VÕ TIẾN
Thảo luận kiến thức CNTT trường BK về KHMT(CScience), KTMT(CEngineering)
https://www.facebook.com/groups/khmt.ktmt.cse.bku

Kỹ Thuật Lập Trình (Cơ bản và nâng cao C++)

KTLT1 - HK242

TASK 2 NMLT và cách debug

Thảo luận kiến thức CNTT trường BK
về KHMT(CScience), KTMT(CEngineering)
https://www.facebook.com/groups/khmt.ktmt.cse.bku

TP. HỒ CHÍ MINH, THÁNG 2/2025

Mục lục

1 Các Khái niệm cơ bản trong c++

1.1 Cú pháp (syntax)

Cú pháp C++ bao gồm nhiều thành phần khác nhau xác định cách tổ chức mã nguồn. Dưới đây là mô tả chi tiết các thành phần chính:

Khoảng trắng (Whitespace)

C++ bỏ qua các khoảng trắng dư thừa (dấu cách, tab, xuống dòng) trừ khi chúng nằm trong chuỗi hoặc dùng để phân tách token. Khoảng trắng giúp cải thiện khả năng đọc mã.

```
int a = 5;   // Hợp lệ
int     b = 10;   // Cũng hợp lệ (dấu cách dư thừa bị bỏ qua)
```

Câu lệnh (Statements)

Câu lệnh là một đơn vị thực thi trong C++, thường kết thúc bằng dấu chấm phẩy (;). Câu lệnh thực thi tuần tự, trừ khi có cấu trúc điều khiển thay đổi luồng thực thi.

```
int x = 10;  // Câu lệnh khai báo
x = x + 5;   // Câu lệnh gán
cout << x;   // Câu lệnh xuất dữ liệu
```

Định danh (Identifiers)

Định danh là tên đặt cho biến, hàm, mảng và các thực thể do người dùng định nghĩa. Chúng phải bắt đầu bằng chữ cái (`a-z, A-Z`) hoặc dấu gạch dưới (`_`), tiếp theo là chữ cái, số (`0-9`) hoặc dấu gạch dưới.

```
int myVar;      // Hợp lệ
double _value;  // Hợp lệ
int 123abc;     // Không hợp lệ (không thể bắt đầu bằng số)
```

Từ khóa (Keywords)

Từ khóa là các từ đã được C++ định nghĩa trước với mục đích cụ thể. Không thể sử dụng từ khóa làm tên biến hoặc hàm.

Một số từ khóa trong C++:

```
int, double, float, char, bool
if, else, switch, case, default
for, while, do, return, break, continue
class, struct, namespace, public, private
```

Ví dụ:

```
int number = 10;  // 'int' là từ khóa
return 0;         // 'return' là từ khóa
```

Biểu thức (Expressions)

Biểu thức kết hợp các biến, hằng số và toán tử để tạo ra giá trị.

Biểu thức số học:

```
int a = 10 + 5;  // Phép cộng
int b = a * 2;   // Phép nhân
```

Biểu thức logic:

```
bool result = (a > b) && (b > 5); // Phép AND logic
```

Biểu thức quan hệ:

```
bool check = (a != b); // So sánh khác nhau
```

Chú thích (Comments)

Chú thích giúp giải thích mã nguồn và không ảnh hưởng đến chương trình.

Chú thích một dòng:

```
// Đây là chú thích một dòng
int x = 5;  // Khai báo biến
```

Chú thích nhiều dòng:

```
/*
Chú thích nhiều dòng.
Có thể bao gồm nhiều dòng văn bản.
*/
int y = 10;
```

Kết luận Trên đây là các quy tắc cú pháp cơ bản trong C++. Nắm vững những quy tắc này giúp bạn viết mã dễ hiểu, có cấu trúc và hiệu quả hơn.

1.2 Khai báo biến (variables) và sử dụng biến

Định nghĩa biến

Biến trong C++ là một tên đại diện cho một vùng nhớ trong bộ nhớ máy tính, nơi lưu trữ dữ liệu có thể thay đổi trong quá trình thực thi chương trình.

Cấu trúc chung của một biến:

`kiểu_dữ_liệu tên_biến;`

Ví dụ:

```
int age;  // Biến age có kiểu int, chưa gán giá trị
```

Khai báo biến

Khai báo biến là quá trình thông báo cho trình biên dịch biết về sự tồn tại của một biến với một kiểu dữ liệu cụ thể.

1. Khai báo biến không khởi tạo

```
int x;  // Chưa gán giá trị, có thể chứa giá trị rác
```

2. Khai báo biến có khởi tạo

```
int y = 10;   // Khởi tạo giá trị cho biến y
float pi = 3.14;
char letter = 'A';
```

3. Khai báo nhiều biến cùng lúc

```
int a = 5, b = 10, c = 15;
```

4. Biến hằng số (const) Hằng số là biến không thể thay đổi giá trị sau khi được khởi tạo.

```
const double PI = 3.14159;
```

Gán giá trị cho biến

Có thể gán giá trị cho biến tại thời điểm khai báo hoặc sau đó bằng toán tử =.

```
int num;    // Khai báo biến num
num = 20;   // Gán giá trị 20 cho num
```

Biến cũng có thể nhận giá trị từ một biểu thức khác:

```
int x = 5, y = 10;
int sum = x + y;  // sum = 15
```

Tầm vực (scope) của biến trong C++

Tầm vực (scope) của biến quyết định nơi mà biến có thể được truy cập và sử dụng trong chương trình. Có hai loại tầm vực chính: **biến cục bộ (local variable)** và **biến toàn cục (global variable)**.

