KỸ THUẬT LẬP TRÌNH

TRẦN ĐÌNH LUYỆN

THÔNG TIN HỌC PHẦN

Thông tin cơ bản

Thời gian: học kỳ 3

Loại học phần: Bắt buộc

Số tín chỉ: 03

Mã HP: 1050276

10%

Chuyên cần

Tham gia đầy đủ, vắng quá **1/3** bị **0** điểm.

20%

Giữa kỳ

Đánh giá bằng bài tập do giáo viên ra đề.

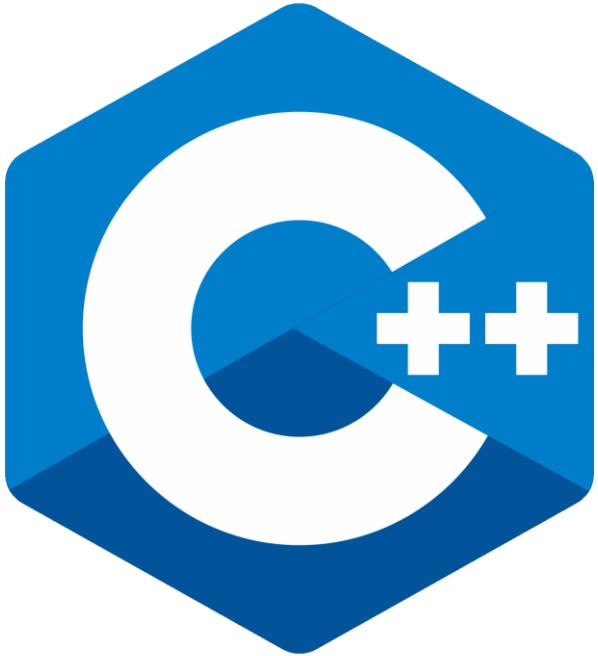
70%

Cuối kỳ

Thực hành, lập trình giải quyết bài toán cho trước.

Tổng quan

Cung cấp các kiến thức cơ bản về C++, các nguyên tắc lập trình và các phương pháp giải quyết bài toán cơ bản



01

Mở đầu

02

Các phương pháp giải quyết bài toán

03

Các nguyên tắc lập trình

04

Ngôn ngữ lập trình C++

struct chung, vào/ra (thiết bị chuẩn, file)

01

Mở đầu

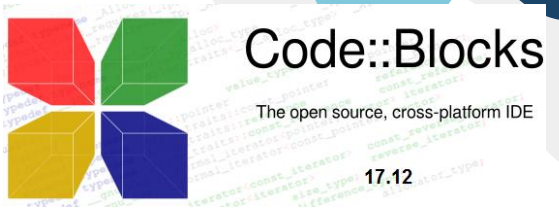
Quan điểm

- Đề cao kiến thức **cơ bản**, **nền tảng**, thiên về **tư duy**, phương pháp lập trình nhằm tạo khả năng thích ứng với các ứng dụng khác nhau, các ngôn ngữ lập trình khác.
- Các nội dung **không** có trong chương trình:
  - Lập trình hệ thống
  - Lập trình đồ họa
  - Lập trình giao tiếp với các cổng vào/ra
  - Lập trình cơ sở dữ liệu
  - Lập trình phân tán (mạng, Internet)

Phương pháp

- Nghe hiểu, làm thử, đọc, thảo luận, luyện tập
- Đọc, làm thử, (nghe), thảo luận, luyện tập
- **Chủ động học (luyện tập) thường xuyên**
- Những điều **không** nên:
  - Học thuộc lòng, học chay
  - Dựa dẫm vào các bài tập mẫu

Công cụ học tập



- Máy tính PC
- Công cụ lập trình: **CodeBlock**, Dev C++,...
- Tài liệu tham khảo:
  - Fundamentals of C++ Programming (free book)
  - The art of computer programming (3th Edition)
  - The Clean Code, Robert Martin

Kỹ thuật lập trình

- Kỹ thuật thực thi một giải pháp phần mềm (struct dữ liệu + giải thuật) dựa trên nền tảng một phương pháp luận (methodology) và một hoặc nhiều ngôn ngữ lập trình phù hợp với yêu cầu đặc thù của ứng dụng.
- Kỹ thuật lập trình:
  - Tư tưởng thiết kế + Kỹ thuật mã hóa
  - struct dữ liệu + Giải thuật + Ngôn ngữ lập trình
- #Phương pháp phân tích & thiết kế (A&D)

Lập trình?

Viết chương trình tính 20!	Viết hàm tính **N!**
Viết chương trình kiểm tra 111 có phải số nguyên tố hay không?	Viết hàm kiểm tra **N** có phải là số nguyên tố hay không?
Viết chương in ra 50 số nguyên tố đầu tiên	Viết chương in ra **N** số nguyên tố đầu tiên
Giải bài toán: "Vừa gà vừa chó, bó lại cho tròn, ba mươi sáu con, một trăm chân chẵn"	Giải bài toán: "Vừa gà vừa chó, bó lại cho tròn, Vừa đủ **X** con, Và **Y** chân chẵn"

Lập trình tốt?

Đúng

Giải đúng đề bài, được khách hàng chấp nhận

Tin cậy

Chương trình chạy đúng
Ít lỗi (số lượng ít, cường độ thấp)

Hiệu suất

Chương trình nhỏ gọn, ít bộ nhớ
Tốc độ nhanh, sử dụng ít thời gian CPU

Hiệu quả

Thời gian lập trình ngắn,
Khả năng bảo trì dễ dàng
Giá trị sử dụng lại lớn
Sử dụng đơn giản, thân thiện
Nhiều chức năng tiện ích

Ví dụ: Tính giai thừa

- Viết **chương trình** hay viết **hàm**?

```
int giaithua(int N);
```

- Cài đặt:
  - Đệ quy (recursive):
  - Lặp (iterative)

```
int giaithua(int N){
    if (N>1)
        return N*giaithua(N-1);
    return 1;
}
```

```
int giaithua(int N){
    int fact = 1;
    while (N > 1)
        fact *=N--;
    return fact;
}
```

Lập trình tốt như thế nào?

- Học cách tư duy và phương pháp lập trình
  - Tư duy toán học, tư duy logic, tư duy có struct, tư duy hướng đối tượng, tư duy tổng quát
  - Tìm hiểu về struct dữ liệu và giải thuật
- Hiểu về máy tính
  - Tương tác giữa CPU, chương trình và bộ nhớ
  - Cơ chế quản lý bộ nhớ
- Nắm vững ngôn ngữ lập trình
  - Biết rõ các khả năng và hạn chế của ngôn ngữ
  - Kỹ năng lập trình (đọc thông, viết thạo)
- Tự rèn luyện trên máy tính

02

Các phương pháp giải quyết bài toán

Phương pháp tiếp cận bài toán

Top-Down

Phương pháp này phù hợp với đối tượng chưa có hứng thú với lập trình, cần có cảm hứng từ việc nhìn thấy được kết quả ngắn hạn mà lập trình mang lại để tạo nên niềm yêu thích trước.

Bottom-Up

Phương pháp này còn có thể hiểu là 1 cách quy nạp, từ những điều cơ bản và chi tiết sau quá trình dài tổng hợp thành 1 hướng đi và ứng dụng.

Top-Down Approach

- Học bằng cách tạo ra những phần mềm (hoặc phần nhỏ của) thực sự.
- Mong muốn mình làm được những thứ hấp dẫn ngay lập tức như một trang web hay một game 2D, 3D...
- Làm theo một bài tutorial - dài và cụ thể, một tutorial được hướng dẫn rất chi tiết. Nếu bạn thực hiện theo các bước chính xác, bạn được đảm bảo sẽ tạo ra một cái gì đó (thú vị với bạn).
- Ví dụ tạo 1 website sử dụng CMS

Top-Down Approach - Advantages

- Nhìn thấy sản phẩm nhanh chóng
- Tạo ra một phần mềm trong thời gian ngắn ->**Vui sướng**
- Tạo động lực vượt qua quá trình học tập.

