Ngôn ngữ lập trình C++

CHƯƠNG I. CƠ BẢN VỀ NGÔN NGỮ LẬP TRÌNH C++

ThS. Phạm Đức Cường

cuongpd@ptit.edu.vn

Posts and Telecommunications Institute of Technology

Học viện Công nghệ Bưu chính Viễn thông

2024

Nội dung chương

- **Phần 1:** Thuật toán và chương trình
- **Phần 2:** Giới thiệu ngôn ngữ C++
- **Phần 3:** Lập trình C++

Lập trình máy tính:

- Gọi tắt là lập trình (programming)
- Cài đặt một hoặc nhiều thuật toán bằng ngôn ngữ lập trình để tạo ra chương trình máy tính

Thuật toán:

- Tập hợp (dãy) hữu hạn các chỉ thị (hành động) được định nghĩa rõ ràng nhằm giải quyết một bài toán cụ thể

- **Tính chính xác:** quá trình tính toán hay thao tác máy tính thực hiện là chính xác
- **Tính rõ ràng:** các câu lệnh minh bạch được sắp xếp theo một thứ tự nhất định
- **Tính khách quan:** có thể được viết bởi nhiều lập trình viên trên nhiều máy tính nhưng kết quả phải như nhau
- **Tính khả dụng:** áp dụng được cho những lớp bài toán có đầu vào tương tự nhau
- **Tính kết thúc:** thuật toán phải kết thúc sau hữu hạn bước tính toán


Xác định vấn đề
- bài toán
Lựa chọn
phương pháp giải
Xây dựng
thuật toán/ thuật giải
Cài đặt
chương trình
Hiệu chỉnh
chương trình
Thực hiện
chương trình
Biểu diễn bằng:
• Ngôn ngữ tự nhiên
• Lưu đồ - Sơ đồ khối
• Mã giả
Lỗi cú pháp
Lỗi ngữ nghĩa

Xây dựng thuật toán giải phương trình bậc nhất: $ax + b = 0$ với $a, b \in \mathbb{R}$

Đầu vào: a, b thuộc R
Đầu ra: nghiệm phương trình ax + b = 0

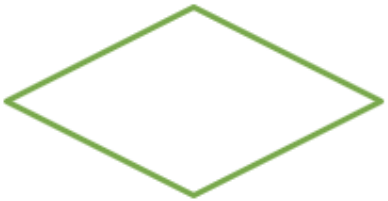
- Nếu a = 0
  - b = 0 thì phương trình có nghiệm bất kì.
  - b ≠ 0 thì phương trình vô nghiệm.
- Nếu a ≠ 0
  - Phương trình có nghiệm duy nhất x = -b/a

Đầu vào: a, b thuộc R
Đầu ra: nghiệm phương trình ax + b = 0

1. Nhập 2 số thực a và b.
2. Nếu a = 0 thì
    - 2.1. Nếu b = 0 thì
        - 2.1.1. Phương trình vô số nghiệm
        - 2.1.2. Kết thúc thuật toán.
    - 2.2. Ngược lại
        - 2.2.1. Phương trình vô nghiệm.
        - 2.2.2. Kết thúc thuật toán.
3. Ngược lại
    - 3.1. Phương trình có nghiệm.
    - 3.2. Giá trị của nghiệm đó là x = -b/a
    - 3.3. Kết thúc thuật toán.

Khối giới hạn
Chỉ thị bắt đầu và kết thúc.

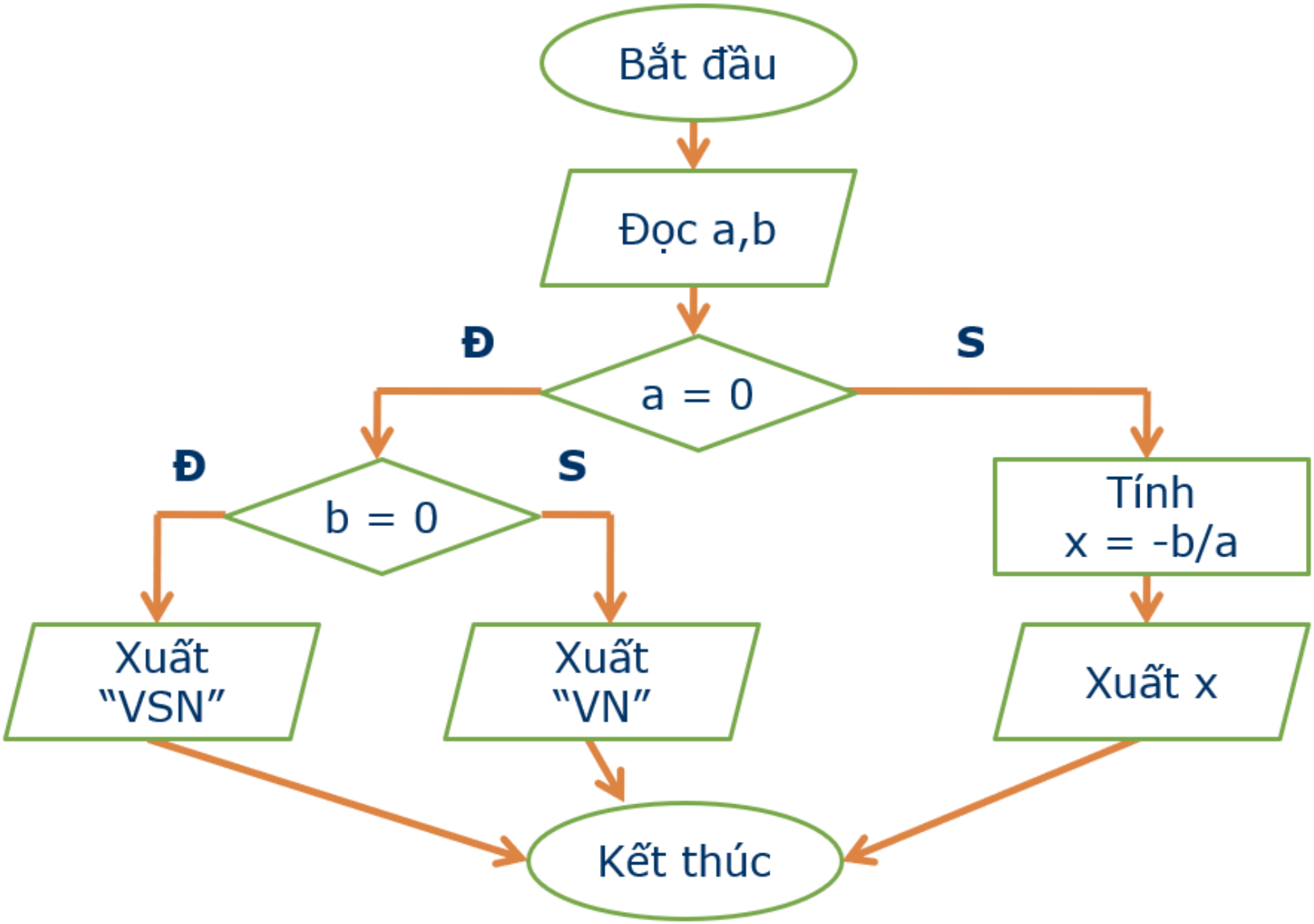
Khối vào ra
Nhập/Xuất dữ liệu.



Khối lựa chọn
Tùy điều kiện sẽ rẽ nhánh.

Khối thao tác
Ghi thao tác cần thực hiện.

Đường đi
Chỉ hướng thao tác tiếp theo.


Bắt đầu
Đọc a,b
a = 0
Đ
S
b = 0
Đ
S
Tính
x = -b/a
Xuất
"VSN"
Xuất
"VN"
Xuất x
Kết thúc

Vay mượn ngôn ngữ lập trình để biểu diễn thuật toán

```
Đầu vào: a, b thuộc R
Đầu ra: nghiệm phương trình ax + b = 0
If a = 0 Then
Begin
      If b = 0 Then
            Xuất "Phương trình vô số nghiệm"
      Else
            Xuất "Phương trình vô nghiệm"
End
Else
      Xuất "Phương trình có nghiệm x = -b/a"
```

```c
#include <stdio.h>

int main() {
    double a, b, root;

    printf("Nhap he so a: ");
    scanf("%lf", &a);
    printf("Nhap he so b: ");
    scanf("%lf", &b);

    if (a == 0) {
        if (b == 0) {
            printf("Phuong trinh co vo so nghiem.\n");
        } else {
            printf("Phuong trinh vo nghiem.\n");
        }
    } else {
        root = -b / a;
        printf("Nghiem cua phuong trinh la x = %.2lf\n", root);
    }

    return 0;
}
```

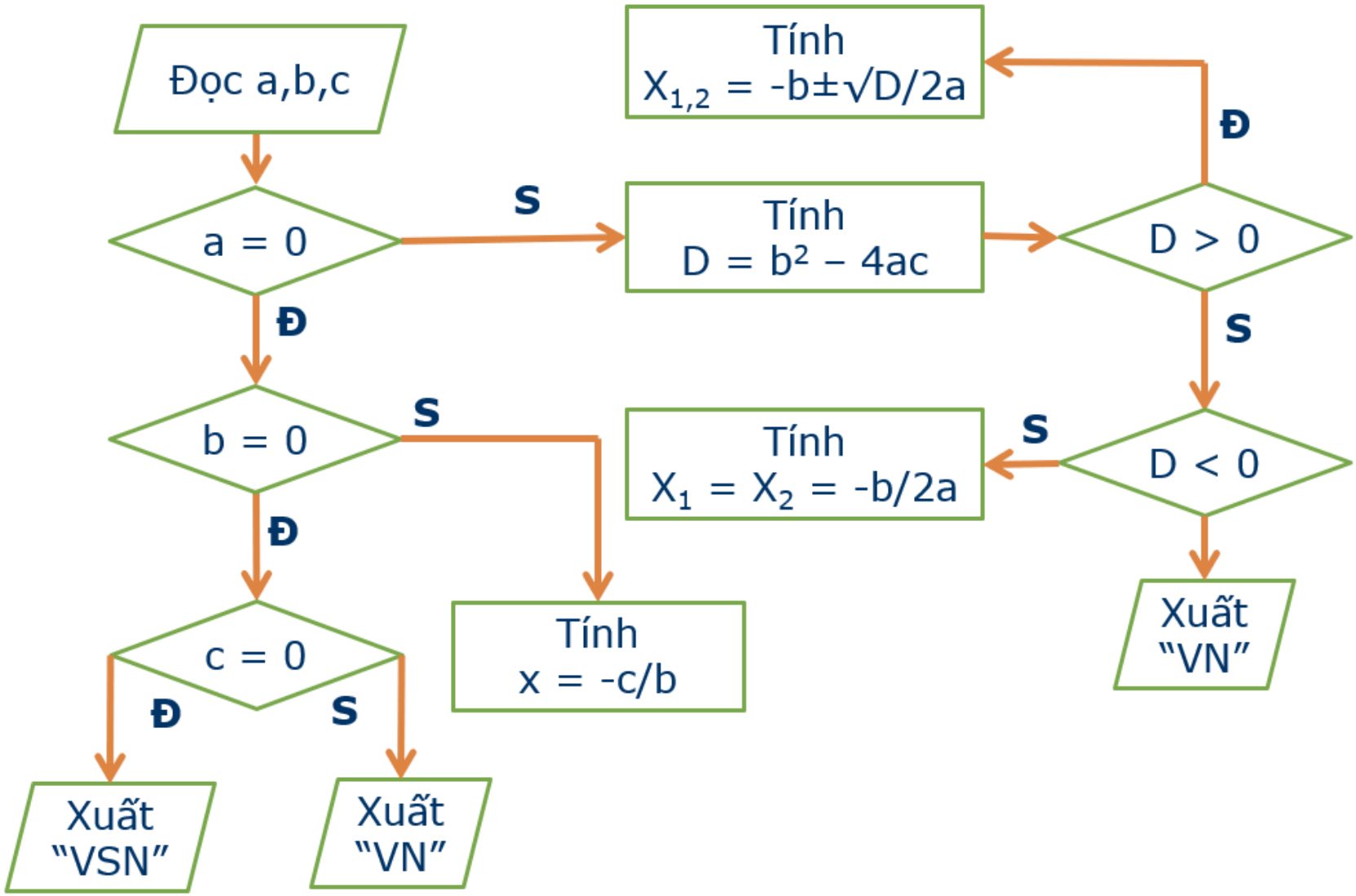
Mã nguồn 1: Mã nguồn C giải phương trình bậc nhất

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

int main() {
    double a, b, root;

    cout << "Nhap he so a: ";
    cin >> a;
    cout << "Nhap he so b: ";
    cin >> b;

    if (a == 0) {
        if (b == 0) {
            cout << "Phuong trinh co vo so nghiem.\n";
        } else {
            cout << "Phuong trinh vo nghiem.\n";
        }
    } else {
        root = -b / a;
        cout << "Nghiem cua phuong trinh la x = " << root << endl;
    }

    return 0;
}
```

Mã nguồn 2: Mã nguồn C++ giải phương trình bậc nhất

Cho phương trình bậc hai: $ax^2 + bx + c = 0$ với $a, b, c \in \mathbb{R}$

- Vẽ sơ đồ khối giải phương trình
- Cài đặt thuật toán giải phương trình với C


Đọc a,b,c
a = 0
S
Đ
b = 0
S
Đ
c = 0
Đ
S
Xuất "VSN"
Xuất "VN"
Tính x = -c/b
Tính D = b² – 4ac
D > 0
Đ
S
Tính X1,2 = -b±√D/2a
D < 0
S
Tính X1 = X2 = -b/2a
Xuất "VN"

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

int main() {
    double a, b, c;
    double delta, root1, root2;

    printf("Nhap he so a: ");
    scanf("%lf", &a);
    printf("Nhap he so b: ");
    scanf("%lf", &b);
    printf("Nhap he so c: ");
    scanf("%lf", &c);

    if (a == 0) {
        if (b == 0) {
            if (c == 0) {
                printf("Phuong trinh co vo so nghiem.\n");
            } else {
                printf("Phuong trinh vo nghiem.\n");
            }
        } else {
            printf("Phuong trinh co mot nghiem: %.2f\n", -c / b);
        }
    } else {
        delta = b * b - 4 * a * c;
        if (delta > 0) {
```

```
28              root1 = (-b + sqrt(delta)) / (2 * a);
29              root2 = (-b - sqrt(delta)) / (2 * a);
30              printf("Hai nghiem phan biet: %.2f va %.2f\n", root1,
    root2);
31          } else if (delta == 0) {
32              root1 = -b / (2 * a);
33              printf("Nghiem kep: %.2f\n", root1);
34          } else {
35              printf("Phuong trinh vo nghiem.\n");
36          }
37      }
38
39      return 0;
40  }
```

Mã nguồn 3: Mã nguồn C giải phương trình bậc hai

Phân tích thuật toán:

- Tính đúng
- Tính đơn giản
- Không gian bộ nhớ
- Thời gian chạy (Độ phức tạp)

Thực nghiệm:

- Phụ thuộc vào nhiều yếu tố như phần cứng, hệ điều hành, môi trường thực thi, tối ưu hóa bộ nhớ, cạnh tranh tài nguyên, ...
- Khó khăn và tốn chi phí lựa chọn bộ dữ liệu đủ đa dạng
- Khi kích thước dữ liệu rất lớn, thời gian thực nghiệm có thể trở nên rất dài và khó đo lường chính xác