1. **Biến cục bộ (Local Variable)**
   - Biến cục bộ là biến được khai báo **bên trong một khối lệnh (block)**, thường là trong một hàm, vòng lặp, hoặc câu lệnh điều kiện.
   - Biến cục bộ **chỉ có hiệu lực trong khối lệnh chứa nó** và **không thể truy cập từ bên ngoài**.
   - Khi khối lệnh kết thúc, biến cục bộ bị hủy.
   - **Quy tắc tìm biến:** Khi truy cập một biến, chương trình sẽ **tìm trong phạm vi gần nhất trước**. Nếu không tìm thấy, nó tiếp tục tìm ở phạm vi bên ngoài. Nếu vẫn không tìm thấy, chương trình sẽ báo lỗi.
   - **Khai báo trước khi sử dụng:** Một biến **phải được khai báo trước khi có thể sử dụng**. Nếu cố gắng sử dụng một biến trước khi khai báo, chương trình sẽ báo lỗi biên dịch.

```cpp
int main() {
    int x = 5; // Biến cục bộ của main()
    cout << "Outside block, x = " << x << endl;

    {
        // Lỗi nếu sử dụng y ở đây vì chưa khai báo
        // cout << y << endl;

        int y = 10;  // Biến cục bộ trong block
        cout << "Inside block, y = " << y << endl;

        cout << "Inside block, x = " << x << endl; // Giá trị 5, lấy từ block ngoài

        int x = 20;  // Khai báo lại biến x trong block, che khuất x bên ngoài
        cout << "Inside block, new x = " << x << endl; // Giá trị 20, vì x trong block này

    } // Kết thúc block, y và x (trong block) bị hủy

    // cout << y; // Lỗi! y chỉ tồn tại trong block
    cout << "Outside block, x = " << x << endl; // Giá trị 5, vì x trong block đã bị hủy

    return 0;
}
```

2. **Biến toàn cục (Global Variable)**
   - Biến toàn cục là biến được khai báo **bên ngoài tất cả các hàm**, thường là **trên đầu chương trình**.
   - Biến toàn cục **có thể được sử dụng ở bất kỳ đâu trong chương trình** (trong mọi hàm).
   - Biến toàn cục tồn tại **suốt vòng đời của chương trình**.

```cpp
int globalVar = 100;  // Biến toàn cục

int main() {
    cout << "Inside main: " << globalVar << endl;
    return 0;
}
```

Cách máy tính lưu trữ biến cục bộ (Local Variables) trong bộ nhớ

Khi một biến **cục bộ (local variable)** được khai báo trong một khối lệnh {}, máy tính sẽ lưu trữ nó trong **Stack Memory**. Dưới đây là các nguyên tắc chính:

1. **Lưu trên Stack Memory**
   - Biến cục bộ được lưu trên **stack**, một vùng bộ nhớ có cấu trúc **LIFO (Last In, First Out)**.
   - Khi một hàm hoặc khối {} được gọi, bộ nhớ cho biến cục bộ sẽ được cấp phát.
   - Khi hàm hoặc khối kết thúc, bộ nhớ của biến sẽ bị giải phóng.
2. **Mỗi biến có phạm vi riêng**
   - Biến chỉ tồn tại trong phạm vi {} mà nó được khai báo.
   - Nếu một biến cùng tên được khai báo trong một khối lồng nhau, nó sẽ **che khuất (shadow)** biến bên ngoài.
3. **Truy tìm và truy cập biến**
   - Khi truy cập một biến, chương trình tìm kiếm theo nguyên tắc từ trong ra ngoài:
     (a) Tìm biến trong khối hiện tại.
     (b) Nếu không tìm thấy, tìm trong khối bao ngoài gần nhất.
     (c) Tiếp tục tìm ra các khối bên ngoài cho đến phạm vi toàn cục.
   - Nếu không tìm thấy biến ở bất kỳ phạm vi nào, trình biên dịch sẽ báo lỗi "undeclared identifier".
   - Nếu có biến trùng tên ở nhiều mức khác nhau, chương trình sử dụng biến thuộc phạm vi gần nhất.

```cpp
int main() {
    int a = 1;  // Biến a ở phạm vi ngoài cùng
    {
        int a = 2;  // Biến a mới che khuất a bên ngoài
        {
            int a = 3;  // Biến a mới tiếp tục che khuất a của khối trên
            cout << a << endl;  // Kết quả in ra: 3
        }
        cout << a << endl;  // Kết quả in ra: 2
    }
    cout << a << endl;  // Kết quả in ra: 1
    return 0;
}
```

Khi thực thi chương trình, các biến cục bộ được cấp phát và thu hồi theo cơ chế **Stack** (LIFO - Last In, First Out). Dưới đây là từng bước hoạt động:

1. **Bắt đầu chương trình**: Stack trống.

2. **Khai báo ‘a = 1‘ trong ‘main()‘**:
   - Biến ‘a = 1‘ được cấp phát trên Stack.

Stack (cao → thấp)
a = 1 (main)

3. **Vào khối ‘ int a = 2; ‘**:
   - Biến ‘a = 2‘ được khai báo, đẩy vào Stack.
   - Biến ‘a = 2‘ **che khuất** biến ‘a = 1‘, nhưng cả hai vẫn tồn tại.

Stack (cao → thấp)
a = 2 (khối trong)
a = 1 (main)

4. **Vào khối ‘ int a = 3; ‘**:
   - Biến ‘a = 3‘ được khai báo, đẩy vào Stack.
   - Biến ‘a = 3‘ **che khuất** biến ‘a = 2‘.

Stack (cao → thấp)
a = 3 (khối trong cùng)
a = 2 (khối trong)
a = 1 (main)

5. **Thoát khối ‘ int a = 3; ‘**:
   - Biến ‘a = 3‘ bị xóa khỏi Stack.