Top-Down Approach - Disadvantages

- Không nắm những định nghĩa, nguyên tắc cơ bản.
- Không hiểu được là sản phẩm (phần mềm) của bạn hoạt động như thế nào.
- Có thể phải vật lộn để giải quyết vấn đề nào đó của nó, khi đi chệch hướng, dù là nhỏ nhất, bạn có thể sẽ không thể hoàn thành nó và không thể chẩn đoán được vấn đề mắc phải.
- Bị ném thẳng vào chỗ sâu nhất của bể bơi trước khi bạn biết bơi.

Bottom-Up Approach

- Tìm hiểu tất cả các khái niệm cơ bản về lập trình.
- Hướng tiếp cận này phổ biến hơn trong các khóa học lập trình chính thức trong các trường đại học hoặc cao đẳng.
- Bắt đầu từ con số 0 và học một khái niệm tại thời điểm đó. Ý tưởng là để xây dựng một nền tảng vững chắc các kỹ năng lập trình chung, mà có thể được sử dụng để làm bất kỳ loại phần mềm nào.

Bottom-Up Approach - Advantages

- Học các kỹ năng lập trình tổng quát thực sự. Không quan trọng nếu bạn tạo một game hoặc trang web tương tác - khái niệm cơ bản về lập trình có thể áp dụng cho mọi thứ.
- Mỗi khái niệm riêng lẻ thì dễ học hơn, vì bạn có thể học nó một cách độc lập.
- Các khái niệm cơ bản được nắm vững (như giá trị và biến), trước khi các khái niệm phức tạp hơn (như các hàm) được xây dựng bên trên.

Top-Down Approach - Disadvantages

- Phải mất một thời gian dài để học tất cả các kỹ năng cần thiết để làm được một cái (phần mềm, chương trình...) gì đó đáng kể.
- Mô hình truyền thống theo hướng tiếp cận này:
  1. strings, integers, variables...
  2. booleans, branching và looping
  3. arrays và structs
  4. functions và control flow
  5. Thay đổi cách code, bắt đầu học về classes, instances, methods, inheritance và về OOP (hướng đối tượng).
  6. Viết code mới theo những khái niệm OOP

Theo bạn cách tiếp cận nào tốt hơn?

Thảo luận

Các phương pháp lập trình

- Lập trình tuần tự (sequential programming)
- Lập trình có struct (structured programming)
- Lập trình module (modular programming)
- Lập trình hướng đối tượng (object-oriented programming)
- Lập trình thành phần (component-based programming)
- Lập trình thời gian thực (real-time programming)

Lập trình tuần tự (sequential programming)

- Phương pháp cổ điển nhất, bằng cách liệt kê các lệnh
- kế tiếp, mức trừu tượng thấp
- Kiểm soát dòng mạch thực hiện chương trình bằng các lệnh rẽ nhánh, lệnh nhảy, lệnh gọi chương trình con (subroutines)
- Ngôn ngữ đặc thù: ngôn ngữ máy, ASSEMBLY, BASIC, IL (Instruction List), STL (Statement List), LD, LAD (Ladder Diagram).

Lập trình tuần tự (sequential programming)

- Ví dụ: tính giai thừa

```
1:   MOV AX, n
2:   DEC n
3:   CMP n, 1
4:   JMPI
5:   MUL AX, n
6:   JMP 2
7:   MOV n, AX
8:   RET
```

Lập trình tuần tự (sequential programming)

- Ưu điểm:
  - Tư duy đơn giản
  - Lập trình ở mức trừu tượng thấp, nên dễ kiểm soát sử dụng tài nguyên
  - Có thể có hiệu suất cao
  - Có thể thích hợp với bài toán nhỏ, lập trình nhúng, lập trình hệ thống
- Nhược điểm:
  - Chương trình khó theo dõi -> dễ mắc lỗi
  - Khó sử dụng lại
  - Hiệu quả lập trình thấp
  - Không thích hợp với ứng dụng qui mô lớn

Lập trình có struct (structured programming)

- struct hóa dữ liệu (xây dựng kiểu dữ liệu) và struct hóa chương trình để tránh các lệnh nhảy.
- Phân tích và thiết kế theo cách từ trên xuống (top-down)
- Thực hiện từ dưới lên (bottom-up)
- Chỉ sử dụng các struct điều khiển tuần tự, rẽ nhánh (if else), lặp (while) và thoát (exit).
- Các ngôn ngữ đặc thù: PASCAL, ALGO, FORTRAN, C, SFC (Sequential Funtion Charts),...

Lập trình có struct (structured programming)

- Ví dụ 1: tính giai thừa (slide 12)
- Ví dụ 2: Quản lý sinh viên:

```
struct Date { int Day, Month, Year; };
struct Student{
    string name;
    Date dob;
    int code;
};
typedef Student* Students; // cấu trúc mảng
Students create(int max_items, int item_size );
void destroy(Students lop);
void add(Students lop, Student sv);
void delete(Students lop, Student sv);
Student find(Students lop, int code);
```

Lập trình module (modular programming)

- Một dạng cải tiến của lập trình có struct. Chương trình được struct nghiêm ngặt hơn, đơn vị là module.
- Module:
  - Một đơn vị struct độc lập, được chuẩn hóa dùng để tạo lập một hệ thống.
  - Mỗi module bao gồm phần giao diện (mở) và phần thực hiện (che giấu)
  - Các module giao tiếp với nhau thông qua các giao diện được đặc tả rất chính xác.
- Ví dụ ngôn ngữ tiêu biểu: Modula-2, xây dựng trên cơ sở PASCAL, do Niclaus Wirth thiết kế năm 1977.

Lập trình hướng đối tượng (OOP)

- Xây dựng chương trình ứng dụng theo quan điểm dựa trên các struct dữ liệu trừu tượng (lớp), các thể nghiệm của các struct đó (đối tượng) và quan hệ giữa chúng (quan hệ lớp, quan hệ đối tượng).
- Ba nguyên lý cơ bản:
  - Đóng gói dữ liệu (data encapsulation)
  - Dẫn xuất/thừa kế (subtyping/inheritance)
  - Đa hình/đa xạ (polymorphism)
- Ví dụ ngôn ngữ hỗ trợ tiêu biểu: C++, C#, Java, ADA,...

Lập trình hướng đối tượng (OOP)

- Ví dụ quản lý sinh viên

```
class Date{
    int Day, Month, Year;
public:
    void setDate(int, int, int);
    ...
};
class Student{
    string name;
    Date dob;
    int code;
public:
    Student(string n, Date d, int c);
    ...
};
class StudentList{
    Student* list;
public:
    void addStudent(Student*);
    ...
};
```

Lập trình thành phần

- Phương pháp xây dựng phần mềm dựa trên các thành phần "IC" có sẵn, hoặc tạo ra các IC đó.
- Tiến hóa từ lập trình hướng đối tượng
- Hầu hết các ứng dụng Windows và ứng dụng Internet ngày nay được xây dựng theo phương pháp luận này
- Các ngôn ngữ tiêu biểu:
  - C/C++, C#
  - — Delphi, Visual Basic
  - — Script, HMTL, XML,...
  - ...

Lập trình thời gian thực

- Xây dựng phần mềm đáp ứng tính năng thời gian thực của hệ thống, ví dụ các hệ thống điều khiển
- Đặc thù:
  - Lập trình cạnh tranh (đa nhiệm, đa luồng)
  - Cơ chế xử lý sự kiện
  - Cơ chế định thời
  - Đồng bộ hóa quá trình
  - Hiệu suất cao
- Ngôn ngữ lập trình: ASM, C/C++, ADA,...
- Cần sự hỗ trợ của nền cài đặt như hệ điều hành, phần cứng, mạng.