Xấp xỉ:

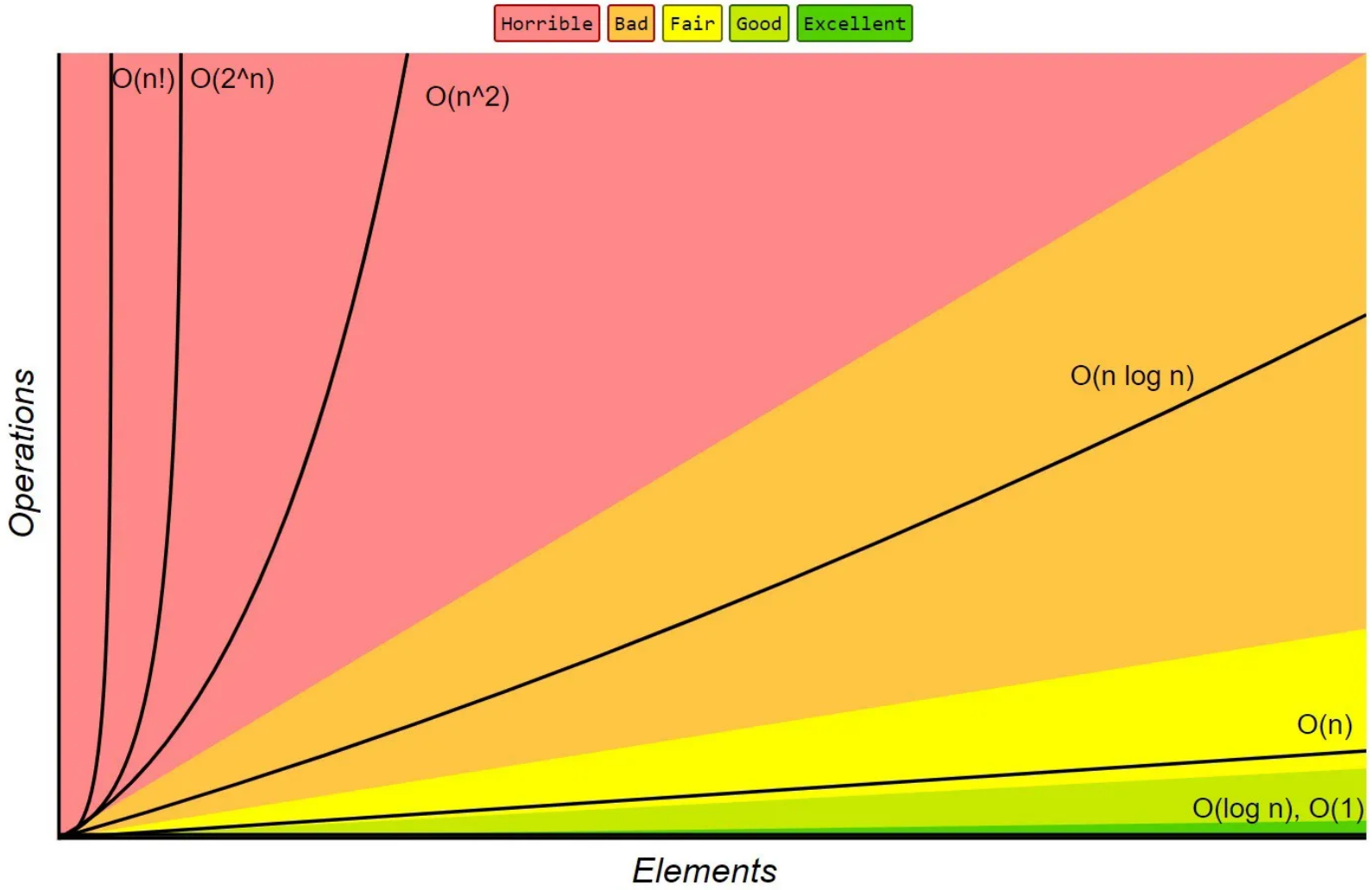
- Đơn giản hóa quá trình phân tích thuật toán, tập trung vào yếu tố chính và loại bỏ chi tiết không quan trọng (hằng số nhỏ, yếu tố phụ)
- Cung cấp cái nhìn tổng quát về hiệu suất của thuật toán mà không cần phải quan tâm đến yếu tố cụ thể như phần cứng, môi trường thực thi, dữ liệu đầu vào, ...
- So sánh được các thuật toán theo một tiêu chuẩn chung
- Giúp lập trình viên hiểu được tác động của quyết định thiết kế đối với hiệu suất của thuật toán

Tính chất:

- Tập trung vào phần chính yếu của sự thay đổi theo kích thước đầu vào, bỏ qua yếu tố không quan trọng như hằng số và yếu tố phụ
- Mô tả độ phức tạp trong trường hợp tồi tệ nhất, đảm bảo thuật toán không vượt quá mức độ phức tạp nhất định

Một số kết quả quan trọng:

- Hàm đa thức:
  - $f(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$
  - Độ phức tạp là $O(x^n)$
- Hàm giai thừa:
  - $f(n) = n!$
  - Độ phức tạp là $O(n!)$
- Logarit của hàm giai thừa:
  - $f(n) = \log(n!)$
  - Độ phức tạp là $O(n \log n)$
- Hàm điều hòa:
  - $H(n) = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}$
  - Độ phức tạp là $O(\log n)$



Hình 1: Biểu đồ độ phức tạp của Big-O

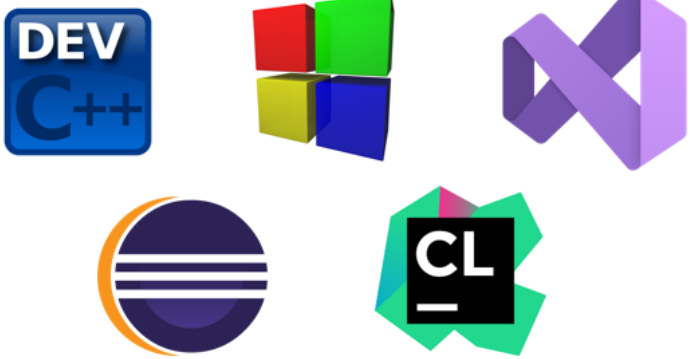
- C++ là một ngôn ngữ lập trình bậc trung được phát triển bởi Bjarne Stroustrup vào đầu những năm 1980
- C++ đã trở thành một trong những ngôn ngữ lập trình quan trọng nhất, được sử dụng rộng rãi trong phát triển phần mềm hệ thống, phần mềm ứng dụng, trò chơi, và nhiều lĩnh vực khác

Tiêu chí so sánh	Trình biên dịch (Compiler)	Trình thông dịch (Interpreter)
Đầu vào	Toàn bộ chương trình	Chỉ một dòng code
Đầu ra	Mã đối tượng trung gian	Không tạo ra mã đối tượng trung gian
Cơ chế hoạt động	Biên dịch hoàn thành trước khi thực thi	Thông dịch và thực thi là đồng thời
Tốc độ	Nhanh hơn	Chậm hơn
Bộ nhớ	Yêu cầu bộ nhớ nhiều hơn	Đòi hỏi ít bộ nhớ hơn
Errors	Hiển thị cùng lúc tất cả lỗi sau khi biên dịch	Hiển thị lỗi của dòng đầu tiên gặp phải
Phát hiện lỗi	Rất khó khăn	Tương đối dễ
Ngôn ngữ đại diện	C, C++, Go	PHP, Python, Ruby

Bảng 1: So sánh giữa trình biên dịch và trình thông dịch

IDE

- Dev-C++
- Code::Blocks
- Visual Studio
- Eclipse
- CLion
- ...



Code Editor

- Visual Studio Code
- Sublime Text
- Notepad++
- Vim
- ...

Website:

- https://www.onlinegdb.com/online_c++_compiler
- https://ideone.com
- https://www.programiz.com/cpp-programming/online-compiler/

Mobile:

- Coding C++ (Android)
- Cxxdroid (Android)
- Mobile C [C/C++ Compiler] (Android, IOS)

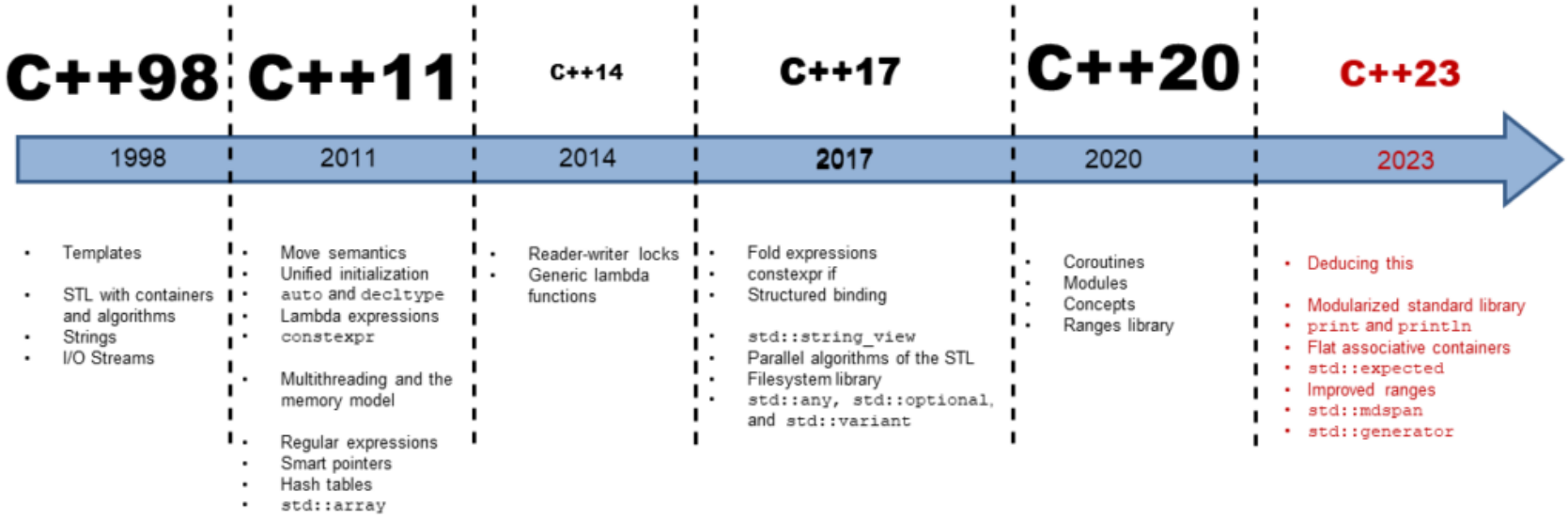
Các thành phần chính:

- Trình soạn thảo mã nguồn (Code Editor)
- Trình biên dịch hoặc thông dịch
- Trình gỡ lỗi
- Quản lý dự án
- Hỗ trợ tích hợp với hệ thống quản lý phiên bản
- Công cụ phân tích mã

Lợi ích:

- Nâng cao hiệu suất làm việc
- Giảm lỗi
- Dễ dàng tổ chức và quản lý dự án

C++ là phiên bản mở rộng của C, bổ sung rất nhiều tính năng hữu ích và không ngừng nâng cấp



Hình 2: Tính năng trong các phiên bản C++

Ngôn ngữ bậc cao	Ngôn ngữ bậc trung	Ngôn ngữ bậc thấp
Đặc điểm: - Mức độ trừu tượng cao, gần gũi với ngôn ngữ tự nhiên, giúp lập trình viên dễ đọc, viết và bảo trì mã nguồn - Không phụ thuộc vào phần cứng và có thể chạy trên nhiều nền tảng khác nhau	**Đặc điểm:** - Cung cấp sự cân bằng giữa trừu tượng hóa và kiểm soát phần cứng - Cho phép cả thao tác phần cứng và cấu trúc chương trình phức tạp.	**Đặc điểm:** - Liên hệ chặt chẽ với phần cứng của máy tính, ít hoặc không có sự trừu tượng hóa và cụ thể cho từng loại máy tính - Sử dụng để viết mã trực tiếp cho phần cứng cụ thể, cho phép kiểm soát chi tiết hoạt động của hệ thống.
Ví dụ: Python, Java, PHP, JavaScript, C#	**Ví dụ:** C, C++, Rust	**Ví dụ:** Assembly, Mã máy

- **Hướng đối tượng:** C++ coi tất cả dữ liệu là đối tượng
- **Linh hoạt:**
  - Hỗ trợ nhiều mô hình lập trình: thủ tục, hướng đối tượng, tổng quát
  - Quản lý bộ nhớ thủ công hoặc tự động
  - Bộ thư viện chuẩn phong phú
  - Khả năng mở rộng thông qua macro, template và thư viện mới

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

template <typename T> T myMax(T x, T y) {
    return (x > y) ? x : y;
}

int main() {
    cout << myMax<int>(6, 8) << endl; // Output: 8
    cout << myMax<double>(6.6, 8.8) << endl; // Output: 8.8
    cout << myMax<char>('f', 'h') << endl; // Output: h

    return 0;
}
```

Mã nguồn 4: Ví dụ lập trình tổng quát trong C++

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

#define PI 3.1416
#define AREA(r) r * r * PI

#ifndef PI
    double PI = 3.1416;
#endif

#ifndef AREA
double AREA(double r) {
    return r * r * PI;
}
#endif

int main() {
    double radius = 1;
    cout << "Area is " << AREA(radius); // Output: Area is 3.1416

    return 0;
}
```

Mã nguồn 5: Ví dụ sử dụng macro trong C++

- **Lập trình mô-đun:** chương trình C++ có thể được tạo thành từ nhiều tệp mã nguồn riêng lẻ được biên dịch riêng biệt và sau đó liên kết lại với nhau, giúp tiết kiệm thời gian

