HK232
- Lớp BTL3 + BTL4 + CK + Harmony của môn PPL HK232

CHÚC CÁC EM HỌC TỐT