Stack (cao → thấp)
a = 2 (khối trong)
a = 1 (main)

6. **Thoát khối ‘ int a = 2; ‘**:
   - Biến ‘a = 2‘ bị xóa khỏi Stack.

Stack (cao → thấp)
a = 1 (main)

7. **Thoát ‘main()‘**:
   - Biến ‘a = 1‘ bị xóa khỏi Stack.
   - Stack trở về trạng thái ban đầu (trống).

Stack (cao → thấp)
(Trống)

1.3 Vào ra dữ liệu

Trong C++, để nhập (input) và xuất (output) dữ liệu, chúng ta sử dụng thư viện `<iostream>` với các đối tượng tiêu chuẩn:

- `cin` (Console Input): Đọc dữ liệu từ bàn phím.
- `cout` (Console Output): Hiển thị dữ liệu ra màn hình.

Xuất dữ liệu (`cout`)

C++ sử dụng `cout` kết hợp với toán tử « để in dữ liệu ra màn hình.

```cpp
#include <iostream>
using namespace std;

int main() {
    cout << "Hello, C++!" << endl; // In chuỗi và xuống dòng
    cout << "Number: " << 10 << endl; // In số nguyên
    cout << "Pi: " << 3.14 << endl; // In số thực

    return 0;
}
```

Kết quả:

```
Hello, C++!
Number: 10
Pi: 3.14
```

`endl` dùng để xuống dòng (có thể thay bằng `""`). Có thể in nhiều kiểu dữ liệu trên cùng một dòng bằng cách dùng « nhiều lần.

Nhập dữ liệu (`cin`)

C++ sử dụng `cin` kết hợp với toán tử » để nhận dữ liệu từ bàn phím.

```cpp
#include <iostream>
using namespace std;

int main() {
    int age;
    cout << "Enter your age: ";
    cin >> age; // Nhập giá trị từ bàn phím
    cout << "Your age is: " << age << endl;

    return 0;
}
```

Ví dụ nhập nhiều giá trị:

```cpp
int a, b;
cout << "Enter two numbers: ";
cin >> a >> b; // Nhập hai số cách nhau bởi dấu cách
cout << "Sum = " << (a + b) << endl;
```

Lưu ý:

- `cin` bỏ qua khoảng trắng, tab, xuống dòng khi nhập chuỗi.
- Nếu nhập sai kiểu dữ liệu, `cin` sẽ bị lỗi và không nhận giá trị tiếp theo.

Các ký tự đặc biệt trong C++ khi sử dụng cout

Ký tự đặc biệt	Ý nghĩa	Ví dụ
\n	Xuống dòng	`cout « "Hello\nWorld!";`
\t	Tab	`cout « "Hello\tWorld!";`
\b	Backspace	`cout « "Hello\bWorld!";`
\r	Quay về đầu dòng	`cout « "Hello\rWorld!";`
\textbackslash	Dấu gạch chéo ngược	`cout « "Path: C:\\Files";`
\’	Dấu nháy đơn	`cout « "It’s a test";`
\"	Dấu nháy kép	`cout « "She said \"Hi!\"";`
\a	Âm thanh (bell)	`cout « "\a";`
\v	Tab dọc (vertical tab)	`cout « "Hello\vWorld!";`
\f	Form feed	`cout « "Hello\fWorld!";`

Bảng 1: *Các ký tự đặc biệt trong C++*

1.4 Kiểu dữ liệu và biểu thức

Kiểu dữ liệu cơ bản

Trong C++, kiểu dữ liệu được sử dụng để xác định loại và phạm vi của dữ liệu mà một biến có thể chứa. Các kiểu dữ liệu chính cơ bản trong C++ bao gồm:

Kiểu dữ liệu	Kích thước	Mô tả	Kiểu giá trị
`int`	4 bytes (thường)	Kiểu số nguyên, sử dụng để lưu trữ các giá trị nguyên	`10, -3`
`float`	4 bytes	Kiểu số thực với độ chính xác đơn	`3.14f, -1.5f`
`double`	8 bytes	Kiểu số thực với độ chính xác kép	`3.14159, -2.71828`
`char`	1 byte	Kiểu ký tự, dùng để lưu trữ các ký tự	`'A', 'b'`
`bool`	1 byte	Kiểu luận lý, chỉ có hai giá trị `true` hoặc `false`	`true, false`

Bảng 2: *Các kiểu dữ liệu cơ bản trong C++ với Kiểu giá trị*

Biểu thức trong C++

Biểu thức trong C++ là một tổ hợp các giá trị, biến, toán tử và hàm mà C++ có thể tính toán để trả về một giá trị. Biểu thức có thể đơn giản hoặc phức tạp. Các loại biểu thức cơ bản trong C++ bao gồm:

1. **Biểu thức số học (Arithmetic operators):** Là biểu thức có liên quan đến các phép toán số học.
   - `+`, `-`, `*`, `/`, `%` (phép cộng, trừ, nhân, chia, chia lấy dư).
   - Mô tả

Toán tử	Mô tả	Ví dụ
`+`	Cộng	`a + b`
`-`	Trừ	`a - b`
`*`	Nhân	`a * b`
`/`	Chia	`a / b` (phép chia nguyên nếu cả hai là số nguyên)
`%`	Chia lấy dư	`a % b` (chỉ áp dụng cho số nguyên)

Bảng 3: *Các toán tử số học trong C++*

2. **Biểu thức quan hệ (Comparison operators):** Là biểu thức để so sánh hai giá trị.
   - `==`, `!=`, `<`, `>`, `<=`, `>=` (so sánh bằng, không bằng, nhỏ hơn, lớn hơn, nhỏ hơn hoặc bằng, lớn hơn hoặc bằng).
   - Mô tả