Hình 3: Ví dụ lập trình mô-đun

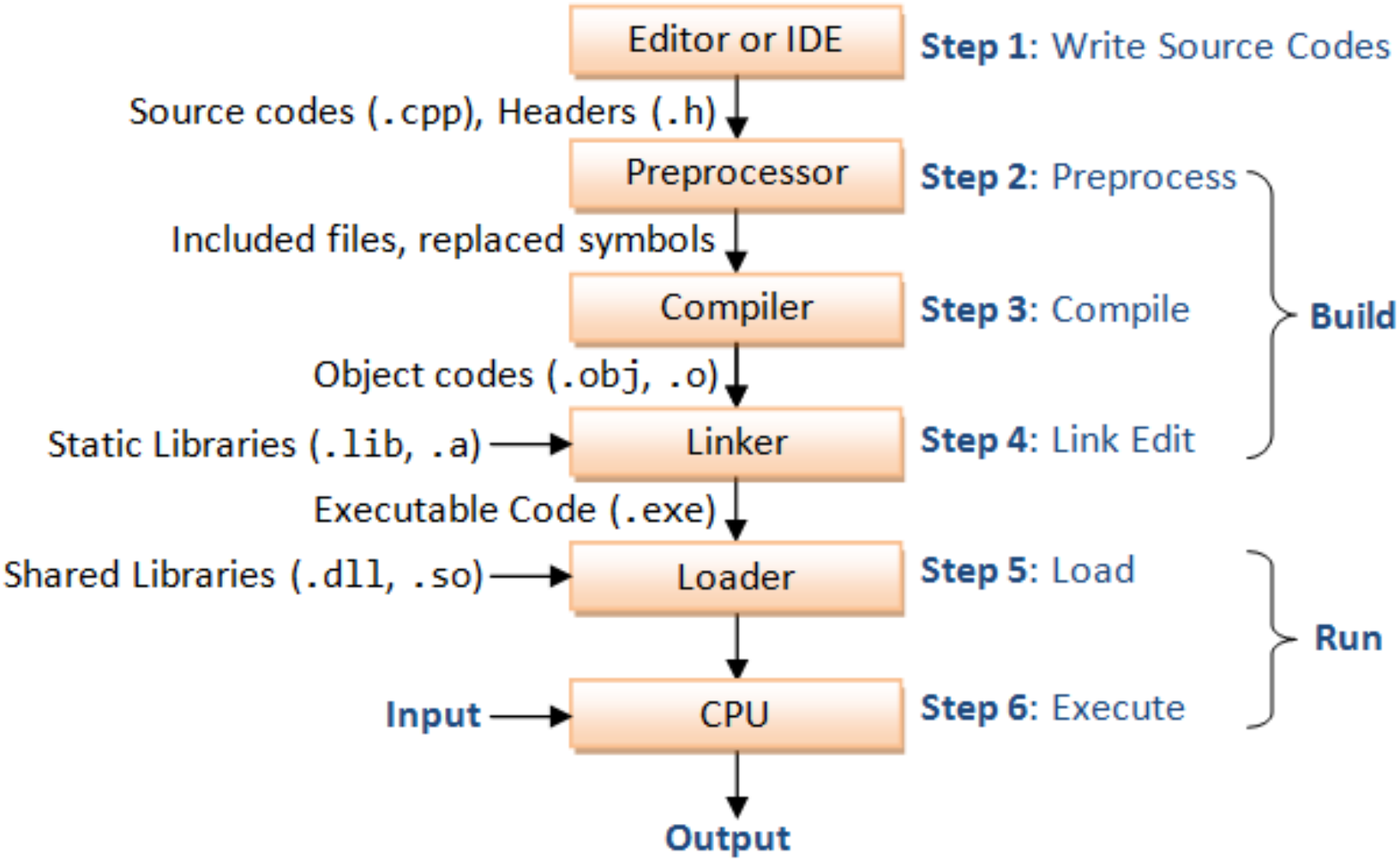
- **Di động:** có thể biên dịch cùng một mã nguồn C++ trên hầu hết mọi loại máy tính và hệ điều hành
- **Tốc độ:** biên dịch và thực thi hiệu quả nhờ tính chất kết hợp giữa ngôn ngữ lập trình bậc cao và bậc thấp
- **Tương thích với C:** bất kỳ mã nào được viết bằng ngôn ngữ C đều có thể dễ dàng được đưa vào chương trình C++

```cpp
#include <stdio.h>
#include <iostream>
using namespace std;

#define d "D23"

int main() {
    printf("Hello, PTIT!\n"); // Output: Hello, PTIT!
    cout << "Welcome " << d << "!!!"; // Output: Welcome D23!!!

    return 0;
}
```

Mã nguồn 6: Ví dụ sử dụng C trong C++

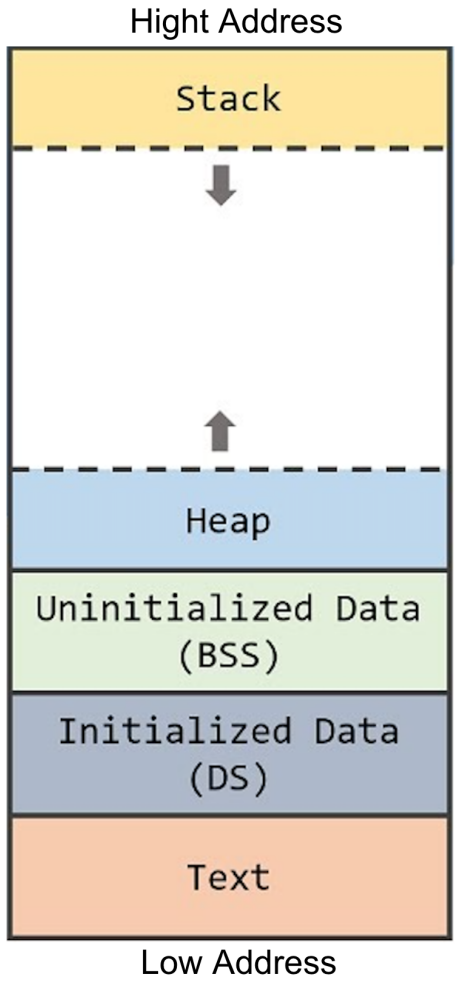
- **Phát triển phần mềm hệ thống:**
  - **Hệ điều hành:**
    - Windows
    - Appple macOS (một phần)
    - Linux (một phần)
  - **Trình biên dịch:**
    - GCC (GNU Compiler Collection)
    - Clang/LLVM
    - → Kỹ thuật Bootstrapping
  - **IDE/Code Editor:**
    - Microsoft Visual Studio
    - Code::Blocks
    - Notepad++
    - Sublime Text

- **Cơ sở dữ liệu:**
  - MySQL
  - MongoDB
- **Trò chơi:**
  - Warcraft
  - Counter-Strike
- **Phần mềm tiện ích:**
  - Photoshop
  - Unikey
  - Microsoft Office
  - Google Chrome
  - Mozilla Firefox
  - WinRAR
  - Autodesk Maya



Hình 4: Sáu bước biên dịch và thực thi mã nguồn C++

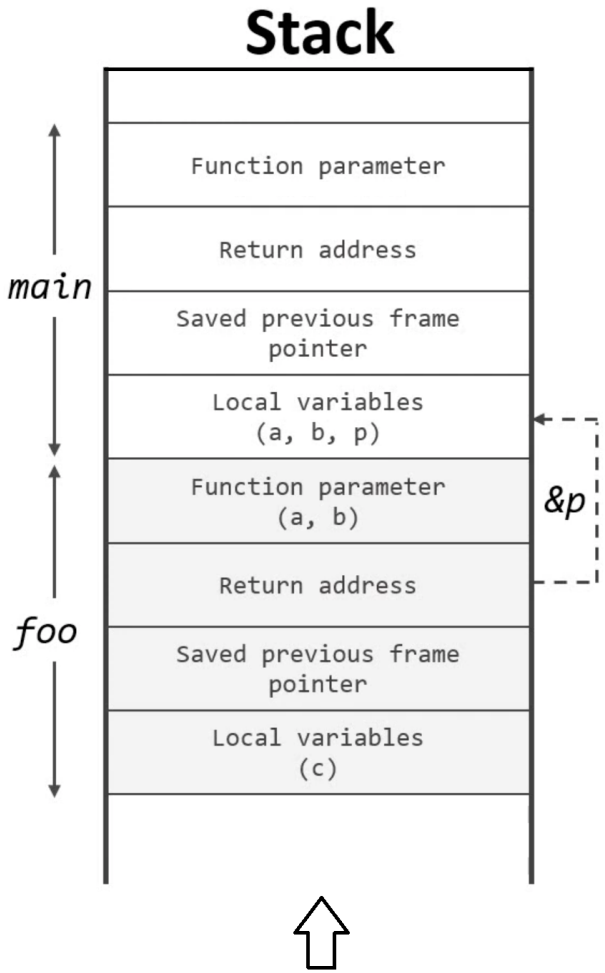
1. **Text Segment (Code Segment):** chứa đoạn mã lệnh của chương trình
2. **Initialized Data Segment (Data Segment):** lưu trữ global variables, static variables được khởi tạo với giá trị khác mặc định
3. **Uninitialized Data Segment (Block Started by Symbol):** lưu trữ global variables, static variables không được khởi tạo hoặc khởi tạo với giá trị mặc định
4. **Heap (Dynamic Memory Allocation):** vùng nhớ kích thước có thể thay đổi, cho phép cấp phát và giải phóng bộ nhớ linh hoạt theo nhu cầu của chương trình (cấp phát động)



5. Stack (Automatic Variable Storage):

- Vùng nhớ kích thước cố định, được cấp phát tự động
- Cấu trúc LIFO (Last In First Out)
- Khi thực thi, các Function Frame được gọi và push vào stack

```
1  int foo(int a, int b) {
2      int c = 2024;
3
4      return c + a * b;
5  }
6
7  int main() {
8      int a = 2024, b = 2024, p;
9      p = foo(a, b);
10
11     return 0;
12 }
```



```cpp
// Comment
/* Comment
multiple lines */

#include <iostream> // Access to I/O library
using namespace std; // Select namespace standard

int main() { // Special fuction main()
    cout << "Hello PTIT!" << endl; // Output a string with endline

    return 0; // Status return code (0 means OK)
}
```

Mã nguồn 7: Ví dụ chương trình C++

1. **Phần chú thích:**
   - Là phần không bắt buộc nhưng rất quan trọng để giải thích mã nguồn
   - Chú thích có thể nằm ở bất kỳ đâu trong chương trình

2. **Chỉ thị tiền xử lý:**
   - Chỉ thị tiền xử lý bắt đầu bằng #, ví dụ như **#include** bao gồm thư viện cần thiết cho chương trình, **#define** định nghĩa hằng số hoặc macro
   - Được xử lý trước khi biên dịch bắt đầu (bước Preprocess)
   - Một chương trình C++ có thể không có chỉ thị tiền xử lý
   - Thư viện của C khi sử dụng nên được **#include** với phiên bản C++ (Ví dụ: **stdio.h** $\rightarrow$ **cstdio**, **stdlib.h** $\rightarrow$ **cstdlib**, **math.h** $\rightarrow$ **cmath**, ...)
3. **Phần khai báo toàn cục:**
   - Khai báo biến, hằng số, hàm hoặc lớp được sử dụng trong toàn bộ chương trình
   - Khai báo này nằm ngoài tất cả các hàm
   - **using namespace** được khai báo ở đây, cho phép sử dụng các thành phần trong một namespace mà không cần phải chỉ định tên đầy đủ (Ví dụ: **using namespace std;** để sử dụng **cout** thay vì **std::cout**)

4. **Hàm main():**
   - Là hàm chính và điểm bắt đầu thực thi của chương trình
   - Mọi chương trình C++ đều phải có một hàm **main()**
5. **Phần khai báo trong hàm main():** khai báo biến và đối tượng được sử dụng trong phạm vi hàm này
6. **Câu lệnh:**
   - Là phần chính của hàm **main()** (và các hàm khác nếu có)
   - Nơi viết những câu lệnh thực thi để thực hiện logic của chương trình
7. **Hàm và lớp bổ sung:**
   - Nơi định nghĩa các hàm và lớp bổ sung (nếu có)
   - Có thể được định nghĩa ngay trong tệp chứa main hoặc định nghĩa trong tệp khác và **#include** để sử dụng

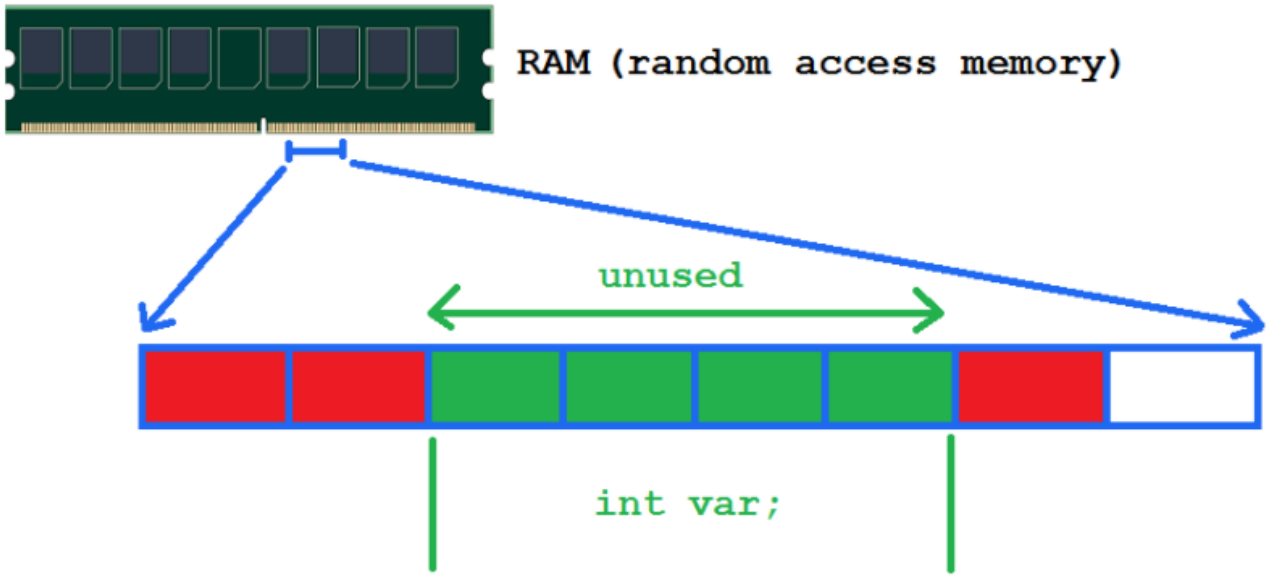
Alphabets	A .... Z, a .... z
Numeric	0 9
Special Characters	+ - * / ! @ # $ % ^ & [ ] ( ) { } = > < _ \ | ? . , ; : ‘ “ ” ;
White space	Blank space ( ), Horizontal tab (\t), Carriage return (\r), Newline (\n), Form feed (\f)
Other characters	256 ASCII characters

Bảng 2: Bộ ký tự trong C++

- **Biến (Variable) là tên được gán cho vùng nhớ trong máy tính để lưu trữ dữ liệu**
- Cho phép sử dụng tên thay vì địa chỉ vùng nhớ để xử lý dữ liệu
- Dữ liệu lưu trữ trong biến có thể thay đổi trong quá trình thực thi chương trình
- Biến cần được khai báo trước khi sử dụng
- Cú pháp khai báo biến:

kiểu_dữ_liệu tên_biến;

- **Kiểu dữ liệu xác định kích thước và loại giá trị của biến**
- Trình biên dịch cấp phát một phần bộ nhớ trên RAM cho biến dựa trên kiểu dữ liệu được khai báo
- Mỗi kiểu dữ liệu yêu cầu một lượng bộ nhớ nhất định và hỗ trợ phép toán tuỳ thuộc vào kiểu dữ liệu



Hình 5: Minh hoạ cấp phát bộ nhớ cho biến

- C++ hỗ trợ các kiểu dữ liệu sau:
  1. **Built-in data type - Kiểu dữ liệu tích hợp**
  2. **Derived data types - Kiểu dữ liệu dẫn xuất**
  3. **User-defined data types - Kiểu dữ liệu người dùng định nghĩa**

Kiểu dữ liệu tích hợp: kiểu dữ liệu đã được định nghĩa, có thể trực tiếp sử dụng trong chương trình

- Integer
- Character
- Boolean
- Floating Point
- Double Floating Point
- Valueless or Void
- Wide Character