03

Các nguyên tắc lập trình

Các nguyên tắc cơ bản

- Trừu tượng hóa
  - Chắt lọc ra những yếu tố quan trọng, bỏ qua những chi tiết kém quan trọng
- Đóng gói
  - Che giấu và bảo vệ các dữ liệu quan trọng qua một giao diện có kiểm soát
- Module hóa
  - Chia nhỏ đối tượng/vấn đề thành nhiều module nhỏ để dễ can thiệp và giải quyết
- Phân cấp
  - Phân hạng hoặc sắp xếp trật tự đối tượng theo các quan hệ trên dưới

Bạn ĐÃ code như thế nào?

- **Không** quan tâm đến coding convention, đặt tên biến vô tội vạ a1, a2.
- **Không** quan tâm đến module hóa, chỉ cần code chạy được là ok, nhìn mớ code hỗn độn như đống rác
- **Không** có comment
- **Không** chịu review code, code xong chạy luôn, review code tốt có thể giảm thiểu được 20% bug
- **Không** nghĩ đến optimize code, không sử dụng struct dữ liệu phù hợp dẫn tới tốn memory, low performance

Bạn ĐÃ code như thế nào?

- **Thích** code như rồng bay phượng múa dù xử lý đơn giản
- **Thích** Google search and Copy paste, mà không hiểu hết xử lý của code đó
- **Thích** cắm đầu code ngay, mà không nghiên cứu kĩ giải pháp, tìm hiểu requirement.

Một số nguyên tắc trong lập trình

- KISS (Keep It Simple, Stupid): “Keep it simple as simpe as possible, but no simpler” (*Albert Einstein*)
- DRY (Don’t repeat yourself): đừng lặp lại những gì giống nhau
- YAGNI (You aren’t gonna need it): chức năng (phần) ấy rồi sẽ không cần thiết.

Khuyến nghị

- Code sao cho đơn giản nhất (simple)
  - Dễ tuân thủ, dễ tiếp cận, dễ improve, maintaion,..
- Code sao cho chọ gàng, sạch sẽ (clear & clean)
  - Tuân thủ bộ nguyên tắc, đặt tên biến, hàm, lớp dễ hiểu, các hàm có input/output rõ ràng.
- Code sao cho có thể tái sử dụng (reusable)
- Chia tách code theo class/module/package độc lập (decoupling)
  - Mỗi class/module/package chỉ nên làm các công việc thuộc về domain của nó

Một số quy tắc đặt tên trong C++

- Quy tắc chung
  - Tên phải bắt đầu bằng chữ cái hoặc dấu gạch dưới (a,_a)
  - Không được trùng các từ khóa (const, char, int,...)
  - Không được có các toán tử
  - Hai biến trong cùng 1 hàm không được trùng nhau
  - Không có dấu cách trong biến

Một số quy tắc đặt tên trong C++

- Hằng (const): viết hoa toàn bộ hoặc có chữ “k” phía trước
- Tệp (file): tất cả tên tệp là chữ thường có thể bao gồm dấu gạch dưới (_) hoặc dấu gạch ngang (-)
- Hàm (function): thường viết hoa chữ cái đầu của từ mới
- Lớp (class): tên nên là 1 danh từ và viết hoa chữ cái đầu của tất cả các từ.
- Biến (variable): đặt tên có ý nghĩa, dễ hiểu, có thể viết thường, dung dấu gạch dưới (_) hoặc viết hoa từ mới,... (tuân thủ theo cách đặt tên chung)

Ví dụ:

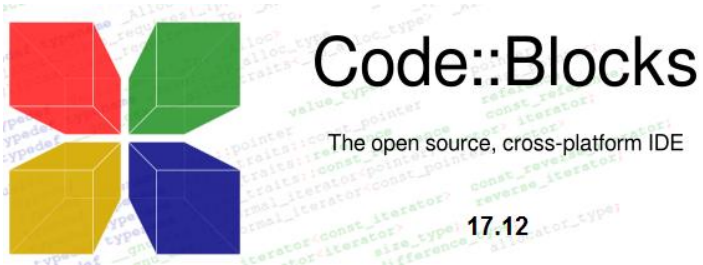
```
#include <iostream>
#define PI 3.14
using namespace std;
const int kDay = 7;
struct Date{int Day, Month, Year;};
struct Student{
    string name;
    Date dob;
    int code;
};
int giaiThua(int intNum);
int main()
{
    cout << "Hello world!" << endl;
    cout << giaiThua(10) << endl;
    return 0;
}
int giaiThua(int intNum){
    int fact = 1;
    while (intNum > 1)
        fact *=intNum--;
    return fact;
}
```

04

C/C++

Môi trường/công cụ

- IDE (Integrated Development Environment): hỗ trợ toàn bộ các bước phát triển chương trình.
- Các công cụ:
  - Trình soạn thảo (Editor)
  - Trình biên dịch (Compiler)
  - Trình liên kết (Linker)
  - Trình nạp (Loader)
  - Trình gỡ rối (Debugger)
  - Trình quản lý dự án (Project Manager)
- Tải và cài đặt: www.codeblock.org (chọn phiên bản có bộ biên dịch đi kèm (*mingw*))



Chương trình tính giai thừa (C)

```c
#include <stdio.h>
#include <stdlib.h>
int giaiThua(int);
int main()
{
    int num;
    printf("Hello C\n");
    printf("Enter an integer number:");
    scanf("%d",&num);
    printf("%d! = %d",num,giaiThua(num));
    return 0;
}
int giaiThua(int intNum){
    int fact = 1;
    while (intNum > 1)
        fact *=intNum--;
    return fact;
}
```

Chương trình tính giai thừa (C++)

```cpp
#include <iostream>
using namespace std;
int giaiThua(int);
int main()
{
    int num;
    cout << "Hello C++" << endl;
    cout << "Enter an integer number:";
    cin >> num;
    cout << num << "! = "<< giaiThua(num) << endl;
    return 0;
}
int giaiThua(int intNum){
    int fact = 1;
    while (intNum > 1)
        fact *=intNum--;
    return fact;
}
```

Quy tắc soạn thảo

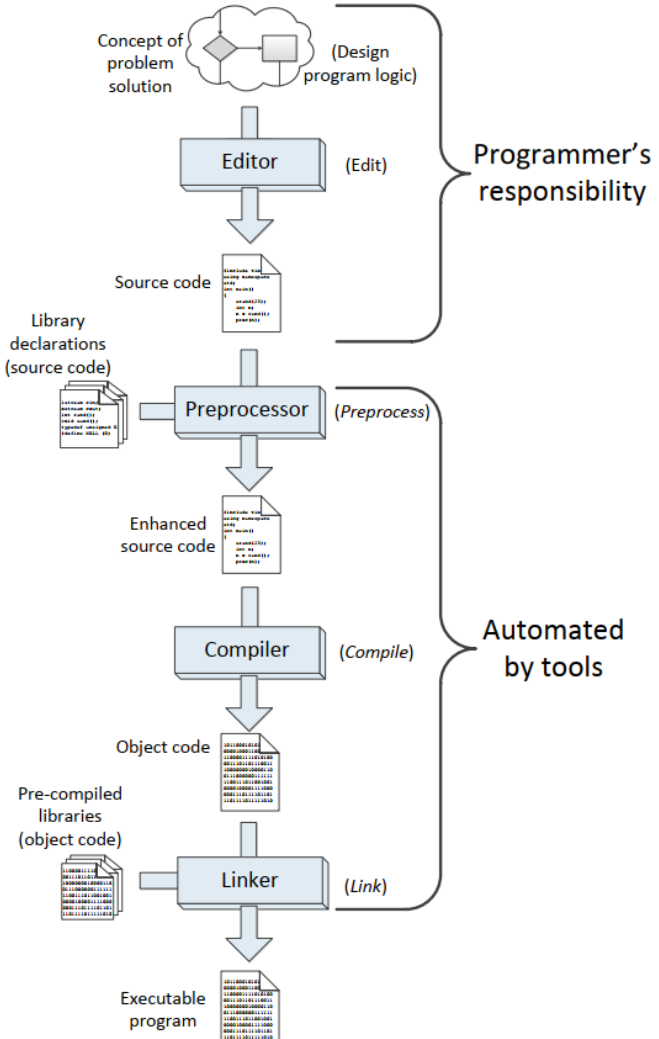
- Tên biến, tên hàm, tên kiểu mới: tuân thủ nguyên tắc đặt tên chung (slide 40)
- Sau mỗi câu lệnh có chấm phảy;
- Đoạn { ... } được coi là nhóm lệnh, không có dấu chấm phảy sau đó, trừ khi khai báo kiểu
- struct mã nguồn theo kiểu phân cấp => dễ đọc
- Bổ sung chú thích hợp lý (/* ...*/ hoặc //)
- Chia một file lớn thành nhiều file nhỏ