Toán tử	Mô tả	Ví dụ
`==`	So sánh bằng	`a == b` (true nếu a bằng b)
`!=`	So sánh khác	`a != b` (true nếu a khác b)
`>`	Lớn hơn	`a > b` (true nếu a lớn hơn b)
`<`	Nhỏ hơn	`a < b` (true nếu a nhỏ hơn b)
`>=`	Lớn hơn hoặc bằng	`a >= b` (true nếu a lớn hơn hoặc bằng b)
`<=`	Nhỏ hơn hoặc bằng	`a <= b` (true nếu a nhỏ hơn hoặc bằng b)

Bảng 4: *Các toán tử quan hệ trong C++*

3. **Biểu thức luận lý (Logical operators):** Là biểu thức để thực hiện các phép toán luận lý (logic).
   - `&&`, `||`, `!` (phép AND, OR, NOT).
   - Mô tả

Toán tử	Mô tả	Ví dụ
`&&`	AND logic (và)	`(a > 0) && (b > 0)` (true nếu cả a và b lớn hơn 0)
`||`	OR logic (hoặc)	`(a > 0) || (b > 0)` (true nếu a hoặc b lớn hơn 0)
`!`	NOT logic (phủ định)	`!(a > 0)` (true nếu a không lớn hơn 0)

Bảng 5: *Các toán tử luận lý trong C++*

- **Chú ý:** Các toán tử luận lý `&&` (AND) và `||` (OR) trong C++ có tính **lười biếng** (short-circuit evaluation).
  - Với `A && B`: Nếu `A` là `false`, thì `B` sẽ không được đánh giá vì kết quả chắc chắn là `false`.
  - Với `A || B`: Nếu `A` là `true`, thì `B` sẽ không được đánh giá vì kết quả chắc chắn là `true`.

4. **Toán tử bitwise (Bitwise Operators)** Toán tử bitwise được sử dụng để thao tác trực tiếp trên các bit của số nguyên.
   - `&`, `|`, `~`, `▮`, `«`, `»`.
   - Mô tả

Toán tử	Mô tả	Ví dụ
`&`	AND bitwise (và từng bit)	`a & b`
`|`	OR bitwise (hoặc từng bit)	`a | b`
`~`	NOT bitwise (phủ định từng bit)	`~a`
`▮`	XOR bitwise (hoặc loại trừ từng bit)	`a ▮ b`
`«`	Dịch trái (shift left)	`a « 2` (dịch 2 bit trái)
`»`	Dịch phải (shift right)	`a » 2` (dịch 2 bit phải)

Bảng 6: *Các toán tử bitwise trong C++*

5. **Biểu thức gán (Assignment operators):** Là biểu thức dùng để gán giá trị cho biến.
   - `=`, `+=`, `-=`, `*=`, `/=`.
   - Mô tả

Toán tử	Mô tả	Ví dụ
`=`	Gán giá trị	`a = b` (Gán giá trị của `b` cho `a`)
`+=`	Cộng và gán	`a += b` (Tương đương `a = a + b`)
`-=`	Trừ và gán	`a -= b` (Tương đương `a = a - b`)
`*=`	Nhân và gán	`a *= b` (Tương đương `a = a * b`)
`/=`	Chia và gán	`a /= b` (Tương đương `a = a / b`)
`%=`	Chia lấy dư và gán	`a %= b` (Tương đương `a = a % b`)
`&=`	AND bitwise và gán	`a &= b` (Tương đương `a = a & b`)
`|=`	OR bitwise và gán	`a |= b` (Tương đương `a = a | b`)
`▮`	XOR bitwise và gán	`a ▮= b` (Tương đương `a = a ▮ b`)
`«=`	Dịch trái và gán	`a «= b` (Tương đương `a = a « b`)
`»=`	Dịch phải và gán	`a »= b` (Tương đương `a = a » b`)

Bảng 7: *Các toán tử gán trong C++*

6. **Toán tử đơn (Unary operators):** Là các toán tử chỉ làm việc với một toán hạng.
   - `+`, `-` (được dùng để thay đổi dấu của giá trị, ví dụ: `-a` là dấu âm của `a`).

- `++`, `-` (toán tử tăng và giảm, ví dụ: `a++` tăng giá trị của `a` lên 1).
- `!` (phép phủ định, ví dụ: `!true` sẽ trả về `false`).
- Ví dụ:

```
        int a = 5;
        a++;  // a = 6
        bool b = true;
        b = !b;  // b = false
```

Tương thích kiểu dữ liệu

Ép kiểu dữ liệu trong C++ là quá trình chuyển đổi kiểu dữ liệu này sang kiểu dữ liệu khác.

1. **Ép kiểu ngầm định (Implicit Casting):**

   C++ tự động thực hiện ép kiểu khi chuyển từ kiểu nhỏ sang kiểu lớn hơn hoặc khi cần kết hợp các kiểu dữ liệu trong phép toán.

```
int a = 10;
float b = a;  // tự động ép từ int sang float
```

2. **Ép kiểu tường minh (Explicit Casting):**

   Khi bạn cần ép kiểu rõ ràng, sử dụng cú pháp ép kiểu tường minh.

```
float f = 3.14;
int i = (int)f;  // Ép từ float sang int
```

Các kiểu ép kiểu phổ biến:

- Từ int sang float:

```
int a = 5;
float b = a;  // tự động ép kiểu
```

- Từ float sang int (mất phần thập phân):

```
float f = 3.14;
int i = (int)f;  // ép kiểu, mất phần thập phân
```

- Từ char sang int (lấy giá trị ASCII):

```
char c = 'A';
int x = c;  // x = 65
```

- Từ int sang char (chuyển số thành ký tự):

```
int i = 65;
char c = (char)i;  // c = 'A'
```

- Từ bool sang int: Giá trị true sẽ được chuyển thành 1, và giá trị false sẽ chuyển thành 0.

```
bool b = true;
int i = b;  // i = 1
cout << i;  // In ra 1
```

- Từ int sang bool: Giá trị khác 0 sẽ được chuyển thành true, còn giá trị 0 sẽ chuyển thành false.