Kiểu dữ liệu	Kích thước (bytes)	Khoảng giá trị
short int	2	-32,768 đến 32,767
unsigned short int	2	0 đến 65,535
unsigned int	4	0 đến 4,294,967,295
int	4	-2,147,483,648 đến 2,147,483,647
long int	4	-2,147,483,648 đến 2,147,483,647
unsigned long int	4	0 đến 4,294,967,295
long long int	8	-2^{63} đến $2^{63}-1$
unsigned long long int	8	0 đến 18,446,744,073,709,551,615
signed char	1	-128 đến 127
unsigned char	1	0 đến 255
float	4	$-3.4 * 10^{38}$ đến $3.4 * 10^{38}$
double	8	$-1.7 * 10^{308}$ đến $1.7 * 10^{308}$
long double	12	$-1.1 * 10^{4932}$ đến $1.1 * 10^{4932}$
wchar_t	4	một chuỗi ký tự dài

Bảng 3: Kiểu dữ liệu tích hợp trong C++

Kiểu dữ liệu dẫn xuất: kiểu dữ liệu được tạo ra từ kiểu dữ liệu tích hợp hoặc từ kiểu dữ liệu dẫn xuất khác

- Function
- Array
- Pointer
- Reference

Kiểu dữ liệu người dùng định nghĩa: kiểu dữ liệu được xác định bởi người dùng

- Class
- Structure
- Union
- Enumeration
- Typedef

Chuyển đổi một toán hạng hoặc biểu thức từ kiểu dữ liệu này sang kiểu dữ liệu khác gọi là ép kiểu (Type Conversion / Typecasting)

- **Ép kiểu ngầm định (Implicit Type Conversion):** được thực hiện tự động khi một giá trị được sao chép sang một kiểu dữ liệu tương thích

```
1 int a = 2024;
2 long long b;
3 b = a;
```

- **Ép kiểu tường minh (Explicit Type Conversion):** chuyển đổi kiểu dữ liệu một cách rõ ràng bởi người lập trình

```
1 int a = 2024;
2 long long b;
3 b = (long long)a;
```

∗ **Chú ý:** Ép kiểu từ kiểu dữ liệu lớn sang kiểu dữ liệu nhỏ có thể dẫn đến sai khi dữ liệu vượt quá giới hạn của kiểu dữ liệu nhỏ

Là đơn vị nhỏ nhất trong chương trình, C++ có năm loại token:

1. Định danh (Identifier)
2. Từ khoá (Keyword)
3. Hằng số (Constant)
4. Toán tử (Operator)
5. Dấu phân cách (Delimiter)

Là tên được gán cho phần tử trong chương trình như biến, hàm, mảng, đối tượng, lớp, ...

- Quy tắc định danh:
  - Định danh là một chuỗi ký tự, nên bắt đầu bằng chữ cái từ A-Z (chữ hoa) hoặc a-z (chữ thường) hoặc _ (dấu gạch dưới)
  - C++ phân biệt chữ hoa và chữ thường
  - Không cho phép chứa ký tự đặc biệt ngoại trừ dấu gạch dưới _
  - Định danh không được bao gồm khoảng trắng
  - Từ khóa không được được sử dụng làm định danh
  - Định danh nên có độ dài hợp lý và liên quan đến mục đích sử dụng
- Một số định danh hợp lệ: Temp, powerOf2, sum_of_array

- **Là những từ được định nghĩa trước**
- **Mang ý nghĩa đặc biệt với trình biên dịch**

asm	else	new	this
auto	enum	operator	throw
bool	explicit	private	true
break	export	protected	try
case	extern	public	typedef
catch	false	register	typeid
char	float	reinterpret_cast	typename
class	for	return	union
const	friend	short	unsigned
const_cast	goto	signed	using
continue	if	sizeof	virtual
default	inline	static	void
delete	int	static_cast	volatile
do	long	struct	wchar_t
double	mutable	switch	while
dynamic_cast	namespace	template	

Bảng 4: Danh sách từ khoá trong C++

Là những giá trị không thay đổi trong quá trình thực thi chương trình, C++ hỗ trợ bốn loại hằng số:

1. Hằng số nguyên (Integer Constant)
2. Hằng số thực (Floating Constant)
3. Hằng số ký tự (Character Constant)
4. Hằng số chuỗi (String Constant)

- **Là số nguyên, có thể là số dương hoặc số âm**
- Không có phần thập phân hoặc số mũ
- Cách sử dụng:

```
int a = 15; // Decimal Integer Constant
int b = 0113; // Octal Integer Constant
int c = 0x4b; // Hexadecimal Integer Constant
int d = 0b1001011; // Binary Integer Constant
unsigned int e = 15u; // Unsigned Integer Constant
```

- **Là số thực, có thể là số dương hoặc số âm**
- Chứa dấu thập phân (.) và có thể chứa số mũ
- Biểu diễn giá trị có phần thập phân và có thể được biểu diễn dưới hai dạng: dạng thập phân và dạng số mũ
- Cách sử dụng:

```
1 float a = 20.24;
2 float b = 202.4e-8;
3 float c = 202.4e8;
```

- **Được xác định bởi một ký tự nằm trong cặp dấu nháy đơn (' ')**
- Cách sử dụng:

```
1 char ch = 'D';
```

- Chuỗi thoát (escape sequence) - hằng số ký tự đặc biệt, cấu trúc là dấu gạch chéo ngược (\ ) và theo sau là một ký tự

Chuỗi thoát	Mô tả	Chuỗi thoát	Mô tả
\'	Dấu nháy đơn	\"	Dấu nháy đôi
\?	Dấu chấm hỏi	\\	Dấu gạch chéo ngược
\0	Ký tự Null	\a	Tiếng chuông
\b	Xoá ký tự	\f	Trang mới
\n	Dòng mới	\r	Trả về đầu dòng
\t	Tab ngang	\v	Tab dọc

Bảng 5: Chuỗi thoát trong C++

- **Là một chuỗi ký tự (có thể có từ 0 đến nhiều ký tự), được quy định đặt trong cặp dấu nháy đôi (" ")**
- Được coi là một mảng ký tự (char array)
- Trình biên dịch tự động đánh dấu kết thúc chuỗi bằng cách thêm ký tự đặc biệt - "Null Character" (\0) vào cuối chuỗi
- Cách sử dụng:

```
1 char str[25] = "Hello PTIT"; // "Hello PTIT\0" in the memory
```

Là ký hiệu được sử dụng để thực hiện phép toán giữa toán hạng (biến, hằng số, lời gọi hàm..) hoặc biểu thức (tạo thành từ toán hạng và toán tử)

1. Toán tử số học (Arithmetic operator)
2. Toán tử gán (Assignment operator)
3. Toán tử tăng và giảm (Increment and Decrement Operator)
4. Toán tử quan hệ (Relational operator)
5. Toán tử logic (Logical operator)
6. Toán tử bit (Bitwise operator)
7. Toán tử điều kiện (Conditional Operator)

- **Sử dụng để thực hiện phép tính toán học thông thường giữa toán hạng hoặc biểu thức**

Toán tử	Mô tả	Ví dụ (a = 15, b = 20)
+	Cộng hai toán hạng	a + b = 35
-	Trừ toán hạng thứ hai từ toán hạng thứ nhất	a - b = -5
*	Nhân hai toán hạng	a * b = 300
/	Chia tử số cho mẫu số	b / a = 1
%	Lấy phần dư sau khi chia phần nguyên	b % a = 5
+	Giữ nguyên dấu của số	+a = 15
-	Đảo ngược dấu của một số	-a = -15

Bảng 6: Toán tử số học trong C++

- **Sử dụng để gán giá trị cho biến, các giá trị gán có thể là biến, giá trị, biểu thức, lời gọi hàm, ...**

Toán tử	Ví dụ	Tương đương
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

Bảng 7: Toán tử gán trong C++

- **Sử dụng để tăng hoặc giảm giá trị của biến đi một đơn vị**
- Nếu đặt trước biến (tiền tố), giá trị trả về là giá trị sau thi thay đổi
- Nếu đặt sau biến (hậu tố), giá trị trả về là giá trị trước khi thay đổi

Toán tử	**Mô tả**	**Ví dụ (a = 15)**
++	Tăng giá trị lên 1 đơn vị	b = a++ hoặc c = ++a đều thay đổi giá trị của a thành 16 nhưng giá trị của b là 15 còn của c là 16
−−	Giảm giá trị đi 1 đơn vị	b = a−− hoặc b = −−a đều thay đổi giá trị của a thành 14 nhưng giá trị của b là 15 còn của c là 14

Bảng 8: Toán tử tăng và giảm trong C++

```cpp
#include <iostream>
#include <cstdio>

using namespace std;

int main() {
    int a = 10, b = 10, c = 10, d = 10;

    printf("%d %d %d\n", a++, a, ++a); // Output: 11 12 12
    cout << b++ << " " << b << " " << ++b << endl;  // Output: 11 12 12

    printf("%d %d %d\n", ++c, c, c++); // Output: 12 12 10
    cout << ++d << " " << d << " " << d++ << endl; // Output: 12 12 10

    a = 10, b = 10, c = 10, d = 10;

    printf("%d %d %d\n", a--, a, --a); // Output: 9 8 8
    cout << b-- << " " << b << " " << --b << endl; // Output: 9 8 8

    printf("%d %d %d\n", --c, c, c--); // Output: 8 8 10
    cout << --d << " " << d << " " << d-- << endl; // Output: 8 8 10

    return 0;
}
```

Mã nguồn 8: Cách hoạt động của toán tử tăng và giảm

- **Sử dụng để so sánh hai toán hạng hoặc biểu thức, trả về true (1) hoặc false (0)**

Toán tử	Mô tả	Ví dụ (a=10, b=5)
<	Kiểm tra nếu giá trị của toán hạng bên trái nhỏ hơn giá trị của toán hạng bên phải	a < b trả về false (0)
<=	Kiểm tra nếu giá trị của toán hạng bên trái nhỏ hơn hoặc bằng giá trị của toán hạng bên phải	a <= b trả về false (0)
>	Kiểm tra nếu giá trị của toán hạng bên trái lớn hơn giá trị của toán hạng bên phải	a > b trả về true (1)
>=	Kiểm tra nếu giá trị của toán hạng bên trái lớn hơn hoặc bằng giá trị của toán hạng bên phải	a >= b trả về false (0)
==	Kiểm tra nếu giá trị của hai toán hạng bằng nhau hay không	a == b trả về false (0)
!=	Kiểm tra nếu giá trị của hai toán hạng không bằng nhau hay không	a != b trả về true (1)

Bảng 9: Toán tử quan hệ trong C++

- **Sử dụng để kết hợp nhiều điều kiện quan hệ logic giữa toán hạng hoặc biểu thức, trả về true (1) hoặc false (0)**

Toán tử	Mô tả	Ví dụ (a = 4)
&&	Trả về true nếu cả hai điều kiện đều đúng Trả về false nếu một trong hai điều kiện sai	$a < 5$ && $a < 10 = \text{true}$
||	Trả về true nếu một trong hai điều kiện đúng Trả về false nếu cả hai điều kiện đều sai	$a < 5 \,\|\|\, a < 4 = \text{true}$
!	Đảo ngược điều kiện, trả về false nếu điều kiện là true và ngược lại	$!(a < 5 \,\&\&\, a < 10) = \text{false}$

Bảng 10: Toán tử logic trong C++

- **Sử dụng để thay đổi giá trị toán hạng hoặc biểu thức ở cấp độ bit (nhị phân)**

Toán tử	Mô tả
&	Toán tử AND theo bit
|	Toán tử OR theo bit
^	Toán tử XOR theo bit
~	Toán tử NOT theo bit
>>	Toán tử dịch bit phải
<<	Toán tử dịch bit trái

Bảng 11: Toán tử bit trong C++

- **Sử dụng để viết ngắn gọn câu lệnh điều kiện**
- Toán tử: **?:**
- Cú pháp:

  <Điều kiện> ? <Toán hạng/Biểu thức 1> : <Toán hạng/Biểu thức 2>
- Điều kiện là biểu thức của toán tử quan hệ hoặc biểu thức của toán tử logic
- Mô tả:
  - Nếu điều kiện true (1), kết quả trả về giá trị toán hạng/biểu thức 1
  - Nếu điều kiện false (0), kết quả trả về giá trị toán hạng/biểu thức 2

Toán tử cũng có thể được phân loại dựa trên số lượng toán hạng hoặc biểu thức chúng tác động

- Toán tử một ngôi (Unary Operator)
- Toán tử hai ngôi (Binary Operator)
- Toán tử ba ngôi (Ternary Operator)

Toán tử một ngôi	Toán tử hai ngôi	Toán tử ba ngôi
Hoạt động trên một toán hạng duy nhất	Hoạt động trên hai toán hạng	Hoạt động trên ba toán hạng
Toán tử tăng và giảm Toán tử bit (chỉ $\sim$)	Toán tử số học Toán tử gán Toán tử quan hệ Toán tử logic Toán tử bit (trừ $\sim$)	Toán tử điều kiện

Bảng 12: Toán tử theo ngôi trong C++

Sử dụng kết hợp với từ khoá, toán tử hoặc biểu thức để thực hiện chức năng đặc biệt

Toán tử	Mô tả
sizeof	Toán tử kích thước, trả về kích thước của kiểu dữ liệu
,	Toán tử dấu phẩy, sử dụng để liên kết các biểu thức liên quan
. và ->	Toán tử thành viên, sử dụng để tham chiếu thành viên riêng lẻ của class, struct và union
cast	Toán tử ép kiểu, chuyển đổi kiểu dữ liệu sang kiểu khác
&	Toán tử địa chỉ, dùng để lấy địa chỉ vùng nhớ của biến
*	Toán tử con trỏ, dùng để trỏ đến biến
>>	Toán tử nhập, dùng để nhập dữ liệu
<<	Toán tử xuất, dùng để xuất dữ liệu

Bảng 13: Toán tử đặc biệt trong C++

Toán tử theo thứ tự giảm dần độ ưu tiên	Tính kết hợp
( ) [ ] -> . ++(hậu tố) −−(hậu tố)	Trái sang phải
! ∼ ++(tiền tố) −−(tiền tố) Toán tử dấu (+ -) * & cast sizeof	Phải sang trái
new delete	Trái sang phải
Toán tử nhân chia (* / %)	
Toán tử cộng trừ (+ -)	
Toán tử dịch bit (<< >>)	
Toán tử quan hệ (< <= > >=)	
Toán tử quan hệ (== !=)	
Toán tử AND bit (&)	
Toán tử XOR bit (^)	
Toán tử OR bit (|)	
Toán tử AND logic (&&)	
Toán tử OR logic (||)	
Toán tử điều kiện (?:)	Phải sang trái
Toán tử gán (= += -= *= /= %= &= |= ^= >>= <<=)	
Toán tử dấu phẩy (,)	Trái sang phải