Các từ khóa (keywords) trong C++

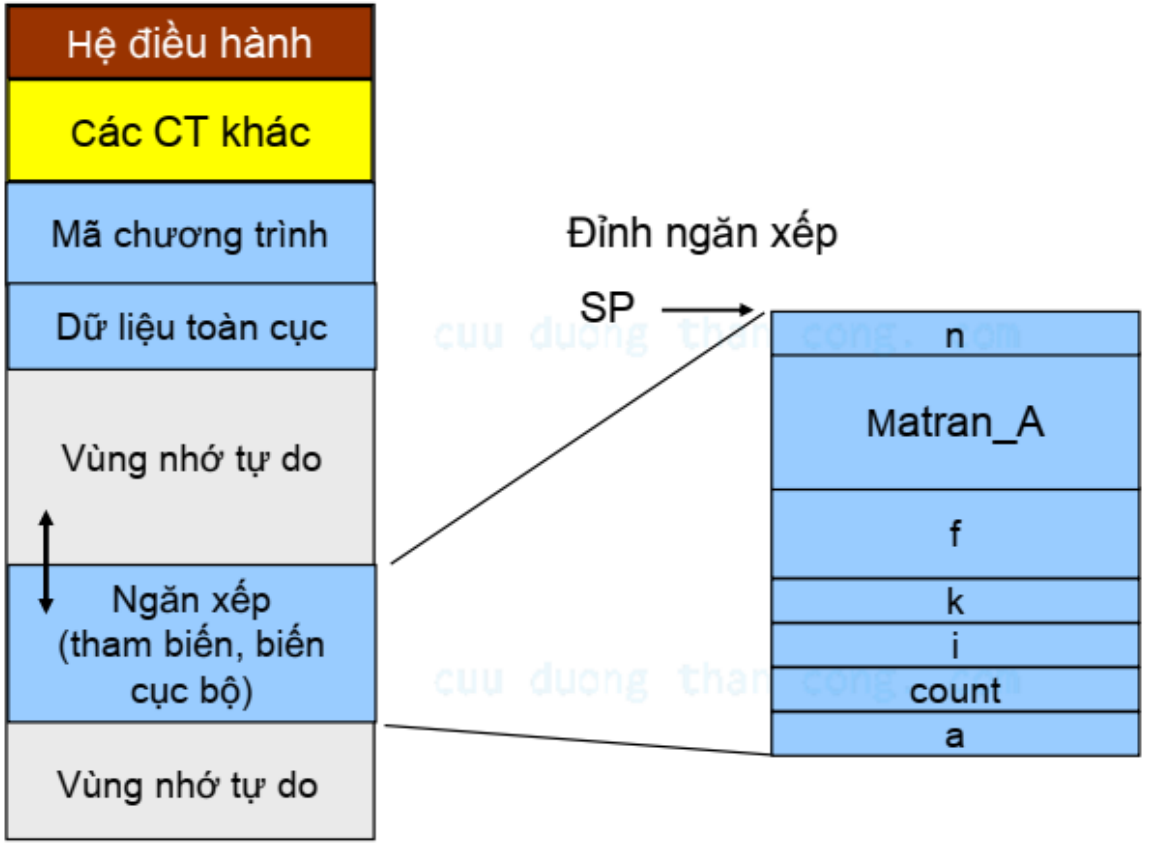
asm	**auto**	bool	break
case	**catch**	char	**class**
const	**const_cast**	continue	default
delete	else	extern	do
enum	**false**	double	**explicit**
float	**dynamic_cast**	**export**	for
friend	goto	if	**inline**
int	long	**mutable**	**namespace**
new	**operator**	**private**	**protected**
public	register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

cuu duong than cong. com

Quá trình từ lúc soạn thảo chương trình nguồn đến chương trình thực thi



Tổ chức bộ nhớ



Các kiểu dữ liệu cơ bản của C++

Kiểu	Kích cỡ thông dụng (tính bằng bit)	Phạm vi tối thiểu
char	8	-127 to 127
signed char	8	-127 .. 127
unsigned char	8	0 .. 255
int	16/32	-32767 .. 32767
signed int	16/32	-nt-
unsigned int	16/32	0 .. 65535
short	16	-32767 .. 32767
signed short	16	nt
unsigned short	16	0 .. 65535
long	32	-2147483647 .. 2147483647
signed long	32	nt
unsigned long	32	0 .. 4294967295
float	32	Độ chính xác 6 chữ số
double	64	Độ chính xác 10 chữ số
long double	80	Độ chính xác 10 chữ số
bool (C++)	-	-
wchar_t (C++)	16	-32767 .. 32767