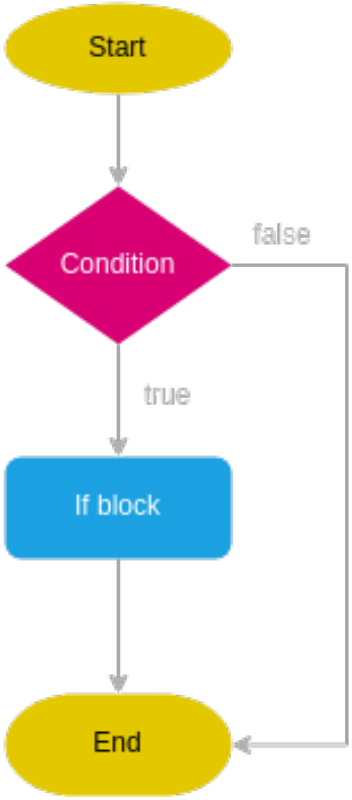
```
int i = 5;
bool b = i;  // b = true
cout << b;  // In ra 1 (true)

int j = 0;
bool b2 = j;  // b2 = false
cout << b2;  // In ra 0 (false)
```

1.5 Luồng điều kiển (Control flow Statements)

Câu lệnh if

Câu lệnh `if` trong C++ được sử dụng để kiểm tra một điều kiện và thực thi một khối mã nếu điều kiện đó đúng. Cấu trúc của câu lệnh `if` như sau:

```
// biểu thức
if( condition )
   statement;

// c++
if (điều_kiện) {
    // Thực thi mã nếu điều kiện đúng
}

// ví dụ
if (score_KTLT < 4) {
    cout << "BẠN BÈ XA LÁNH, NY ĐI THEO NGƯỜI KHÁC";
}
```

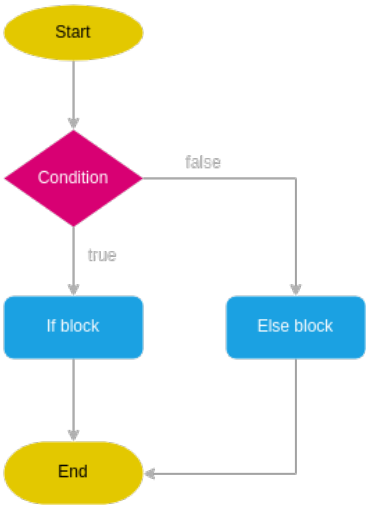


Hình 1: *Sơ đồ flowchart mô tả cách thức hoạt động của biểu thức điều kiện if*

Câu lệnh if-else

Câu lệnh `if-else` trong C++ được sử dụng để kiểm tra một điều kiện và thực thi một trong hai khối mã: một khối nếu điều kiện đúng và khối còn lại nếu điều kiện sai. Cấu trúc của câu lệnh `if-else` như sau:

```
// Biểu thức if-else cơ bản
if( condition )
   statement;
else
   statement2;

// Câu lệnh if-else trong C++
if (điều_kiện) {
    // Thực thi mã nếu điều kiện đúng
} else {
    // Thực thi mã nếu điều kiện sai
}

// Ví dụ thực tế
if (score_KTLT >= 4) {
    cout << "Chúc mừng bạn đã vượt qua môn!";
} else {
    cout << "Cố gắng lần sau nhé!";
}
```

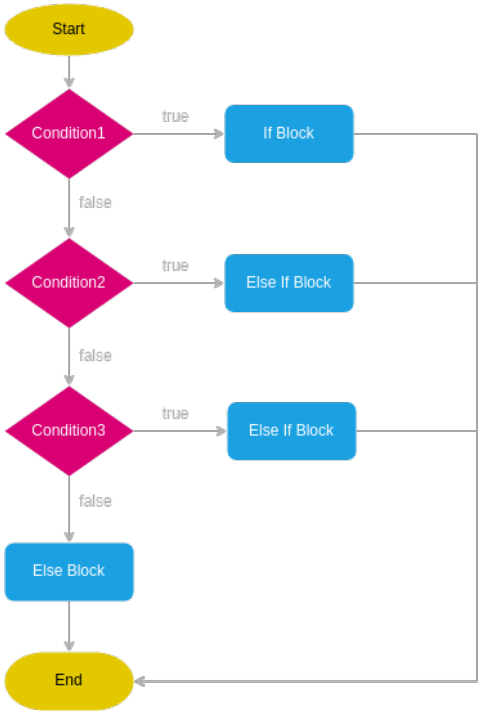


Hình 2: *Sơ đồ flowchart mô tả cách thức hoạt động của biểu thức điều kiện* `if-else`.

Câu lệnh if-elseif

Câu lệnh `if-elseif` trong C++ được sử dụng khi bạn muốn kiểm tra nhiều điều kiện khác nhau. Câu lệnh này cho phép kiểm tra các điều kiện theo thứ tự và thực thi một khối mã tương ứng với điều kiện đầu tiên đúng. Nếu không điều kiện nào đúng, khối `else` sẽ được thực thi. Cấu trúc của câu lệnh `if-elseif` như sau:

```
// Biểu thức if-elseif cơ bản
if( condition1 )
   statement1;
else if( condition2 )
   statement2;
else if( condition3 )
   statement3;
else
   statement4;

// Câu lệnh if-elseif trong C++
if (điều_kiện_1) {
    // Thực thi mã nếu điều kiện 1 đúng
} else if (điều_kiện_2) {
    // Thực thi mã nếu điều kiện 2 đúng
} else if (điều_kiện_3) {
    // Thực thi mã nếu điều kiện 3 đúng
} else {
    // Thực thi mã nếu không có điều kiện nào đúng
}

// Ví dụ thực tế
if (score_KTLT >= 8) {
    cout << "Xuất sắc!";
} else if (score_KTLT >= 6) {
    cout << "Khá!";
} else if (score_KTLT >= 4) {
    cout << "Trung bình!";
} else {
    cout << "Yếu!";
}
```