Bảng 14: Thứ tự ưu tiên của toán tử trong C++

Tính toán giá trị của biến **result** trong các trường hợp sau:

```
int x = 5, result;
result = ++x * 2 + x-- + ++x + x--;

int x = 5, y = 5, result;
result = ++x * 2 + --y + y-- + x++;

int x = 5, y = 10, z = 15, result;
result = (x + y > z) && (y - z < x) || (x * y == z * z);

int result = 10, x = 5;
result *= x + 2 * result - --x;

int a = 5, b = 15, c = 10, result;
result = (a > b) ? ((a > c) ? a : c) : ((b > c) ? b : c);

int result = 5;
result += result++ * 2 + --result;

int a = 10, b = 15, c = 5, result;
result = a > b && (c = b - a) || (c += a);

int a = 10, b = 5, result;
result = ++a > b-- ? a++ : --b;
```

Là các ký tự đặc biệt, đóng vai trò quan trọng xác định cấu trúc và thao tác thực thi mã nguồn

Ký hiệu	Tên	Mô tả
%	Dấu phần trăm	Dùng với các định dạng
;	Dấu chấm phẩy	Đánh dấu kết thúc câu lệnh
[ ]	Dấu ngoặc vuông	Chỉ định chỉ số của mảng
( )	Dấu ngoặc đơn	Gọi hàm, chỉ định các tham số hàm và sử dụng trong câu lệnh điều kiện, vòng lặp, ...
{ }	Dấu ngoặc nhọn	Bao quanh khối lệnh
#	Dấu thăng	Chỉ thị tiền xử lý
\	Dấu gạch chéo ngược	Biểu thị chuỗi thoát
::	Hai dấu hai chấm	Chỉ định phạm vi khi truy cập thành viên của class hoặc namespace
//	Dấu gạch chéo	Biểu thị chú thích
/* ... */	Dấu gạch chéo và dấu sao	Biểu thị chú thích nhiều dòng
' '	Dấu nháy đơn	Đánh dấu ký tự đơn
" "	Dấu nháy kép	Đánh dấu chuỗi ký tự

Bảng 15: Dấu phân cách trong C++

- **bits/stdc++.h** - thư viện không chính thức được giới thiệu bởi GCC, bao gồm rất nhiều thư viện tiêu chuẩn của C++
- Ưu điểm:
  - Tiết kiệm thời gian liệt kê từng thư viện cần dùng, tránh lỗi biên dịch
  - Đặc biệt hữu ích trong môn học hay lập trình thi đấu
- Nhược điểm:
  - Tăng thời gian biên dịch
  - Tăng kích thước mã đối tượng trung gian
  - Không thuộc tiêu chuẩn C++
  - Khó quản lý mã nguồn trong dự án lớn

- **Là sự kết hợp của các token**
- Thực hiện một chức năng cụ thể trong chương trình (gán giá trị, điều kiện, vòng lặp, gọi hàm, ...)
- Trình biên dịch bỏ qua khoảng trắng (dấu cách, tab và xuống dòng) chen giữa lệnh

```
1 int a=2024;
2 a = 2024;
3 a          =                2024;
4 a
5 =
6 2024;
```

- Phân loại:
  - Câu lệnh đơn (Simple statement): thực hiện một hành động cụ thể và chỉ chiếm một dòng mã
  - Khối lệnh (Code block): một nhóm câu lệnh đơn đặt trong dấu ngoặc nhọn { }

```
1 // Simple statement
2 int a = 2024;
3
4 //Code block
5 {
6     int b = 2;
7     a *= b;
8 }
```

- Thư viện: **iostream**
- Namespace: **std**
- Cú pháp:

```
std::cin >> variable1 >> variable2 >> variable3 >> ...;
std::cout << argument1 << argument2 << argument3 << ...;
```

- Chú ý:
  - **cin** nhận giá trị nhập từ bàn phím cho đến khi gặp khoảng trắng (dấu cách, tab và xuống dòng) hoặc giá trị không hợp lệ, sau đó gán giá trị đó cho biến nhập tương ứng
  - Nếu không có giá trị hợp lệ, biến nhập giữ nguyên giá trị trước đó
  - **cout** tính toán (nếu cần) sau đó hiển thị giá trị của biến, toán hạng hoặc biểu thức truyền vào lên màn hình
  - Chuỗi thoát có thể sử dụng trong **cout**
  - **endl** (thuộc namespace **std**) thường được sử dụng thay thế **\n** khi xuống dòng

- Thư viện: **iomanip**
- Namespace: **std**
- Tính năng thường dùng:
  - Định dạng số chữ số được hiển thị của số thực

```
double pi = 3.1415926535;
std::cout << std::setprecision(4) << pi;
// Tinh ca phan nguyen va phan thap phan, output: 3.142
std::cout << std::fixed << std::setprecision(4) << pi;
// Chi tinh phan thap phan, output: 3.1416
```

  - Thiết lập chiều rộng tối thiểu của giá trị khi in giá trị ra màn hình

```
std::cout << std::setw(5) << 123; // Output: "  123"
```

  - Thay thế ký tự khác vào khoảng trống khi sử dụng setw

```
std::cout << std::setw(5) << std::setfill('0') << 123;
// Output: "00123"
```

  - Hiển thị số thực theo dạng thập phân hoặc số mũ

```
double pi = 3.14159;
std::cout << std::fixed << pi << std::endl;
// Output: "3.141590"
std::cout << std::scientific << pi;
// Output: "3.141590e+00"
```

- Tối ưu thời gian chạy trong bài toán yêu cầu nhập xuất nhiều dữ liệu

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    ios_base::sync_with_stdio(false);
    // ios_base::sync_with_stdio(0);
    cin.tie(NULL); cout.tie(NULL);
    // cin.tie(0); cout.tie(0);
    // cin.tie(nullptr); cout.tie(nullptr);

    // Code here

    return 0;
}
```

- Lỗi làm tròn số của số thực

```cpp
double a = 0.3 * 3 + 0.1;
std::cout << std::fixed << std::setprecision(20) << a;
// Output: 0.99999999999999988898
```

- Rủi ro khi sử dụng toán tử so sánh với số thực
- Sử dụng **fabs** kết hợp với một hằng số rất bé epsilon (thường là 10^{-9}) để so sánh hai số thực

```cpp
double a = 0.3 * 3 + 0.1;
double b = 1;
if (fabs(a - b) < 1e-9) {
    std::cout << "a = b";
}
```

- Thư viện: **cmath**
- Namespace: **std**
- Hàm thông dụng:
  - Hàm lượng giác: **sin, cos, tan, asin, acos, atan, atan2**
  - Hàm mũ và logarit: **exp, log, log10**
  - Hàm lũy thừa và căn bậc hai: **pow, sqrt**
  - Hàm làm tròn số: **ceil, floor, round**
  - Hàm giá trị tuyệt đối: **abs, fabs**

- Cú pháp:

```
if (condition) {
    // Code block
}

if (condition) {
    // Code block
}
else {
    // Code block
}
```

- Điều kiện có thể là biểu thức của toán tử quan hệ hoặc biểu thức của toán tử logic
- Điều kiện cũng có thể chỉ bao gồm duy nhất một biến hoặc hằng số, khi đó, khối lệnh sẽ được thực thi nếu giá trị của biến hoặc hằng số đó khác 0 và ngược lại
- Nếu khối lệnh là một câu lệnh duy nhất (câu lệnh đơn), dấu ngoặc nhọn có thể được bỏ qua - **Không khuyến khích**

- Câu lệnh **if** là câu lệnh đơn và có thể lồng vào nhau, **else** (nếu có) sẽ tương ứng với **if** gần nhất chưa **else**

```
if (a != 0) {
    if (b > 0) {
        std::cout << "a != 0 va b > 0";
    }
    else {
        std::cout  << "a != 0 va b <= 0";
    }
}
else {
    if (b != 0) {
        std::cout << "a = 0 va b != 0";
    }
}
```

- Nên dùng **else** để loại trừ trường hợp

```
1  // Cach khong nen dung
2  if (delta < 0) {
3      std::cout << "Phuong trinh vo nghiem";
4  }
5
6  if (delta == 0) {
7      std::cout << "Phuong trinh co nghiem kep";
8  }
9
10 if (delta > 0) {
11     std::cout << "Phuong trinh co hai nghiem phan biet";
12 }
13
14 // Cach nen dung
15 if (delta < 0) {
16     std::cout << "Phuong trinh vo nghiem";
17 }
18 else {
19     if (delta == 0) {
20         std::cout << "Phuong trinh co nghiem kep";
21     }
22     else {
23         std::cout << "Phuong trinh co hai nghiem phan biet";
24     }
25 }
```

- Cú pháp:

```
switch (expression) {
    case value1: {
        // Code block
        break;
    }
    case value2: {
        // Code block
        break;
    }
    ...
    default: {
        // Code block
    }
}
```

- **expression** là biểu thức hoặc biến
- Các giá trị của **case** phải là hằng số hoặc biểu thức hằng số và không trùng lặp
- **default** là trường hợp mặc định, không bắt buộc

- **switch** nhảy đến case tương ứng, lần lượt thực hiện khối lệnh ở đó và các case bên dưới, cho đến khi gặp break hoặc kết thúc switch

```
switch (a) {
    case 1:
    case 3:
    case 5:
    case 7:
    case 8:
    case 10:
    case 12: {
        std::cout << "Thang co 31 ngay";
        break;
    }
    case 4:
    case 6:
    case 9:
    case 11: {
        std::cout << "Thang co 30 ngay";
        break;
    }
    case 2: {
        std::cout << "Thang co 28 hoac 29 ngay";
        break;
    }
}
```

- Câu lệnh **switch** là câu lệnh đơn và có thể lồng vào nhau

```
switch (a) {
    case 1: {
        std::cout << "Mot";
        break;
    }
    case 2: {
        switch (b) {
            case 1: {
                std::cout << "Mot";
                break;
            }
            case 2: {
                std::cout << "Hai";
                break;
            }
        }
        break;
    }
    default: {
        std::cout << "Khong biet";
    }
}
```

- Tại sao cần cấu trúc lặp?
  - Viết chương trình in ra màn hình các số từ 1 đến 10 → dùng 10 câu lệnh **cout**
  - Viết chương trình in ra màn hình các số từ 1 đến 10000 → dùng 10000 câu lệnh **cout**!!!
- Cấu trúc lặp giúp lặp lại một câu lệnh hoặc một khối lệnh nhiều lần khi còn thỏa mãn một điều kiện nào đó
  - **for**
  - **while**
  - **do ... while**

- **Cú pháp:**

```
//Normal loop
for (Initialization; Condition; Update) {
    // Code block
}

// Infinity loop
for {;;} {
    //Code block
}
```

- Phân tích:
  - **Initialization - Khởi đầu:** khởi tạo giá trị ban đầu cho biến lặp
  - **Condition - Điều kiện:** điều kiện lặp, khi điều kiện còn trả về giá trị đúng, khối lệnh sẽ được thực thi
  - **Update - Cập nhật:** cập nhật giá trị biến lặp sau mỗi lần lặp (còn gọi là bước nhảy)
- Có thể để trống các thành phần khi muốn sử dụng vòng lặp vô hạn

- Ví dụ:

```cpp
for (int i = 0; i < 10; ++i) {
    std::cout << i << " ";
}
// Output: 0 1 2 3 4 5 6 7 8 9

int j;
for (j = 0; j < 10; j = j + 1) {
    std::cout << j << " ";
}
// Output: 0 1 2 3 4 5 6 7 8 9

for (int k = 0; k < 10; k += 1) {
    std::cout << k << " ";
}
// Output: 0 1 2 3 4 5 6 7 8 9
```

- Câu lệnh **for** là câu lệnh đơn và có thể lồng vào nhau

```cpp
for (int i = 2; i < 10; ++i) {
    for (int j = 1; j <= 10; ++j) {
        std::cout << std::setw(2) << i * j << " ";
    }
    std::cout << std::endl;
}
/* Output:
2  4  6  8 10 12 14 16 18 20
3  6  9 12 15 18 21 24 27 30
4  8 12 16 20 24 28 32 36 40
5 10 15 20 25 30 35 40 45 50
6 12 18 24 30 36 42 48 54 60
7 14 21 28 35 42 49 56 63 70
8 16 24 32 40 48 56 64 72 80
9 18 27 36 45 54 63 72 81 90
*/
```

- Lệnh **break** kết thúc **for** gần nhất chứa nó
- Lệnh **continue** bỏ qua phần còn lại của vòng lặp hiện tại và chuyển sang vòng lặp tiếp theo

```cpp
for (int i = 0; i < 10; ++i) {
    if (i % 2 == 1) {
        break;
    }
    std::cout << i << " ";
}
// Output: 0

for (int i = 0; i < 10; ++i) {
    if (i % 2 == 1) {
        continue;
    }
    std::cout << i << " ";
}
// Output: 0 2 4 6 8
```

- **Cú pháp:**

```
while (Condition) {
    // Code block
}
```

- Phân tích:
  - **Condition - Điều kiện:** điều kiện lặp, khi điều kiện còn trả về giá trị đúng, khối lệnh sẽ được thực thi
  - Điều kiện có thể là biểu thức của toán tử quan hệ hoặc toán tử logic
- Có thể đặt giá trị **true** hoặc một số khác **0** làm điều kiện khi muốn sử dụng vòng lặp vô hạn

- Ví dụ:

```cpp
int i = 0;
while (i < 10) {
    std::cout << i << " ";
    ++i;
}
// Output: 0 1 2 3 4 5 6 7 8 9

int a = 2024;
while (a != 0) {
    std::cout << a % 10 << " ";
    a /= 10;
}
// Output: 4 2 0 2
```

- Câu lệnh **while** là câu lệnh đơn và có thể lồng vào nhau

```cpp
int i = 2;

while (i <= 9) {
    int j = 1;
    while (j <= 10) {
        std::cout << std::setw(2) << i * j << " ";
        j++;
    }
    std::cout << std::endl;
    i++;
}
/* Output:
2  4  6  8 10 12 14 16 18 20
3  6  9 12 15 18 21 24 27 30
4  8 12 16 20 24 28 32 36 40
5 10 15 20 25 30 35 40 45 50
6 12 18 24 30 36 42 48 54 60
7 14 21 28 35 42 49 56 63 70
8 16 24 32 40 48 56 64 72 80
9 18 27 36 45 54 63 72 81 90
*/
```

- Lệnh **break** kết thúc **while** gần nhất chứa nó
- Lệnh **continue** bỏ qua phần còn lại của vòng lặp hiện tại và chuyển sang vòng lặp tiếp theo

```
int a = 2345;
while (a != 0) {
    int b = a % 10;
    a /= 10;
    if (b % 2 == 0) {
        break;
    }
    std::cout << b << " ";
}
// Output: 5

int a = 2345;
while (a != 0) {
    int b = a % 10;
    a /= 10;
    if (b % 2 == 0) {
        continue;
    }
    std::cout << b << " ";
}
// Output: 5 3
```

- **Cú pháp:**

```
do {
    // Code block
} while (Condition);
```

- **do ... while** tương tự như **while**, nhưng khối lệnh sẽ được thực thi ít nhất một lần trước khi kiểm tra điều kiện
- **do ... while** yêu cầu dấu chấm phẩy (;) ở cuối câu lệnh
- Ví dụ:

```
// Nhap vao so n >= 0 va <= 100
int n;
do {
    std::cin >> n;
} while (n < 0 || n > 100);
```

- Template cho bài toán có nhiều bộ test

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    int t;
    cin >> t;

    while (t--) {
        cin >> input;

        // Code here

        cout << output << endl;
    }

    return 0;
}
```