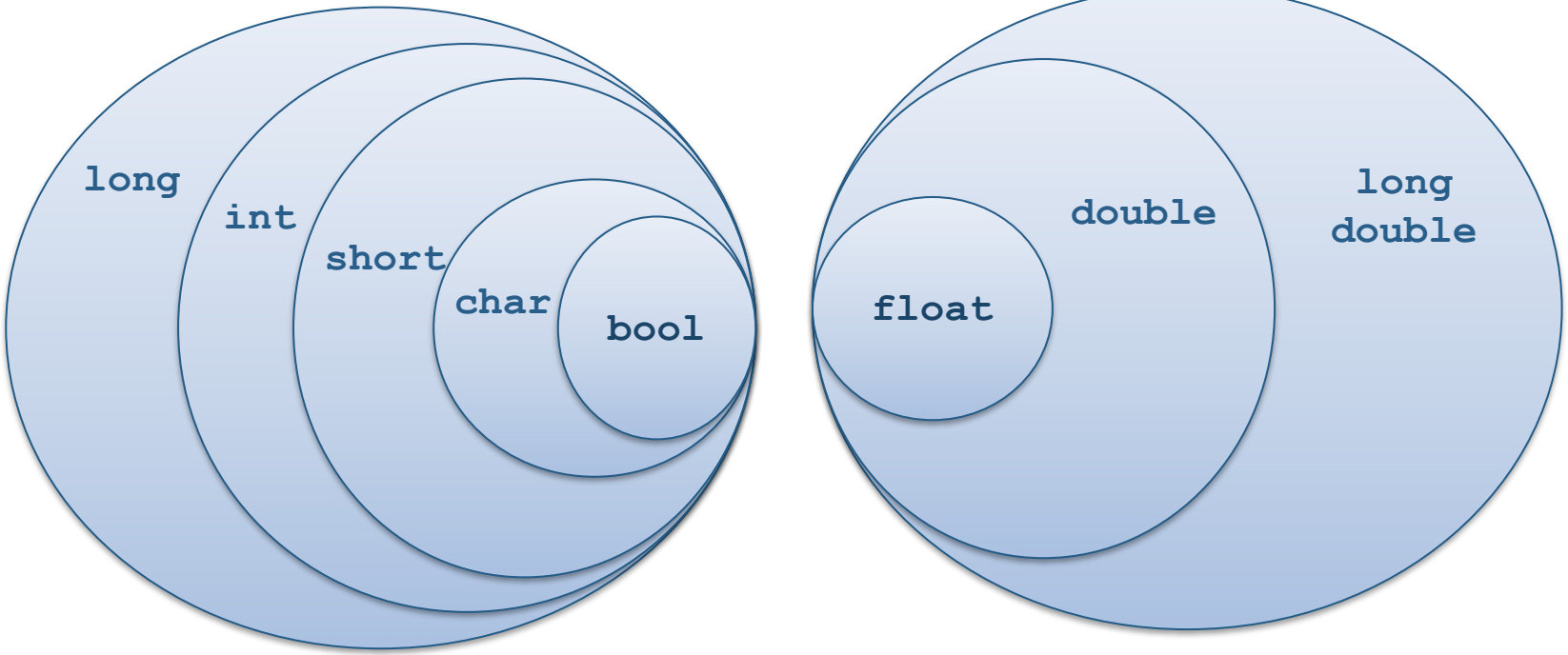
Các phép toán cơ bản

Phép toán	Ký hiệu	Kiểu nguyên	Kiểu số thực	Kiểu bool
Gán	=	X	X	X
Số học	+, -, *, /, +=, -=, *=, /=	X	X	x
	%, %=	X		x
	++, --	X		x
So sánh	>, <, >=, <=, ==, !=	X	X	X
Logic	&&, ||, !	X	X	X
Logic bit	&, |, ^, ~ &=, |=, ^=	X		x
Dịch bit	<<,>>, <<=,>>=	X		x
Lựa chọn	? :	X	X	X

Tương thích kiểu và chuyển đổi kiểu

- Tương thích kiểu => Tự động chuyển đổi kiểu
  - Giữa các kiểu số nguyên với nhau (lưu ý phạm vi giá trị)
  - Giữa các kiểu số thực với nhau (lưu ý độ chính xác)
  - Giữa các kiểu số nguyên và số thực (lưu ý phạm vi giá trị và độ chính xác)
  - Kiểu bool sang số nguyên, số thực: true => 1, false => 0
  - Số nguyên, số thực sang kiểu bool: ≠ 0 => true, 0 => false
- Nếu có lỗi hoặc cảnh báo => khắc phục bằng cách ép chuyển đổi kiểu:
  - VD:
```
i = int(2.2) % 2;
j = (int)2.2 + 2; // C+
```

Tương thích kiểu và chuyển đổi kiểu



Khai báo biến

- Cú pháp: **kiểu_dữ_liệu tên_biến** [= **giá_trị_khởi_tạo**];
- Ví dụ:

```
char      c = 'N';
bool      b = true; //bool b{true}; bool b{!false};
int       kq;
double    d;
long      count, i=0;
unsigned  vhexa=0x00fa;
unsigned  voctal=082;
```

- Có thể khai báo tại bất kỳ vị trí nào cần (*Ngôn ngữ C toàn bộ biến phải được khai báo ở đầu thân hàm*)
- Phân loại biến: biến toàn cục, biến cục bộ, tham biến

Khai báo biến

```
#include <iostream>
using namespace std;
int N = 1;
void main() {
    char c = 'N';
    do {
        cout << "\nEnter a number > 0:";
        cin >> N;
        int kq = factorial(N); // C++ only!
        ...
    } while (c == 'y' || c == 'Y')
}
int factorial(int n) {
    int kq = 1;
    while (n > 1)
    kq *= n--;
    return kq;
}
```

Hằng số

Kiểu	Ví dụ
int	1 123 21000 -234 0x0A 081
long int	35000L -34l -234L 0x0AL 081L
unsigned int	10000U 987u 40000u
float	123.23F 4.34e-3f .1f
double	123.23 1.0 -0.9876324 .1e-10
long double	1001.2L
char	'A' 'B' ' ' 'a' '\n' '\t' '\b'
Bool	true false

Vào/ra chuẩn

- Sử dụng các đối tượng:
  - `cin` thuộc kiểu `istream`, tương đương với `stdin`
  - `cout` thuộc kiểu `ostream`, tương đương với `stdout`
  - `cerr` thuộc kiểu `ostream`, tương đương với `stderr`
  - Ngoài ra còn các đối tượng `wcin`, `wcout`, `wcerr` để làm việc với Unicode

```
int main()
{
    int i;
    float a[10];
    std::cout << "Nhap i va a[i]: ";
    std::cin >> i >> a[i];
    std::cout << "a[" << i << "]=" << a[i] <<
    return 0;
}
```

```
using namespace std;
int main()
{
    int i;
    float a[10];
    cout << "Nhap i va a[i]: ";
    cin >> i >> a[i];
    cout << "a[" << i << "]=" << a[i] <<endl;
    return 0;
}
```

Vào/ra chuẩn

```
int main()
{
    int i;
    float a[10];
    std::cout << "Nhap i va a[i]: ";
    std::cin >> i >> a[i];
    std::cout << "a[" << i << "]=" << a[i] << st
    return 0;
}
```

```
using namespace std;
int main()
{
    int i;
    float a[10];
    cout << "Nhap i va a[i]: ";
    cin >> i >> a[i];
    cout << "a[" << i << "]=" << a[i] <<endl;
    return 0;
```

```
using namespace std;
int main()
{
    int i;
    float a[10];
    cout << "Nhap i va a[i]: ";
    if (cin >> i >> a[i]){
        cout << "a[" << i << "]=" << a[i] <<endl;
        cout.flush();
    }else cerr << "Nhap du lieu loi!" << endl;
    return 0;
}
```

Vào/ra chuẩn

- Đọc 1 dòng:

```
string s;
getline(cin,s);
```

```
using namespace std;
int main()
{
    string s;
    cout << "Nhap vao 1 chuoi: ";
    getline(cin,s);
    cout <<"Chuoi vua nhap la: " << s ;
    return 0;
}
```

```
Nhap vao 1 chuoi: hello
Chuoi vua nhap la: hello
```

```
using namespace std;
int main()
{
    string s;
    cout << "Nhap vao 1 chuoi: ";
    cin >> s;
    cout <<"Chuoi vua nhap la: " << s ;
    return 0;
}
```

What's the difference?

Định dạng dữ liệu xuất

- Các hàm thay đổi định dạng:
  - `setf(fmtflags flag, fmtflags mask)`: thay đổi các cờ định dạng
    - `dec/hex/oct`: số nguyên hệ cơ số 10/16/8 (`basefield`)
    - `fixed/scientific`: số thực thập phân hoặc khoa học (`floatfield`)
    - `internal/left/right`: căn lề (`adjustfield`)
  - `width(int w)`: thay đổi độ rộng của trường
  - `precision(int p)`: thay đổi độ chính xác

Định dạng dữ liệu xuất

```
using namespace std;
int main()
{
    float f=34.5678;
    cout.width(15);
    cout.setf(ios::right, ios::adjustfield);
    cout.setf(ios::scientific, ios::floatfield);
    cout.precision(3);
    cout << f;
    return 0;
}
```

Định dạng dữ liệu xuất

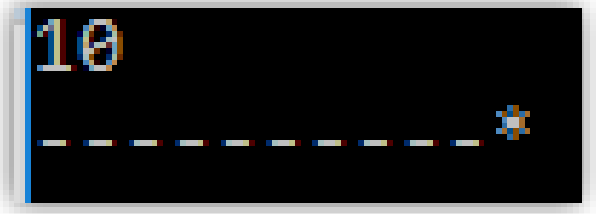
- Dùng các manipulator:
  - Baogồm: `left, right, dec, hex, oct, fixed, scientific, setprecision(p), setw(w), setfill(ch), setiosflag(flags), endl, ends, flush`

```
int main()
{
    float f=34.56789;
    cout.width(15);
    cout.setf(ios::right, ios::adjustfield);
    cout.setf(ios::scientific, ios::floatfield);
    cout.precision(3);
    cout << f << endl;
    cout << setw(15) << right << fixed << setprecision(3) << f;

    return 0;
}
```

Bài tập

1. Viết chương trình nhập vào 1 số nguyên dương N (<20) và in ra N ký tự - và 1 ký tự * (tham khảo hình bên dưới)
2. Viết chương trình nhập vào 1 số nguyên N ở dạng thập phân, in ra số ở dạng thập lục phân và bát phân.
3. Viết chương trình nhập vào 1 số nguyên N ở dạng thập lục phân, in ra số ở dạng thập phân và bát phân.