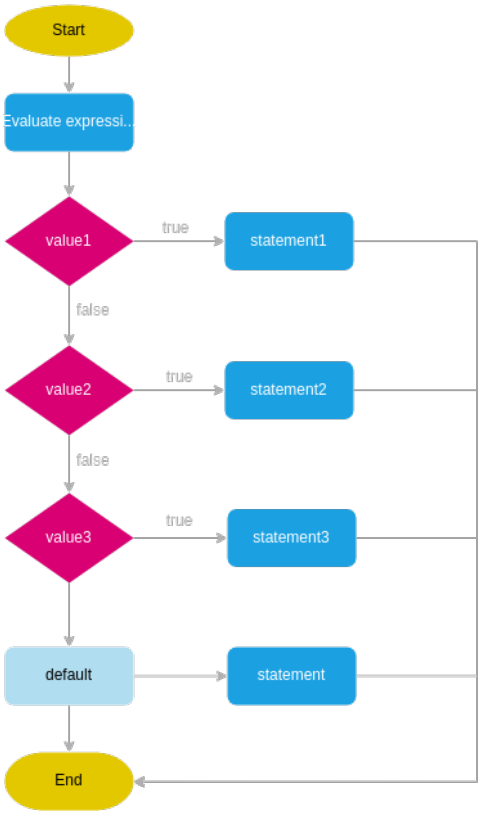


Hình 3: *Sơ đồ flowchart mô tả cách thức hoạt động của biểu thức điều kiện* `if-elseif`.

Câu lệnh switch-case

Câu lệnh `switch-case` trong C++ được sử dụng để kiểm tra giá trị của một biểu thức và so sánh nó với các giá trị có sẵn trong các trường hợp `case`. Nếu tìm thấy trường hợp phù hợp, câu lệnh tương ứng sẽ được thực thi. Nếu không có trường hợp nào khớp, phần `default` sẽ được thực thi (nếu có). Cấu trúc của câu lệnh `switch-case` như sau:

```
// Biểu thức switch-case cơ bản
switch(expression) {
    case value1:
        statement1;
        break;
    case value2:
        statement2;
        break;
    default:
        defaultStatement;
}

// Câu lệnh switch-case trong C++
switch (x) {
    case 1:
        // Thực thi mã nếu x = 1
        break;
    case 2:
        // Thực thi mã nếu x = 2
        break;
    default:
        // Thực thi mã nếu x không phải 1, 2
}

// Ví dụ thực tế
int day = 3;
switch (day) {
    case 1:
        cout << "Thứ Hai";
        break;
    case 2:
        cout << "Thứ Ba";
        break;
    case 3:
        cout << "Thứ Tư";
        break;
    default:
        cout << "Ngày không hợp lệ";
}
```

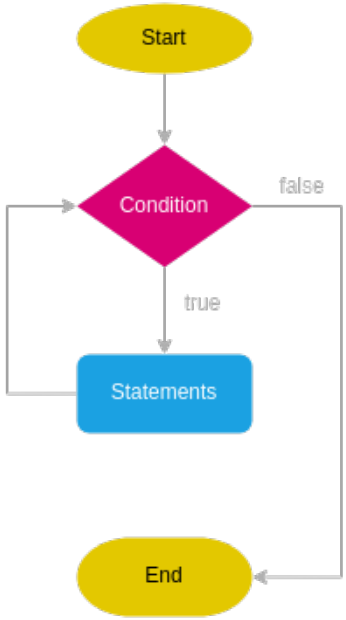


Hình 4: *Sơ đồ flowchart mô tả cách thức hoạt động của câu lệnh `switch-case`.*

Vòng lặp while

Vòng lặp `while` trong C++ được sử dụng để thực thi một khối lệnh liên tục miễn là điều kiện trong biểu thức `while` là đúng. Khi điều kiện trở thành sai, vòng lặp sẽ kết thúc. Cấu trúc của vòng lặp `while` như sau:

```
// Biểu thức vòng lặp while
while (condition) {
    statement;
}

// Vòng lặp while trong C++
while (điều_kiện) {
    // Thực thi mã nếu điều kiện đúng
}

// Ví dụ thực tế
int i = 0;
while (i < 3) {
    cout << "Vòng lặp thứ " << i + 1 << endl;
    i++;
}
```



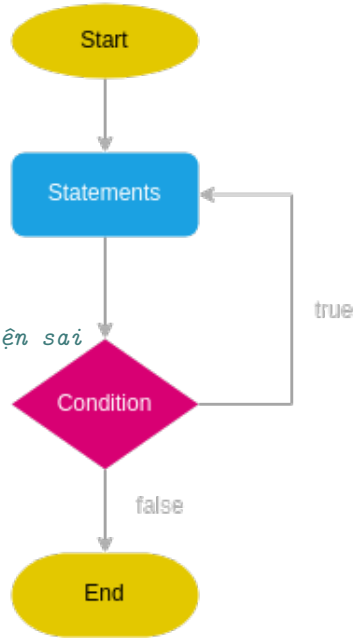
Hình 5: *Sơ đồ flowchart mô tả cách thức hoạt động của vòng lặp* `while`.