- **Mã bài:** CPP05092
- **Tên bài:** TAM GIÁC ĐỀU RỖNG
- **Đề bài:** Viết chương trình nhập vào N là độ dài cạnh của tam giác đều. Thực hiện in ra tam giác đều rỗng tương ứng như ví dụ mẫu.
- **Input:** Một số nguyên N (5 <= N <= 100)
- **Output:** Ghi ra kết quả theo mẫu
- **Ví dụ:**
  - Input: 5
  - Output:

```
    *
   * *
  *   *
 *     *
* * * * *
```

- **Mã bài:** CPP05093
- **Tên bài:** HÌNH THOI RỖNG
- **Đề bài:** Viết chương trình nhập vào N là độ dài cạnh hình thoi. Thực hiện in ra hình thoi rỗng tương ứng như ví dụ mẫu.
- **Input:** Một số nguyên N (3 <= N <= 100)
- **Output:** Ghi ra kết quả theo mẫu
- **Ví dụ:**
  - Input: 3
  - Output:

```
  *
 * *
*   *
 * *
  *
```

- **Hàm (chương trình con) là một khối mã lệnh được định nghĩa để thực hiện một nhiệm vụ cụ thể**
- Lợi ích:
  - Giảm độ phức tạp của chương trình
  - Dễ quản lý, dễ bảo trì
  - Tăng khả năng tái sử dụng
- Trước khi sử dụng, hàm cần được định nghĩa bởi người dùng hoặc trình biên dịch (**#include** thư viện chứa hàm)
- Hàm thực thi qua lời gọi hàm, sau đó trả về giá trị hoặc không
- Hàm **main** là hàm đặc biệt, luôn được gọi đầu tiên trong quá trình thực thi chương trình

- Cú pháp:

```
data_type name_of_function (parameter1, parameter2, ...) {
    // Code block

    return value;
}
```

- **data_type** là kiểu dữ liệu của C++
- **name_of_function** là tên của hàm - một định danh hợp lệ
- **parameter1, parameter2, ...** là tham số - các biến được sử dụng trong nội bộ hàm, không bắt buộc nếu hàm không cần tham số
- **return value** - câu lệnh trả về giá trị của hàm, không bắt buộc nếu **data_type** là **void**

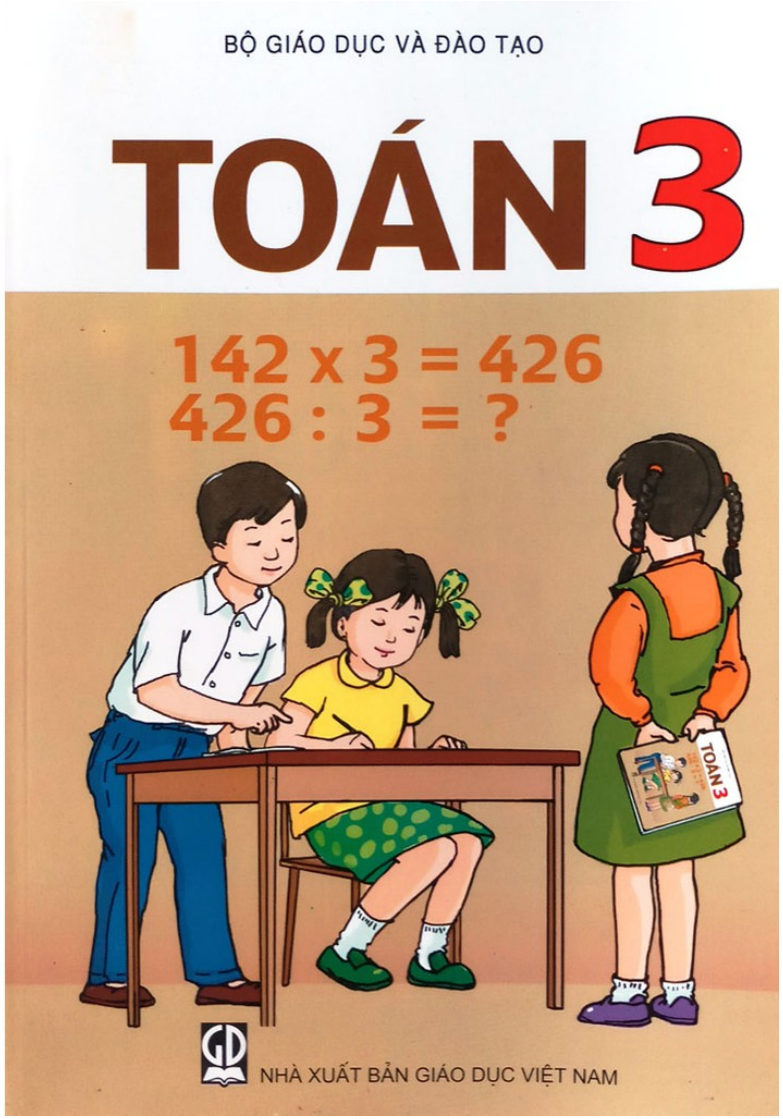
- Ví dụ hàm trả về:

```
int sum(int a, int b) {
    return a + b;
}
```

- Ví dụ hàm không trả về:

```
void print_hello_ptit() {
    std::cout << "Hello PTIT!" << std::endl;
}
```

- Có nhiều cách định nghĩa hàm:
  - Khai báo và định nghĩa hàm cùng lúc - thường dùng trong môn học hay lập trình thi đấu
  - Khai báo hàm sau đó định nghĩa hàm ở cuối mã nguồn - ít sử dụng
  - Khai báo, định nghĩa hàm ở tệp mã nguồn riêng và **#include** khi sử dụng (lập trình mô-đun) - thường áp dụng trong dự án thực tế
- Hàm có thể được gọi trong hàm khác hoặc trong chính nó - đệ quy



Hình 6: Minh họa đệ quy

- Thành phần của hàm đệ quy:
  - **Phần cơ sở:** điều kiện để hàm đệ quy kết thúc, nếu như không có phần này, hàm đệ quy sẽ thực hiện vô hạn gây ra tràn bộ nhớ Stack
  - **Phần đệ quy:** thân hàm chứa phần gọi đệ quy, thực hiện khi còn thỏa mãn điều kiện ở phần cơ sở
- Đệ quy giúp chương trình dễ đọc, dễ hiểu hơn tuy nhiên gây tốn bộ nhớ và giảm tốc độ thực thi
- Ví dụ:

```
1  int factorial(int n) {
2      if (n == 0) {
3          return 1;
4      }
5      return n * factorial(n - 1);
6  }
7
8  int sum_of_digits(int n) {
9      if (n == 0) {
10         return 0;
11     }
12     return n % 10 + sum_of_digits(n / 10);
13 }
```

- **Tên bài:** ĐỘ BỀN CỦA SỐ NGUYÊN
- **Đề bài:** Độ bền của số nguyên không âm N được định nghĩa như sau:
  - Nếu N có 1 chữ số thì độ bền N = 0;
  - Nếu N có từ 2 chữ số thì độ bền N bằng độ bền tích các chữ số của N cộng thêm 1.

Ví dụ: do_ben(99) = do_ben(81) + 1 = do_ben(8) + 1 + 1 = 0 + 1 + 1 = 2 .

Cho số nguyên N. Tính độ bền của N.

- **Input:**
  - Dòng đầu tiên số lượng bộ test T (T <= 100).
  - T dòng tiếp theo mỗi dòng chứa 1 số nguyên N (1 <= N <= 10^{18}).
- **Output:** Ghi kết quả trên T dòng, mỗi dòng in ra độ bền của N.
- **Ví dụ:**

Input	Output
2 99 100	2 1

- **Tên bài:** SỐ PHONG PHÚ
- **Đề bài:** Trong toán học, số phong phú là các số mà tổng các ước số của số đó (không kể chính nó) lớn hơn số đó.

  Ví dụ, số 12 có tổng các ước số (không kể 12) là $1 + 2 + 3 + 4 + 6 = 16 > 12$. Do đó 12 là một số phong phú.

  Bạn hãy lập trình đếm số lượng số phong phú trong đoạn [L, R].
- **Input:** Gồm 2 số L, R ($1 <= L <= R <= 10^6$).
- **Output:** Số lượng số phong phú trong đoạn [L, R].
- **Ví dụ:**

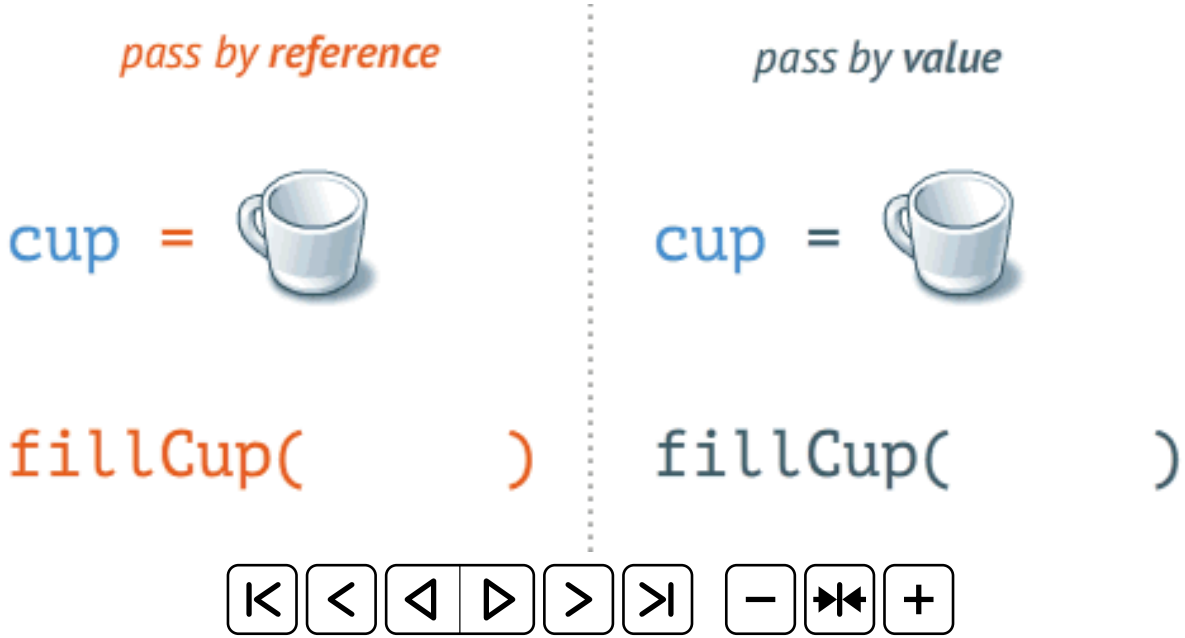
Input	Output
1 50	9

Từ 1 đến 50 có 9 số phong phú là: 12, 18, 20, 24, 30, 36, 40, 42, 48

- **Biến toàn cục (global variable):** biến được khai báo ngoài tất cả các hàm, có thể sử dụng trong toàn bộ chương trình
- **Biến cục bộ (local variable):** biến được khai báo trong một hàm hoặc khối lệnh {}, chỉ có thể sử dụng trong hàm hoặc khối lệnh đó, biến cục bộ bị xoá khi hàm hoặc khối lệnh kết thúc

```cpp
#include <bits/stdc++.h>
using namespace std;

int global_var;
void function() {
    int local_var1;
    // global_var, local_var1
}

int main() {
    int local_var2;
    {
        int local_var3;
        // global_var, local_var2, local_var3
    }
    // global_var, local_var2
    return 0;
}
```

- **Truyền tham trị (Pass by value):** truyền giá trị của biến, giá trị của biến gốc không bị thay đổi
- **Truyền tham chiếu (Pass by reference):** truyền địa chỉ của biến, giá trị của biến gốc bị thay đổi
- **Truyền con trỏ (Pass by pointer):** truyền con trỏ của biến, giá trị của biến gốc bị thay đổi



```cpp
#include <bits/stdc++.h>
using namespace std;

// Pass by value
void swap1(int a, int b) {
    int temp = a;
    a = b;
    b = temp;
}

// Pass by reference
void swap2(int& a, int& b) {
    int temp = a;
    a = b;
    b = temp;
}

// Pass by pointer
void swap3(int* a, int* b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

int main() {
    int x, y;

```

```
28      x = 5, y = 10;
29      swap1(x, y);
30      cout << x << " " << y << endl; // 5 10
31
32      x = 5, y = 10;
33      swap2(x, y);
34      cout << x << " " << y << endl; // 10 5
35
36      x = 5, y = 10;
37      swap3(&x, &y);
38      cout << x << " " << y << endl; // 10 5
39
40      return 0;
41  }
```

Mã nguồn 9: Mã nguồn ví dụ truyền tham số

- **Tham chiếu (reference) là biến được tạo ra để làm bí danh (tên giả, tên gọi khác) của biến gốc**
  - Địa chỉ của biến tham chiếu chính là địa chỉ của biến mà nó tham chiếu đến
  - Tham chiếu chia sẻ cùng một vùng nhớ với biến gốc, bất kỳ thay đổi nào đối với giá trị của tham chiếu thực chất là thay đổi giá trị của biến gốc
- Cú pháp:

```
1 data_type& reference_name = variable_name; // Nen dung
2 data_type & reference_name = variable_name;
3 data_type &reference_name = variable_name;
```

- Trong trường hợp này, toán tử **&** có nghĩa là "tham chiếu đến"

- Ví dụ:

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    int x = 5, y = 10;
    int z = x;
    int& ref = x;

    x = 15;

    cout << x << " " << &x << std::endl;
    cout << z << " " << &z << std::endl;
    cout << ref << " " << &ref << std::endl;

    ref = y;

    cout << x << " " << &x << std::endl;
    cout << ref << " " << &ref << std::endl;
}
```

- Chú ý: tham chiếu phải được khởi tạo ngay khi khai báo và không thể thay đổi tham chiếu đến biến khác

- **Con trỏ (pointer) là biến dùng để lưu trữ địa chỉ của biến khác**
  - Con trỏ có địa chỉ riêng
  - Giá trị lưu trữ trong vùng nhớ của con trỏ là địa chỉ của biến mà nó trỏ đến
  - Kích thước vùng nhớ của con trỏ phụ thuộc vào kiến trúc máy tính, 4 bytes trên hệ thống x86 và 8 bytes trên hệ thống x64

Hình 7: Minh hoạ con trỏ

- Cú pháp:

```
data_type* pointer_name = &variable_name; // Nen dung
data_type * pointer_name = &variable_name;
data_type *pointer_name = &variable_name;
```

- Khi khai báo con trỏ mà chưa khởi tạo địa chỉ trỏ đến, nên dùng **0**, **NULL** hoặc **nullptr** để tránh lỗi