Các kiểu dữ liệu dẫn xuất trực tiếp

- Kiểu liệt kê
- Kiểu hằng
- Kiểu con trỏ
- Kiểu mảng
- Kiểu tham chiếu

Kiểu liệt kê (enum)

- Tương tự một kiểu nguyên, cần hạn chế phạm vi sử dụng
- Sử dụng thuận tiện bằng tên => hằng số nguyên
- Ví dụ:

```
enum Color {Red, Green, Blue};
enum WeekDay {
    Mon = 2,
    Tue, Wed, Thu, Fri, Sat,
    Sun = 1
};
enum {
    DI_MOTOR1_STARTED = 0x01,
    DI_MOTOR1_RUNNING = 0x02,
    DI_MOTOR2_STARTED = 0x04,
    DI_MOTOR2_RUNNING = 0x08,
    DI_PUMP1_STARTED = 0x10,
    DI_PUMP1_RUNNING = 0x20,
    DI_OVERLOADED = 0x40,
    DI_VALVE1_OPENED = 0x80
};
```

```
int main()
{
    enum Color c = Red; // c = Red
    WeekDay d = Tue; // OK, d = Tue
    int i=c, j=d; // j=0, i=3
    Color c2 = i+1; // Error!
    Color c3 = Color(i+1); // OK, c3 = Green
    int di1 = 0x01; // OK, but...
    int di2 = DI_MOTOR1_STARTED;// this is better
    ++c; // Error
    return 0;
}
```

Kiểu hằng (const)

- Tác dụng làm cho một biến trở thành không thay đổi được => khai báo hằng số:

const *dataType* **variableName**=*value*;

```
int main()
{
    const double pi = 3.1412; // initializing is OK!
    const int ci = 1; // initializing is OK!
    ci = 2; // error!
    ci = 1; // error, too!
    int i = ci; // const int is a subset of int
    const Color cc = Red;
    cc = Green; // error
    const double d; // potential error
    return 0;
}
```

Kiểu con trỏ

- Con trỏ thực chất là một biến mang địa chỉ của một ô nhớ (một biến khác hoặc một hàm)
- Một ô nhớ: địa chỉ ô nhớ & giá trị của ô nhớ đó (address & data).
- 2 toán tử: &: lấy địa chỉ ô nhớ; * lấy giá trị ô nhớ.

```
int i = 1;
int* p = &i; // p has the address of i
*p = 2; // i = 2
int j;
p = &j; // now p has the address of j
*p = 3; // j = 3, i remains 2
```

Kiểu con trỏ - Ví dụ

```
using namespace std;
int main()
{
  int i=10;
  int *p=&i;
  int j=*p;
  cout << &i << "=" << i << endl;
  cout << p <<"=" << *p << endl;
  cout << &j <<"=" << j << endl;
  cout << setw(20)<<setfill('-')<<'-'<<endl;
  *p=12;
  cout << &i << "=" << i << endl;
  cout << p <<"=" << *p << endl;
  cout << &j <<"=" << j << endl;
  cout << setw(20)<<setfill('-')<<'-'<<endl;
  i=15;
  cout << &i << "=" << i << endl;
  cout << p <<"=" << *p << endl;
  cout << &j <<"=" << j << endl;
  cout << setw(20)<<setfill('-')<<'-'<<endl;
  j=16;
  cout << &i << "=" << i << endl;
  cout << p <<"=" << *p << endl;
  cout << &j <<"=" << j << endl;
    return 0;
}
```

```
0x61fe34=10
0x61fe34=10
0x61fe30=10
```

```
0x61fe34=12
0x61fe34=12
0x61fe30=10
```

```
0x61fe34=15
0x61fe34=15
0x61fe30=10
```

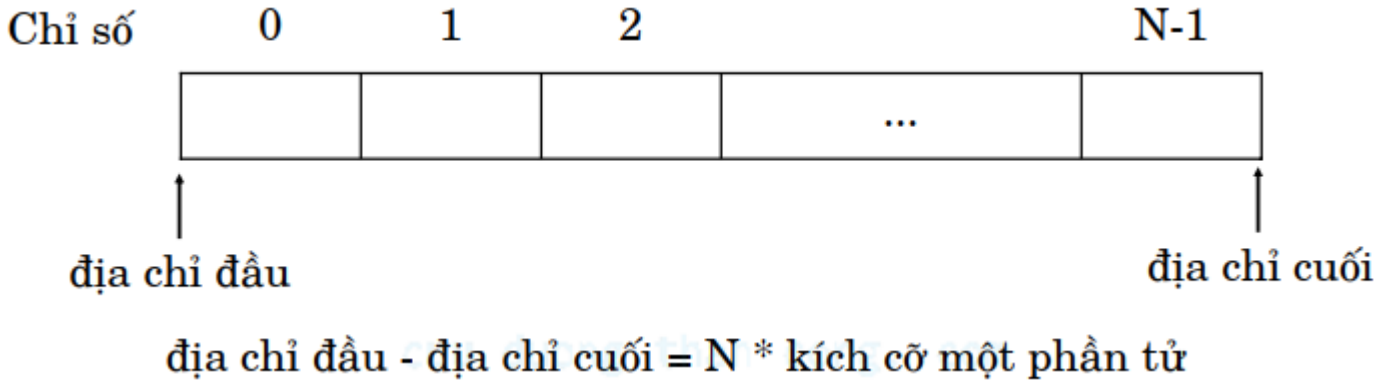
```
0x61fe34=15
0x61fe34=15
0x61fe30=16
```

Kiểu con trỏ

- Con trỏ là một biến chứa địa chỉ byte đầu của một biến dữ liệu, được sử dụng để truy cập gián tiếp dữ liệu đó
- Sau khi khai báo mà không khởi tạo, mặc định con trỏ mang một địa chỉ bất định
- Địa chỉ con trỏ mang có thể thay đổi được => con trỏ có thể mỗi lúc đại diện cho một biến dữ liệu khác
- Toán tử lấy địa chỉ của một biến (&) trả về con trỏ vào kiểu của biến => thường gán cho biến con trỏ
- Toán tử truy nhập nội dung (*) => có thể đọc hoặc thay đổi giá trị của biến đó
- Không bao giờ sử dụng toán tử truy nhập nội dung, nếu con trỏ chưa mang một địa chỉ ô nhớ

Kiểu mảng

- Đặc điểm:
  - Số lượng các phần tử cố định
  - Các phần tử có cùng kiểu
  - Các phần tử được sắp xếp kế tiếp trong bộ nhớ
  - Có thể truy nhập từng phần tử một cách tự do theo chỉ số hoặc theo địa chỉ



địa chỉ đầu - địa chỉ cuối = N * kích cỡ một phần tử

Kiểu mảng – Khai báo

- Số phần tử của mảng phải là hằng số nguyên
- Khai báo không khởi tạo:

```
int a[3];
enum {index = 5};
double b[index];
const int N = 2;
char c[N]; // C++ only
```