Vòng lặp do while

Vòng lặp `do while` trong C++ thực thi một khối lệnh ít nhất một lần và tiếp tục thực thi khi điều kiện trong biểu thức `while` là đúng. Khác với vòng lặp `while`, điều kiện trong `do while` được kiểm tra sau khi thực thi khối lệnh. Điều này có nghĩa là khối lệnh trong vòng lặp sẽ luôn được thực thi ít nhất một lần, bất kể điều kiện ban đầu có đúng hay không.

Cấu trúc của vòng lặp `do while` như sau:

```
// Biểu thức vòng lặp do while
do {
    statement;
} while (condition);

// Vòng lặp do while trong C++
do {
    // Thực thi mã
} while (điều_kiện); // Thoát khỏi vòng thực thi nếu kiều kiện sai

// Ví dụ thực tế
int i = 0;
do {
    cout << "Vòng lặp thứ " << i + 1 << endl;
    i++;
} while (i < 3);
```



Hình 6: *Sơ đồ flowchart mô tả cách thức hoạt động của vòng lặp* `do while`.

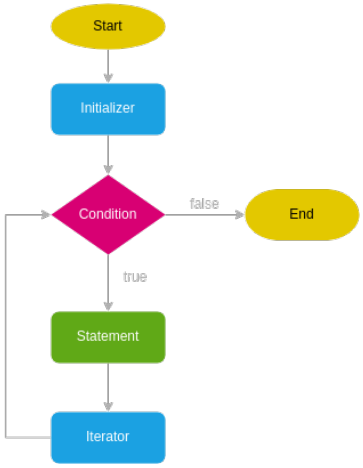
Cấu trúc vòng lặp `for`

Cấu trúc cơ bản của vòng lặp `for` trong C++ có ba phần chính: khởi tạo (initializer), điều kiện (condition) và bước tiến (iterator). Vòng lặp `for` sẽ tiếp tục thực hiện các câu lệnh bên trong khi điều kiện là `true`.

- **Khởi tạo (initializer)**: Đây là bước đầu tiên trong vòng lặp, nơi bạn khởi tạo các biến sử dụng trong vòng lặp. Ví dụ: `int i = 0;`
- **Điều kiện (condition)**: Điều kiện này sẽ được kiểm tra trước mỗi lần lặp. Nếu điều kiện là `true`, vòng lặp tiếp tục. Nếu là `false`, vòng lặp dừng lại. Ví dụ: `i < 5;`
- **Bước tiến (iterator)**: Sau mỗi lần lặp, phần này sẽ thực hiện thay đổi biến lặp (tăng hoặc giảm). Ví dụ: `i++` (tăng `i` lên 1 sau mỗi lần lặp).
- **Statement**: Đây là phần chứa các câu lệnh được thực thi trong mỗi lần lặp.

Cấu trúc vòng lặp `for` có dạng như sau:

```cpp
// Biểu thức vòng lặp for
for (initializer; condition; iterator) {
    statement;
}

// Vòng lặp do while trong C++
for (biểu_thức_1; điều_kiện; biểu_thức_2) {
    // Thực thi nếu điều kiện đúng
}

// Ví dụ thực tế
for (int i = 0; i < 5; i++) {
    cout << "Giá trị của i là: " << i << endl;
}
```



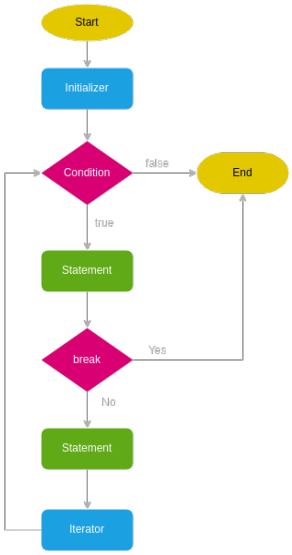
Hình 7: *Sơ đồ flowchart mô tả cách thức hoạt động của vòng lặp `for`.*

Câu lệnh `break` trong vòng lặp `for` và `while`

Trong C++, câu lệnh **`break`** được sử dụng để dừng vòng lặp ngay lập tức khi một điều kiện nhất định được thỏa mãn. Khi gặp **`break`**, chương trình sẽ thoát khỏi vòng lặp mà không thực hiện các lần lặp còn lại.

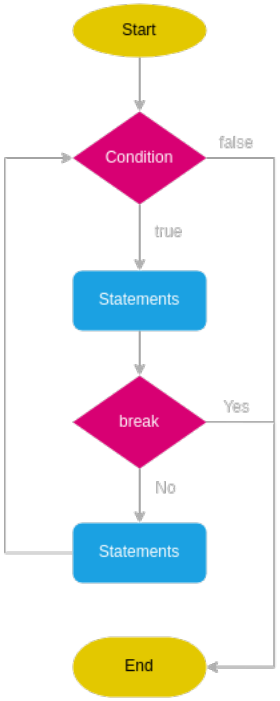
- **Chức năng**: Thoát khỏi vòng lặp ngay khi gặp điều kiện phù hợp.
- **Ứng dụng**: Thường được sử dụng khi tìm kiếm một giá trị thỏa mãn điều kiện và không cần tiếp tục vòng lặp nữa.

```cpp
// Câu lệnh break trong vòng lặp for
int i = 0;
for (; i < 10; i++) {
    if (i == 5) {
        break; // Dừng vòng lặp khi i == 5
    }
    cout << "Giá trị của i: " << i << endl;
}
cout << "Giá trị của i sau for: " << i << endl;

// Kết quả:
// Giá trị của i: 0
// Giá trị của i: 1
// Giá trị của i: 2
// Giá trị của i: 3
// Giá trị của i: 4
// Giá trị của i sau for: 5
```