```
int* p = 0;
int* q = NULL;
int* r = nullptr; // Nen dung
```

- Thao tác với con trỏ:
  - **Gán giá trị cho con trỏ:** con trỏ có thể được gán giá trị là địa chỉ của một biến hoặc con trỏ cùng kiểu (con trỏ thứ hai sẽ trỏ đến cùng địa chỉ mà con trỏ thứ nhất đang trỏ tới), khác với tham chiếu, con trỏ có thể trỏ đến địa chỉ của biến khác sau khi khởi tạo

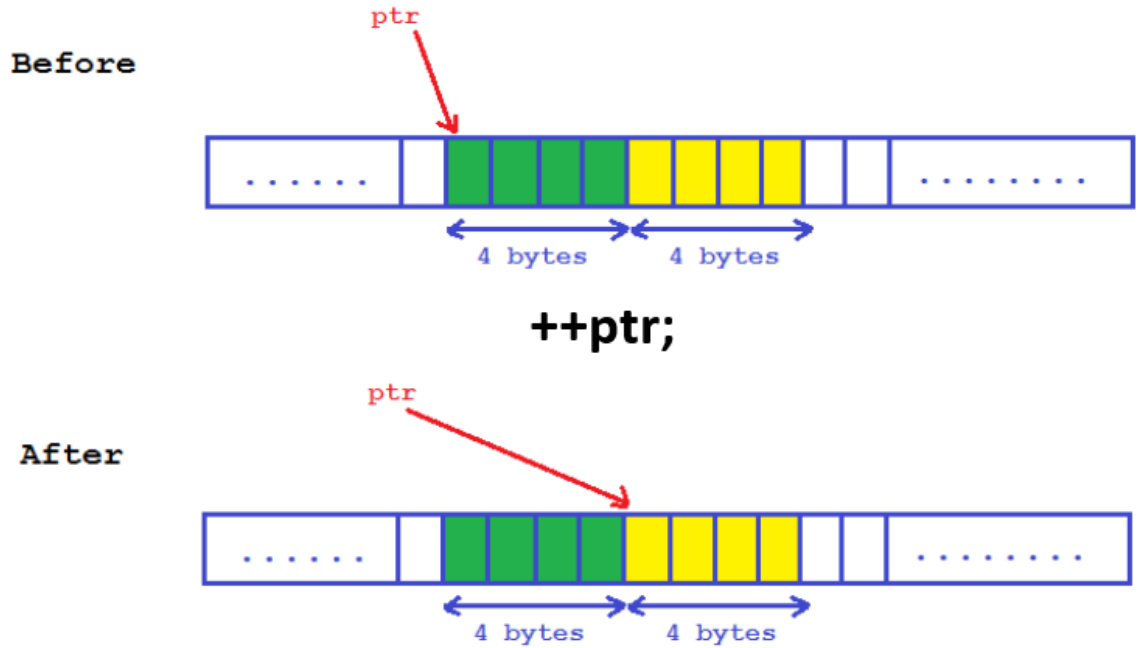
```
int x = 5, y = 10;
int* ptr = &x;
ptr = &y;
int* ptr_1 = ptr;
```

  - **Truy xuất giá trị ở địa chỉ mà con trỏ trỏ đến:** sử dụng toán tử giải tham chiếu *, cũng có thể thay đổi giá trị được lưu trữ ở địa chỉ đó (tương đương thay đổi giá trị của biến)

```
int x = 5, y = 10;
int* ptr = &x;
std::cout << *ptr << std::endl; // Output: 5
ptr = &y;
*ptr = 15;
std::cout << y << std::endl; // Output: 15
std::cout << *ptr << std::endl; // Output: 15
```

- **Tăng và giảm con trỏ:** con trỏ cũng có thể sử dụng các toán tử ++, −−, +, -, +=, -=

```
1 int x = 5;
2 int* ptr = &x;
3 ++ptr;
```



Hình 8: Minh hoạ toán tử ++ con trỏ

C++ hỗ trợ ba loại cấp phát bộ nhớ:

- **Cấp phát bộ nhớ tĩnh (Static memory allocation):**
  - Áp dụng cho biến **static** và biến toàn cục
  - Vùng nhớ được cấp phát khi chương trình bắt đầu chạy và giải phóng khi chương trình kết thúc
  - Kích thước vùng nhớ cố định, phải biết trước khi biên dịch
- **Cấp phát bộ nhớ tự động (Automatic memory allocation):**
  - Áp dụng cho tham số hàm và biến cục bộ
  - Vùng nhớ được cấp phát khi chương trình đi vào khối lệnh và giải phóng khi khối lệnh kết thúc
  - Kích thước vùng nhớ cố định, phải biết trước khi biên dịch

Thông thường, cấp phát tĩnh và tự động là đủ cho chương trình. Tuy nhiên, hai cách này không thể mở rộng vùng nhớ khi cần lưu trữ nhiều hơn và thu hồi vùng nhớ cấp phát thừa hay không còn sử dụng.

- **Cấp phát bộ nhớ động (Dynamic memory allocation):**
  - Cấp phát và giải phóng vùng nhớ linh hoạt, tối ưu sử dụng bộ nhớ
  - Vùng nhớ cấp phát được lưu trữ trong Heap, không tự động giải phóng khi kết thúc khối lệnh
  - Con trỏ đến vùng nhớ vẫn được lưu trữ trong Data Segment khi là biến toàn cục và lưu trữ trong Stack khi là biến cục bộ
  - Sử dụng toán tử **new** để cấp phát vùng nhớ và toán tử **delete** để giải phóng vùng nhớ đã cấp phát
  - Luôn giải phóng vùng nhớ được cấp phát khi không còn sử dụng, tránh rò rỉ bộ nhớ
  - Khi giải phóng vùng nhớ cấp phát sẽ tạo ra con trỏ lơ lửng (dangling pointer), nên gán giá trị **nullptr** sau khi giải phóng

Ví dụ cấp phát bộ nhớ động

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    int* p = new int;
    int* q = p;
    int *r = new int(5);

    *p = 5;
    cout << *p << " " << *r << endl; // Output: 5 5

    delete p;
    delete q;

    cout << *p << " " << *q << endl; // Error
    p = nullptr;

    return 0;
}
```

- Đặt vấn đề:
  - Chương trình cần lưu trữ 3 số nguyên → khai báo 3 biến **int a, b, c**
  - Chương trình cần lưu trữ 100 số nguyên → khai báo 100 biến **int a1, a2, a3, ..., a100** !!!
  - Chương trình cần lưu trữ **n** số nguyên với **n** nhập từ bàn phím → không thể biết trước số lượng biến cần khai báo
- Kiểu dữ liệu mảng giúp lưu trữ nhiều giá trị cùng kiểu dữ liệu trong một biến và truy cập dễ dàng
- Mảng có kích thước cố định, không thể thay đổi sau khi khai báo
- Trình biên dịch luôn chỉ định vùng bộ nhớ liên tục cho mảng

- Cú pháp:

```
data_type array_name[array_size];
data_type array_name[array_size1][array_size2]...[array_sizeN];
```

- Chú ý:
  - Phải xác định kiểu dữ liệu và kích thước (số lượng phần tử) của mảng trước khi sử dụng
  - Mảng nhiều chiều có số lượng phần tử là tích các chiều
  - **Bộ nhớ sử dụng = tổng số phần tử * kiểu dữ liệu**
  - Khai bảo mảng là biến toàn cục có kích thước giới hạn lớn hơn so với khai báo mảng là biến cục bộ

- Mảng cho phép khởi tạo giá trị ban đầu, những giá trị chưa được khởi tạo thường sẽ có giá trị mặc định hoặc giá trị rác

```
int a[4] = {1, 2, 3, 4}; // a = {1, 2, 3, 4}
int b[4] = {1}; // b = {1, ?, ?, ?}
int c[4] = {0} // c = {0, 0, 0, 0}
```

- Để tránh giá trị rác, nên chủ động khởi tạo giá trị ban đầu cho mảng

```
int a[4] = {0}; // a = {0, 0, 0, 0}
int b[4]; // b = {?, ?, ?, ?}
// Use memset function in <cstring>
memset(b, 0, sizeof(b)); // b = {0, 0, 0, 0}
int c[4]; // c = {?, ?, ?, ?}
//Use fill function of namespace std in <algorithm>
std::fill(c, c + 4, 0); // c = {0, 0, 0, 0}
```

- Mảng có thể khai báo không cần số lượng phần tử, nhưng cần khởi tạo giá trị ban đầu, khi đó số lượng phần tử là số giá trị khởi tạo

```
int a[] = {1, 2, 3, 4}; // a = {1, 2, 3, 4}
std::cout << sizeof(a) / sizeof(a[0]); // Output: 4
```

- Truy xuất đến phần tử của mảng:
  - Sử dụng toán tử **[ ]** và chỉ số của phần tử
  - Chỉ số có thể là biến số nguyên, hằng số nguyên hoặc biểu thức trả về giá trị nguyên
  - Chỉ số bắt đầu từ 0 đến số phần tử - 1
  - Chỉ số âm hay vượt quá số phần tử - 1 trả về giá trị rác
  - Ví dụ với mảng kích thước 4 phần tử: **a[0], a[1], a[2], a[3]** là các phần tử hợp lệ, **a[-1], a[4]** là các phần tử không hợp lệ
- Tên của mảng là con trỏ trỏ đến phần tử đầu tiên của mảng

```
1 int a[4] = {1, 2, 3, 4};
2 std::cout << *a << " " << a[0] << std::endl; // Output: 1 1
3 std::cout << *(a + 3) << " " << a[3] << std::endl; // Output: 4 4
4 std::cout << a << " " << &a[0] << std::endl; // same address
5 std::cout << a + 3 << " " << &a[3] << std::endl; // same address
```

- Mảng có thể được khai báo kích thước là số phần tử + 1 để sử dụng chỉ số từ 1 đến số phần tử
- Gán dữ liệu kiểu mảng: không thể gán trực tiếp mảng cho mảng khác, chỉ có thể gán từng phần tử của mảng

```
int a[4] = {1, 2, 3, 4};
int b[4];
b = a; // Error
for (int i = 0; i < 4; i++) {
    b[i] = a[i]; // Correct
}
```

- Truyền mảng vào hàm:

```
void nhap_mang(int arr[], int& n);
void xuat_mang(int* arr, int n);
```

- Mảng có thể cấp phát động để linh hoạt thay đổi kích thước mảng thông qua thay đổi vùng nhớ lưu trữ
- Dữ liệu trong vùng nhớ cũ cần được sao chép sang vùng nhớ mới và giải phóng sau khi sao chép xong, tránh rò rỉ bộ nhớ
- Ví dụ:

```
1 int* arr = new int[5];
2 int* newArr = new int[10];
3 std::copy(arr, arr + 5, newArr);
4 delete[] arr;
5 arr = newArr;
```

- **Tên bài:** SỨC MẠNH CỦA MẢNG
- **Đề bài:** Sức mạnh của một mảng được định nghĩa là phần tử có số lần xuất hiện nhiều nhất trong mảng, nếu có nhiều hơn một phần tử có cùng số lần xuất hiện nhiều nhất thì phần tử nào lớn nhất là sức mạnh của mảng. Yêu cầu là tìm sức mạnh của một mảng cho trước.
- **Input:**
  - Dòng đầu tiên chứa số nguyên dương n – số phần tử của mảng ($1 <= n <= 10^5$).
  - Dòng thứ hai gồm n phần tử của mảng, giá trị các phần tử này là số nguyên dương và không vượt quá 10^6.
- **Output:** In ra số nguyên duy nhất – sức mạnh của mảng cần tìm.

- **Ví dụ:**

Input	Output
3 1 2 1	1
5 1 3 2 2 3	3

- Trong test đầu tiên, 1 xuất hiện 2 lần, 2 xuất hiện 1 lần. Vậy sức mạnh của mảng là 1.
- Trong test thứ hai, 1 xuất hiện 1 lần, 2 xuất hiện 2 lần, 3 xuất hiện 2 lần. Tuy cùng xuất hiện 2 lần nhưng $3 > 2$ nên đáp án là 3.

- **Mảng đánh dấu (marking array):** là một mảng có kích thước bằng với **giá trị lớn nhất** của phần tử trong mảng khác cần xử lý
- Mỗi phần tử của mảng đánh dấu được sử dụng để lưu trữ thông tin nào đó, thường là số lần xuất hiện của một giá trị

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

#define MAXN 10000001
int n, a[MAXN], count[MAXN];

int main() {
    std::fill(count, count + MAXN, 0);
    cin >> n;
    for (int i = 1; i <= n; i++) {
        cin >> a[i];
        count[a[i]]++;
    }
    // count[value] la so lan xuat hien gia tri value cua a

    return 0;
}
```

Mã nguồn 10: Mã nguồn ví dụ mảng đánh dấu

- **Tên bài:** TRUY VẤN TỔNG
- **Đề bài:** Cho một mảng A gồm N số nguyên và Q truy vấn yêu cầu tính S là tổng các phần tử từ vị trí l đến vị trí r: $S = A_l + A_{l+1} + A_{l+1} + ... + A_{r-1} + A_r$.
- **Input:**
  - Dòng đầu tiên gồm hai số N và Q (1 <= N, Q <= 10^6), số phần tử của mảng và số truy vấn cần thực hiện.
  - Dòng tiếp theo chứa N số nguyên dương <= 10^6.
  - Q dòng tiếp theo chứa các truy vấn, mỗi dòng chứa hai số nguyên tương ứng l và r (1 <= l <= r <= N).
- **Output:** In ra trên Q dòng, mỗi dòng một số nguyên S là kết quả của truy vấn tương ứng.

- **Ví dụ:**

Input	Output
5 3 1 2 3 4 5 1 2 2 4 1 5	3 9 15

- Truy vấn 1, tổng các phần tử từ vị trí 1 đến 2 là $1 + 2 = 3$.
- Truy vấn 2, tổng các phần tử từ vị trí 2 đến 4 là $2 + 3 + 4 = 9$.
- Truy vấn 3, tổng các phần tử từ vị trí 1 đến 5 là $1 + 2 + 3 + 4 + 5 = 15$.

- **Mảng cộng dồn (prefix sum array):** là một mảng có kích thước bằng với **số lượng phần tử** của mảng cần xử lý
- Mỗi phần tử của mảng cộng dồn được sử dụng để lưu trữ tổng các phần tử từ vị trí đầu tiên đến vị trí đó

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

#define MAXN 10000001
int n, a[MAXN], sum[MAXN];

int main() {
    std::fill(sum, sum + MAXN, 0);
    cin >> n;
    for (int i = 1; i <= n; i++) {
        cin >> a[i];
        sum[i] = sum[i - 1] + a[i];
    }
    // sum[index] la tong cac phan tu cua a tu 1 den index

    return 0;
}
```

Mã nguồn 11: Mã nguồn ví dụ mảng cộng dồn

- **Tên bài:** SỐ NGUYÊN TỐ
- **Đề bài:** Cho số nguyên N, in ra tất cả các số nguyên tố nhỏ hơn hoặc bằng N.
- **Input:** Số nguyên N duy nhất ($2 <= N <= 10^7$).
- **Output:** Dòng duy nhất chứa các số nguyên tố nhỏ hơn hoặc bằng N, ngăn cách nhau bởi dấu space.
- **Ví dụ:**

Input	Output
10	2 3 5 7

- **Sàng nguyên tố (Sieve of Eratosthenes):** là thuật toán tìm tất cả các số nguyên tố nhỏ hơn hoặc bằng một số nguyên dương N
- Thuật toán:
  - Bắt đầu từ số 2, đánh dấu tất cả các bội số của 2 không là số nguyên tố
  - Tìm số nguyên tố tiếp theo chưa được đánh dấu, đánh dấu tất cả các bội số của số đó
  - Lặp lại cho đến khi tất cả các số nguyên tố nhỏ hơn hoặc bằng N được tìm thấy
- Độ phức tạp: $O(N \log \log N)$