- Khai báo với số phần tử và khởi tạo giá trị các phần tử

```
int d[3]= {1, 2, 3};
double e[5]= {1, 2, 3};
char f[4]= {0};
```

Kiểu mảng – Khai báo

- Khai báo và khởi tạo giá trị các phần tử, số phần tử được tự động xác định

```
int a[]= {1, 2, 3, 4, 5};
double b[]= {1, 2, 3};
double c[]= {0};
char s[]= {'a'};
```

- Khai báo mảng nhiều chiều

```
double M[2][3];
int X[2][]={{1,2},{3,4},{5,6}};
short T[2][2]={1,2,3,4,5,6
```

Kiểu mảng – Sử dụng

- Giải thích các lệnh sau

```
1  int main(){
2      int a[5];
3      int b[5]= {1,3,5,7,9};
4      double c[];
5      double x = 1.0, y = 2.0;
6      double d[]= {x,y,3.0};
7      const int m=10; // C++ OK
8      double v2[m]; // C++ OK
9      a[0] = 1;
10     int i= 1;
11     a[i] = 2;
12     a[5] = 6;
13     int k = a[5];
14     a = {1,2,3,4,5};
15     a = b;
16     int M[2][3];
17     M[0][1] = 0;
18     M[0][2] = 1;
19     return 0;
```

Chuỗi - mảng ký tự

- Trong C/C++, chuỗi ký tự không phải là kiểu cơ bản, mà là một mảng các ký tự kết thúc bởi ký tự '\0'
- Với C++, chuỗi ký tự được định nghĩa bằng lớp string trong thư viện chuẩn, không sử dụng byte kết '\0'
- Ví dụ:

```
char city1[]= {'H','A','N',' ','O','I','\0'};
char city2[]= "DA NANG";
string city3="TP HCM";
```

Mảng và con trỏ

- Khi khai báo mảng, biến mảng thực chất là giữ địa chỉ phần tử đầu tiên, các phần tử tiếp theo ở các địa chỉ liên tục nhau.
- Ngoài truy cập tới các phần tử mảng thông qua chỉ số thì có thể truy cập thông qua địa chỉ.
- Ví dụ:

```
int a[5];//a giữ địa chỉ phần tử đầu tiên a[0]
a[0]=10;//tương đương *a=10;
a[1]=15;//tương đương *(a+1)=15;
```

Mảng và con trỏ

```
int main()
{
    int a[5];    // a has 5 elements with
                 // uncertain values
    int* p;
    p = a;       // p refers to a[0]
    p = &a[0];   // the same as above
    *p = 1;      // a[0]=1
    ++p;         // now p points to a[1]
    *p = 2;      // a[1]=2
    p++;         // now p points to a[2]
    *p = 3;      // a[2]=3
    p += 2;      // now p points to a[4]
    *p = 5;      // a[4] = 5
    ++p;         // OK, no problem until we dereference it
    *p = 6;      // Now is a BIG BIG problem!
    a = p;
    return 0;
}
```

Mảng và con trỏ

- Cho biết kế quả chương trình sau

```
int main()
{
    int a[5];
    int* p = a;
    *(a+1)=10;
    cout << p[1]<<endl;
    a[2]=15;
    p++;
    cout << p[1]<<endl;
    return 0;
}
```

Mảng

- Mảng là một tập hợp các dữ liệu cùng kiểu, sắp xếp liền kề trong bộ nhớ
- Có thể truy cập các phần tử mảng với biến mảng kèm theo chỉ số hoặc với biến con trỏ (theo địa chỉ của từng phần tử)
- Số phần tử của mảng là cố định
- Biến mảng (tĩnh) thực chất là một con trỏ hằng, mang địa chỉ của phần tử đầu tiên
- Có thể đặt giá trị đầu cho các phần tử của mảng qua danh sách khởi tạo, **không gán** được mảng cho nhau. Nếu cần sao chép hai mảng thì phải sử dụng hàm
- Không truy nhập với chỉ số nằm ngoài phạm vi.

Kiểu tham chiếu (C++)

- Một biến tham chiếu là một biến đại diện trực tiếp cho một biến khác (thay cho con trỏ)
- Sử dụng chủ yếu về sau trong truyền tham số cho hàm *(sẽ đề cập kỹ hơn trong phần hàm)*

```
int main()
{
    double d = 2.0;
    double& r = d; // r represents d
    double *p1 = &d, *p2 = &r;
    r = 1.0; // OK, d = 1.0
    double& r2; // error, r has to be assigned to a var.
    double& r3 = 0; // error, too
    double d2 = 0;
    r = d2; // r = 0, d=0
    r = 1.0; // r = d = 1, d2 =0
    return 0;
}
```

typedef

- Từ khóa typedef tạo ra một tên mới cho một kiểu có sẵn, không định nghĩa một kiểu mới
- Mục đích: tên mới dễ nhớ, phù hợp với ứng dụng cụ thể, dễ thay đổi về sau

```
typedef float REAL;
typedef int AnalogValue;
typedef int Vector[10];
typedef AnalogValue AnalogModule[8];
typedef int* IPointer;
AnalogValue av1 = 4500;
Vector x = {1,2,3,4,5,6,7,8,9,10};
AnalogModule am1 = {0};
IPointer p = &av1;
```

struct

- Định nghĩa các struct

```
struct Time{
    int hour;
    int minute;
    int second;
};
struct Date{
   int day, month, year;
};
struct Student{
   char name[32];
   struct Date birthday;
   int id_number;
};
```

Tên kiểu mới không được trùng lặp

C++ có thể lược từ khóa struct

struct

- Khai báo biến:

```
Time classTime = {6,45,0};
Time lunchTime = {12};
Date myBirthday, yourBirthday = {30,4,1975};
Student I = {"Nguyen Van A", {2,9,1975}};
…
```

struct

```
int main(){
    Time classTime = {6,45,0};
    Time lunchTime = {12};
    Date myBirthday, yourBirthday = {30,4,1975};
    Student I = {"Nguyen Van A", {2,9,1975}};
    lunchTime.minute = 15;
    lunchTime.hour = classTime.hour + 6;
    Student U = I; // in C++ also possible: Student U(I);
    U.name[11] = 'B'; // "Nguyen Van B"
    U.id_number++; // 1
    U.birthday.day = 30; // 30-9-1975
    U.birthday.month = 4; // 30-4-1975
    U.birthday = yourBirthday; // structs can be assigned
    return 0;
}
```

struct

- struct được sử dụng để nhóm các dữ liệu liên quan mô tả một đối tượng, các dữ liệu có thể cùng hoặc khác kiểu
- Định nghĩa struct bằng cách khai báo tên các biến thành viên, không được đặt giá trị đầu cho các biến
- Kích cỡ của struct >= tổng kích cỡ các thành viên
- Truy cập một biến struct thông qua tên biến, toán tử (.) và tên biến thành viên
- Các kiểu struct có thể lồng vào nhau, trong struct có thể sử dụng mảng, một mảng có thể có các phần tử là struct, v.v...
- Các biến có cùng kiểu struct có thể gán cho nhau, có thể sử dụng để khởi tạo cho nhau (khác hẳn với mảng)
- Có thể sử dụng con trỏ để truy nhập dữ liệu struct thông qua toán tử (*.) và toán tử (->)
- Hai kiểu struct có khai báo giống nhau hoàn toàn vẫn là hai kiểu struct khác nhau

Cấu trúc điều khiển phân nhánh

- Các kiểu phân nhánh
  - `if .. else`: Phân nhánh lựa chọn một hoặc hai trường hợp
  - `switch .. case`: Phân nhánh lựa chọn nhiều trường hợp
  - `break`: Lệnh nhảy kết thúc (sớm) một phạm vi
  - `return`: Lệnh nhảy và kết thúc (sớm) một hàm
  - `goto`: Lệnh nhảy tới một nhãn (không nên dùng)

if....else

- Lựa chọn một trường hợp: sử dụng if

```
if (npoints >= 60)
   cout << "Passed";
if (npoints >= 80 && npoints <= 90){
   grade = 'A';
   cout << grade;
}
```