Hình 8: *Sơ đồ flowchart mô tả cách thức hoạt động của câu lệnh **`break`** trong vòng lặp **`for`**.*

```cpp
// Câu lệnh break trong vòng lặp while
int i = 0;
while (i < 10) {
    if (i == 5) {
        break; // Dừng vòng lặp khi i == 5
    }
    cout << "Giá trị của i: " << i << endl;
    i++;
}
cout << "Giá trị của i sau while: " << i << endl;

// Kết quả:
// Giá trị của i: 0
// Giá trị của i: 1
// Giá trị của i: 2
// Giá trị của i: 3
// Giá trị của i: 4
// Giá trị của i sau while: 5
```



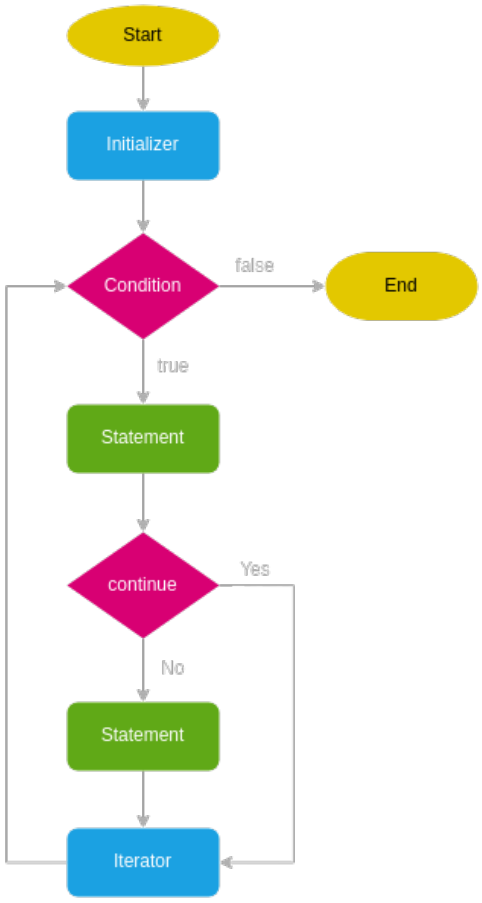
Hình 9: *Sơ đồ flowchart mô tả cách thức hoạt động của câu lệnh **`break`** trong vòng lặp **`while`**.*

Câu lệnh `continue` trong vòng lặp `for` và `while`

Trong C++, câu lệnh **`continue`** được sử dụng để bỏ qua phần còn lại của lần lặp hiện tại và tiếp tục với lần lặp tiếp theo. Khi gặp **`continue`**, chương trình sẽ bỏ qua các lệnh còn lại trong lần lặp đó và chuyển sang lần lặp kế tiếp.

- **Chức năng**: Bỏ qua các lệnh sau **`continue`** trong vòng lặp hiện tại và tiếp tục với lần lặp kế tiếp.
- **Ứng dụng**: Thường được sử dụng khi có điều kiện để bỏ qua một số giá trị không cần xử lý trong vòng lặp.

```cpp
// Câu lệnh continue trong vòng lặp for
for (int i = 0; i < 10; i++) {
    if (i == 5) {
        continue; // Bỏ qua khi i == 5
    }
    cout << "Giá trị của i: " << i << endl;
}

// Kết quả:
// Giá trị của i: 0
// Giá trị của i: 1
// Giá trị của i: 2
// Giá trị của i: 3
// Giá trị của i: 4
// Giá trị của i: 6
// Giá trị của i: 7
// Giá trị của i: 8
// Giá trị của i: 9
```



Hình 10: *Sơ đồ flowchart mô tả cách thức hoạt động của câu lệnh* ***`continue`*** *trong vòng lặp* ***`for`***.

Toán tử 3 ngôi (ternary operator)

Toán tử ba ngôi (`?:`) trong C++ là một toán tử điều kiện rút gọn giúp viết các biểu thức điều kiện ngắn gọn hơn so với câu lệnh `if-else`. Nó có cú pháp như sau:

```
condition ? expression1 : expression2;
```

- `condition`: Biểu thức điều kiện trả về giá trị `true` hoặc `false`.
- Nếu `condition` là `true`, `expression1` sẽ được thực thi.
- Nếu `condition` là `false`, `expression2` sẽ được thực thi.

Toán tử ba ngôi thay thế `if-else`

```cpp
int main() {
    int a = 10, b = 20;

    // Sử dụng toán tử ba ngôi để tìm số lớn hơn
    int max = (a > b) ? a : b;

    cout << "Số lớn hơn là: " << max << endl; // Output: Số lớn hơn là: 20
}
```

Lồng nhiều toán tử ba ngôi

```cpp
int main() {
    int a = 10, b = 20, c = 15;

    // Tìm số lớn nhất trong ba số
    int max = (a > b) ? ((a > c) ? a : c) : ((b > c) ? b : c);

    cout << "Số lớn nhất là: " << max << endl;

    return 0;
}
```

1.6 Bài Tập

1. Truy cập https://www.overleaf.com/read/gfkqtjmpctfq#e1390f
2. Võ trang https://www.overleaf.com/project Copy latex vừa xem ra ngoài thành latex riêng của bạn vì anh chỉ share ở dạng view
3. Đổi tên trang latex mới theo Nick name của bạn trong nhóm discord
4. Đánh đáp án của bạn vào (bạn nào chưa biết latex tự tìm hiểu nha khá nhanh)
5. Chỉ **include{TASK 2 NMLT và cách debug/Các Khái niệm cơ bản trong c++}**
6. chọn **nút share** trên thanh công cụ > chọn **Turn on link sharing** > copy link có thể view
7. Gửi Link nộp bài vào các kênh sau tùy vào nhóm của bạn
   - CS https://discord.com/channels/1334472759945990184/1337244525822742579
   - CE https://discord.com/channels/1334472759945990184/1337244684988448788
   - DS https://discord.com/channels/1334472759945990184/1337244698078871562