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

#define MAXN 10000001
int prime[MAXN];

int main() {
    std::fill(prime, prime + MAXN, 1);
    prime[0] = 0;
    prime[1] = 0;

    for (int i = 2; i * i <= MAXN; i++) {
        if (prime[i] == 1) {
            for (int j = i * i; j <= MAXN; j += i) {
                prime[j] = 0;
            }
        }
    }
    // prime[i] = 1 neu i la so nguyen to, nguoc lai prime[i] = 0

    return 0;
}
```

Mã nguồn 12: Mã nguồn ví dụ sàng nguyên tố

- **Mảng hai chiều:** là mảng mà mỗi phần tử là một mảng một chiều
- Địa chỉ của các phần tử cũng là các vùng nhớ liên tiếp (thực chất là mảng một chiều có kích thước là tích số hàng và cột)
- Cú pháp:

```
data_type array_name[row_size][column_size];
```

- Các phần tử của mảng hai chiều được truy cập bằng hai chỉ số, chỉ số hàng và chỉ số cột

```
int a[2][3] = {{1, 2, 3}, {4, 5, 6}};
for (int i = 0; i < 2; i++) {
    for (int j = 0; j < 3; j++) {
        std::cout << a[i][j] << " ";
    }
    std::cout << std::endl;
}
```

- Truyền mảng hai chiều vào hàm:

```
void nhap_mang(int arr[][1000], int& n, int& m);
void xuat_mang(int arr[][1000], int n, int m);
```

- **Tên bài:** TÍCH CHẬP
- **Đề bài:** Cho hai ma trận A và N có kích thước lần lượt là N x M và K x K. Tích chập của hai ma trận A và B được định nghĩa là ma trận C có kích thước (N - K + 1) x (M - K + 1) với các phần tử được tính theo công thức:

$$C[i][j] = \sum_{u=1}^{K} \sum_{v=1}^{K} A[i+u-1][j+v-1] \cdot B[u][v]$$

Với:

  - i chạy từ 1 đến N - K + 1
  - j chạy từ 1 đến M - K + 1
  - u và v là các chỉ số của ma trận B, chạy từ 1 đến K

Yêu cầu của bài toán là tính ma trận C và in ra màn hình.

- **Input:**
  - Dòng đầu tiên chứa ba số nguyên N, M và K (5 <= M, N <= 100, 2 <= K <= 10, K <= min(N, M)) - kích thước của ma trận A và B.
  - N dòng tiếp theo, mỗi dòng chứa M số nguyên không âm <= 100, là các phần tử của ma trận A.
  - K dòng tiếp theo, mỗi dòng chứa K số nguyên không âm <= 100, là các phần tử của ma trận B.
- **Output:** Ma trận kết quả C kích thước (N - K + 1) x (M - K + 1).

- **Ví dụ:**

Input	Output
4 4 2 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 1 0 0 1	7 9 11 15 17 19 23 25 27

- **Chuỗi kí tự - string** là kiểu dữ liệu chuỗi kí tự được nâng cấp so với C, thực chất là một mảng char động
- Thư viện: **string**
- Namespace: **std**
- Cú pháp:

```
std::string str;
std::cin >> str; // Read string without space
std::getline(std::cin, str); // Read string with space
std::getline(std::cin, str, chr); // Read string until character chr
std::cout << str; // Print string
for (int i = 0; i < str.length(); i++) { // or str.size()
    std::cout << str[i] << " "; // Access character i
}
```

- Chú ý:
  - Không cần khai báo kích thước trước khi sử dụng
  - Có thể thực hiện các phép toán gán, cộng chuỗi, so sánh chuỗi, ...
  - Không kết thúc bằng ký tự NULL
  - Lỗi trôi lệnh khi nhập:

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

int main() {
    int a;
    string str;
    cin >> a;
    cin.ignore();
    getline(cin, str);
    cout << a << " " << str;

    return 0;
}
```

- **Các phương thức cơ bản:**
  - **length() hoặc size():** trả về độ dài của chuỗi
  - **clear():** xoá toàn bộ chuỗi, độ dài chuỗi trở về 0
  - **empty():** kiểm tra chuỗi, trả về true nếu rỗng, ngược lại trả về false
  - **swap(str):** hoán đổi giá trị giữa hai chuỗi
  - **append(str):** nối chuỗi str vào cuối chuỗi
  - **insert(index, str):** chèn chuỗi str vào vị trí index
  - **erase(index, num):** xóa num ký tự từ vị trí index
  - **replace(index, num, str):** thay thế num ký tự từ vị trí index bằng chuỗi str
  - **find(str):** tìm chuỗi str trong chuỗi, trả về vị trí đầu tiên của chuỗi str trong chuỗi, nếu không tìm thấy trả về -1
  - **rfind(str):** tìm chuỗi str trong chuỗi, trả về vị trí cuối cùng của chuỗi str trong chuỗi, nếu không tìm thấy trả về -1

- **substr(index, num):** trả về chuỗi con bắt đầu từ vị trí index và có độ dài num
- **compare(str):** so sánh chuỗi với chuỗi str, trả về 0 nếu bằng nhau, -1 nếu chuỗi nhỏ hơn, 1 nếu chuỗi lớn hơn
- **c_str():** chuyển chuỗi sang chuỗi ký tự kiểu C, cần các hàm trong thư viện **cstring** nếu muốn xử lý chuỗi kiểu C
- **push_back(chr):** thêm ký tự chr vào cuối chuỗi
- **pop_back():** xóa ký tự cuối cùng của chuỗi
- **resize(num, chr):** thay đổi độ dài chuỗi thành num, nếu num lớn hơn đồ dài hiện tại thì ký tự chr (NULL nếu không truyền chr) được thêm vào cuối chuỗi, ngược lại xóa các ký tự từ vị trí num
- **at(index):** trả về ký tự tại vị trí index
- **front():** trả về ký tự đầu tiên của chuỗi
- **back():** trả về ký tự cuối cùng của chuỗi

- **Tên bài:** CHUẨN HOÁ CHUỖI
- **Đề bài:** Cho một chuỗi chứa các từ, trong đó các từ có thể cách nhau bởi nhiều dấu cách. Viết chương trình chuẩn hóa chuỗi này theo quy tắc như sau:
  - Xóa tất cả các dấu cách thừa (chỉ giữ lại một dấu cách giữa các từ).
  - Tất cả ký tự đầu tiên của mỗi từ phải viết hoa, các ký tự còn lại của từ phải viết thường.
  - Không có dấu cách nào ở đầu hoặc cuối chuỗi.

- **Input:**
  - Dòng đầu tiên chứa số nguyên t (1 <= t <= 100), số lượng bộ test.
  - t dòng tiếp theo, mỗi dòng chứa một chuỗi ký tự s (1 <= |s| <= 10^5) - chuỗi cần chuẩn hoá, được tạo thành từ các ký tự chữ cái (A-Z, a-z) và dấu cách.
- **Output:** Với mỗi chuỗi đầu vào, in ra trên một dòng chuỗi đã được chuẩn hoá.
- **Ví dụ:**

Input	Output
2 cOdIng Is Fun hELlo pTiT	Coding Is Fun Hello Ptit

- **Tên bài:** XOÁ KÝ TỰ TRÙNG LẶP
- **Đề bài:** Cho một chuỗi ký tự bao gồm các chữ cái thường và số, hãy viết chương trình xóa tất cả các ký tự xuất hiện nhiều lần trong chuỗi với quy tắc: ký tự đầu tiên xuất hiện của mỗi ký tự được giữ lại, các ký tự trùng lặp tiếp theo sẽ bị loại bỏ. Chuỗi kết quả giữ nguyên thứ tự của các ký tự xuất hiện ban đầu.
- **Input:**
  - Dòng đầu tiên chứa số nguyên t ($1 <= t <= 100$), số lượng bộ test.
  - t dòng tiếp theo, mỗi dòng chứa một chuỗi ký tự s ($1 <= |s| <= 10^5$) - chuỗi cần xử lý. Chuỗi s chỉ chứa các ký tự chữ cái thường và số.
- **Output:** Với mỗi chuỗi đầu vào, in ra trên một dòng chuỗi kết quả.

- **Ví dụ:**

Input	Output
3 programming abcabc 112131123	progamin abc 123

- **Kiểu dữ liệu cấu trúc - struct** là kiểu dữ liệu cho phép lưu trữ các biến thuộc nhiều kiểu dữ liệu khác nhau trong một biến duy nhất
- Là kiểu dữ liệu người dùng định nghĩa, có thể chứa biến, hàm, ...
- Trình biên dịch chỉ định vùng nhớ liên tiếp cho các biến trong **struct**
- Cú pháp:

```
1  struct struct_name {
2      data_type1 member1;
3      data_type2 member2 = value; // Init value
4      ...
5
6      data_type function_name(data_type arg1, data_type arg2, ...)
         {
7            // Function
8      }
9  };
10 struct_name a, b[1000];
11 struct_name *c;
```

- Truy cập biến và hàm trong struct:

```
struct_name a;
a.member1 = value;
a.function_name(arg1, arg2, ...);

struct_name b[1000];
b[100].member1 = value;
b[100].function_name(arg1, arg2, ...);

struct_name* c;
c->member1 = value;
c->function_name(arg1, arg2, ...);
(*c).member1 = value;
(*c).function_name(arg1, arg2, ...);
```

- **struct** là value type, khi truyền vào hàm thì truyền theo giá trị, không thay đổi giá trị của biến gốc
- **struct** và **class** chỉ khác nhau ở quyền truy cập mặc định và kiểu kế thừa mặc định - chi tiết sẽ trình bày ở chương II

- **Tên bài:** CUỘC THI ICPC PTIT
- **Đề bài:** Để chuẩn bị cho kì thi lập trình ICPC PTIT 2024, câu lạc bộ ITPTIT quyết định mở một cuộc thi giữa các thành viên trong câu lạc bộ. Chủ nhiệm CLB muốn càng nhiều thành viên thành gia càng tốt. Tuy nhiên, một thành viên sẽ cảm thấy kém cỏi và không tham gia, nếu như trong cuộc thi có một người GPA hơn mình ít nhất k điểm. Các bạn hãy giúp chủ nhiệm CLB tìm số lượng thành viên nhiều nhất có thể tham gia.
- **Input:**
  - Dòng đầu tiên chứa hai số nguyên N và k ($1 <= N <= 10^5$, $1 <= k <= 10^9$).
  - N dòng tiếp theo chứa hai số nguyên mỗi dòng a_i, b_i ($0 <= a_i, b_i <= 10^9$) - có b_i thành viên GPA là a_i.
- **Output:** In ra trên một dòng số thành viên tối đa có thể tham gia cuộc thi.

- **Ví dụ:**

Input	Output
4 5 80 10 1 100 120 25 80 3	100

- Thư viện: **fstream**
- Namespace: **std**
- Cú pháp:

```
1 ifstream fin("input.txt"); // Read text file
2 ofstream fout("output.txt"); // Write text file
3 fstream fio("file.txt"); // Read and write text file
```

- Cách sử dụng chi tiết: tham khảo fstream

- Thư viện: **cstdio**
- Namespace: Không thuộc namespace nào
- Cú pháp:

```
1 freopen("input.txt", "r", stdin); // Read text file
2 freopen("output.txt", "w", stdout); // Write text file
```

- Kém linh hoạt hơn so với **fstream**, nhưng hữu ích trong môn học hay lập trình thi đấu vì giúp tiết kiệm thời gian
- Hỗ trợ đọc và ghi file nhị phân

- **const** là từ khóa dùng để khai báo một biến hoặc con trỏ hằng số, không thể thay đổi giá trị sau khi khởi tạo
- Cú pháp:

```
const data_type variable_name = value;
const data_type* pointer_name = &variable_name;
```

- **const** Có thể sử dụng với tham số hàm:

```
void function_name(const data_type arg, ...)
```

- Nếu thay đổi giá trị của biến hoặc con trỏ hằng số sẽ báo lỗi

- **auto** là từ khóa dùng để tự động xác định kiểu dữ liệu của biến dựa vào giá trị khởi tạo
- Cú pháp:

```
auto variable_name = value;
```

- Ví dụ:

```
auto a = 5; // int
auto b = 5.5; // double
auto c = a; // int
auto* ptr1 = &a; // int*
int d = 10; // const int
const auto* ptr2 = &d; // const int*
```

- Không thể sử dụng auto mà không khởi tạo giá trị, giá trị có thể là biến, hằng số, con trỏ, ...
- C++ 11 trở lên mới hỗ trợ **auto**

Định nghĩa phạm vi và vòng đời của biến và/hoặc các hàm bên trong chương trình

- **auto**
- **register**
- **static**
- **extern**
- **mutable** - chỉ áp dụng cho các đối tượng class

- **auto** là mặc định và chỉ áp dụng cho các biến cục bộ, tự động giải phóng bộ nhớ khi ra khỏi phạm vi của khối lệnh mà biến được khai báo (bộ nhớ lưu trữ là stack)
- Ví dụ:

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

auto int a = 5; // Error

int main() {
    auto int b = 5; // OK, same as int b = 5
    return 0;
}
```

- **register** chỉ áp dụng cho biến cục bộ, được sử dụng khi biến yêu cầu truy cập nhanh (lưu trong thanh ghi CPU), vì vậy không thể lấy địa chỉ của biến
- Ví dụ:

```
1 register int a = 5;
```

- **static** có thể sử dụng cho cả biến và hàm
- Biến **static** cục bộ tồn tại trong toàn thời gian chương trình chạy
- Ví dụ:

```
1  void increment() {
2      static int count = 0;  // count is initialized only once
3      count++;
4  }
5
6  int main() {
7      increment(); // Output: 1
8      increment(); // Output: 2
9      increment(); // Output: 3
10     return 0;
11 }
```

- Biến **static** toàn cục chỉ có thể truy cập trong file được khai báo
- Hàm **static** chỉ có thể truy cập trong file được khai báo
- Ví dụ:

```cpp
// File1.h
#include <iostream>
using namespace std;

static int count = 0;  // Access only in File1.h

static void display() {  // Access only in File1.h
    cout << "This is a static function.\n";
}

int main() {
    increment(); // Output: 1
    increment(); // Output: 2
    increment(); // Output: 3
    return 0;
}
```

- **extern** có thể áp dụng cho cả biến và hàm, cho phép truy cập các biến hoặc hàm được định nghĩa trong một file khác
- Là chế độ mặc định của hàm

```cpp
// File1.cpp
int count;
extern void print_count();

int main() {
    count = 5;
    print_count(); // Output: 5
    return 0;
}

// File2.cpp
#include <iostream>
using namespace std;

extern int count;
void print_count(void) {
    cout << count << endl;
}
```

Mã nguồn 13: Ví dụ extern không sử dụng header file

```cpp
// global.h
extern int count;
extern void print_count();

// File1.cpp
#include "global.h"
#include <iostream>
using namespace std;

int count = 10;
void print_count() {
    cout << count << endl;
}

//File2.cpp
#include "global.h"

int main() {
    count = 5;
    print_count(); // Output: 5
    return 0;
}
```

Mã nguồn 14: Ví dụ extern sử dụng header file