- Phân nhánh hai trường hợp: sử dụng if .. else

```
if (npoints >= 90)
   cout << 'A';
else if (npoints >= 80)
   cout << 'B';
   else if (npoints >= 70)
      cout << 'C';
           else if (npoints >= 60)
                    cout << 'D';
           else cout << 'F';
```

Ví dụ: viết hàm trả về số lớn nhất

```
int max1(int a, int b) {
    int c;
    if (a > b)
        c = a;
    else
        c = b;
    return c;
}
```

```
int max2(int a, int b) {
    int c = a;
    if (a < b)
        c = b;
    return c;
}
```

```
int max3(int a, int b) {
    if (a < b)
        a = b;
    return a;
}
```

```
int max4(int a, int b) {
    if (a > b)
        return a;
    else
        return b;
}
```

```
int max5(int a, int b) {
    if (a > b)
        return a;
    return b;
}
```

```
int max6(int a, int b) {
    return (a > b)? a: b;
}
```

switch...case

```
switch(bieu_thuc)
{
    case bieu_thuc_hang:
        statement(s);
        break;      //Tùy chọn
    case bieu_thuc_hang:
        statement(s);
        break;      //Tùy chọn
    //Số lượng case statement không bị giới hạn.
    default: //Tùy chọn
        statement(s);
}
```

switch...case

```
char hocluc = 'B';
switch(hocluc){
    case 'A' :
        cout << "Gioi!" << endl;
        break;
    case 'B' :
    case 'C' :
        cout << "Kha" << endl;
        break;
    case 'D' :
        cout << "Trung binh" << endl;
        break;
    case 'F' :
        cout << "Phai hoc lai!!" << endl;
        break;
    default :
        cout << "Gia tri khong hop le" << endl;
}
```

switch...case

- Cho biết kết quả chương trình sau:

```
int main()
{

    char hocluc = 'A';
    switch(hocluc){
        case 'A' :
            cout << "Gioi!" << endl;
        case 'B' :
        case 'C' :
            cout << "Kha" << endl;
        case 'D' :
            cout << "Trung binh" << endl;
            break;
        case 'F' :
            cout << "Phai hoc lai!!" << endl;
            break;
        default :
            cout << "Gia tri khong hop le" << endl;
    }
}
```

Cấu trúc vòng lặp

- Các kiểu vòng lặp trong C/C++
  - `while (condition) { }`
  - `do { } while (condition)`
  - `for (init;condition;post_action) { }`
- Vòng lặp có thể thực hiện với `if..else + goto` (*không bao giờ nên như vậy*)
- Vòng lặp chủ yếu trong làm việc với mảng và các cấu trúc dữ liệu tổng quát khác => truy nhập qua biến mảng + chỉ số, qua con trỏ hoặc qua `iterator` (sẽ đề cập sau này)
- Có thể sử dụng `break` để dừng vòng lặp ngay lập tức hoặc `continue` để bỏ qua phần còn lại của vòng lặp.

while...

```
#include <iostream>
using namespace std;
int main(){
   int a = 4;
   while(a < 10)
   {
          cout << "Gia tri cua a la: " << a << endl;
          a++;
   }
   return 0;
}
```

for...

```
#include <iostream>
using namespace std;
int main ()
{
   for(int a = 5; a < 15; a = a + 1)
   {
          cout << "Gia tri cua a la: " << a << endl;
   }
   return 0;
}
```

do...while

```
#include <iostream>
using namespace std;
int main(){
   int a = 5;
   do
   {
          cout << "Gia tri cua a la: " << a << endl;
          a = a + 1;
   }while( a < 15 );
   return 0;
}
```

break

```
#include <iostream>
using namespace std;
int main(){
   int a = 10;
   do{
       cout << "Gia tri cua a la: " << a << endl;
       a = a + 1;
       if(a > 15) break;
   }while(a < 20);
   return 0;
}
```

continue

```
#include <iostream>
using namespace std;
int main ()
{
   int sum=0;
   for(int a = 1; a < 15; a = a + 1){
         if (a%2==0) continue;
         sum+=a;
   }
   cout << sum << endl;
   return 0;
}
```

Vòng lặp

- Các cấu trúc vòng lặp while và for tương tự như nhau, thực ra ta chỉ cần một trong hai
- Cấu trúc do..while tuy có ý nghĩa khác một chút, song cũng có thể chuyển về cấu trúc while hoặc for
- Các cấu trúc có thể lồng vào nhau, tuy nhiên tránh lồng quá nhiều để còn dễ bao quát, khi cần có thể phân hoạch lại thành hàm
- Điều khiển vòng lặp có thể nằm trực tiếp trên điều kiện, hoặc có thể kết hợp bên trong vòng lặp với các lệnh if..else và break, return
- Thận trọng trong kiểm tra điều kiện vòng lặp (chỉ số mảng, con trỏ, ...)

Đọc/ghi với file

- C ++ sử dụng các luồng (`streams`) cho đầu vào và đầu ra
- `stream` - là một chuỗi dữ liệu để đọc (`input stream`) hoặc dữ liệu do chương trình tạo ra để xuất (`output stream`)
- Mặc định các luồng chuẩn để đọc và ghi dữ liệu là `cin` (console input) và `cout` (console output)
- File được sử đụng để lưu trữ lâu dài dữ liệu. Để đọc/ghi dữ liệu với file cần khai báo thư viện `fstream` khi đó sử dụng 2 luồng để đọc ghi dữ liệu là `std::ifstream` và `std::ofstream`

```
#include <fstream>
```

Đọc/ghi với file

- Khai báo:

```
ifstream in_stream; // input stream
ofstream out_stream; // output stream
in_stream.open("myfile.dat"); // opening file
```

- Hoặc kết hợp:

```
ifstream instream("myfile.dat");
```

- Các khác: dùng c_str()

```
string filename="myfile.dat";
instream.open(filename.c_str());
```

Đọc/ghi với file

- Sau khi khai báo và mở file, có thể dung các toán tử >> và << để đọc/ghi với file như thông thường. Sau khi thao tác với file xong thì sử dụng phương thức close() để đóng file.

```
ifstream in_stream;
ofstream out_stream;
in_stream.open("songuyen.txt");
in_stream >> n >> a >> b;
out_stream << n << "\n" << a << " " << b;
in_stream.close();
out_stream.close();
```

Đọc/ghi với file

- Hàm eof() trả về true nếu kết thúc file (end of file)

```
while(!in_stream.eof()){ … }
```

- Hàm fail()

```
in_stream.open("myfile.dat"); // opening file
if (in_stream.fail()){
    cout << ”Cannot open file myfile.dat”;
    exit(1);
}
```

freopen()

```
FILE *freopen(const char *filename, const char *mode, FILE *stream)
```

- Tham số
  - filename − Đây là chuỗi chứa tên file để được mở.
  - mode − Đây là chuỗi chứa chế độ truy vập file. Bao gồm:
    - "r" Mở một file để đọc. File phải tồn tại
    - "w" Tạo một file trống để ghi. Nếu file đã tồn tại, nội dung của nó bị ghi đè.
    - "a" Thêm (append) tới một file. Với các hoạt động ghi, phụ thêm dữ liệu tại cuối file. File được tạo nếu nó chưa tồn tại
    - "r+" , "w+“, "a+“: Mở một file để đọc và ghi
- stream − Đây là con trỏ tới một đối tượng FILE mà nhận diện Stream để được mở lại.

freopen()

```
#include <iostream>
#include <stdio.h>
using namespace std;
int main()
{
	freopen("INPUT.TXT", "r", stdin);
	freopen("OUTPUT.TXT", "w", stdout);
	int x;
	cin >> x;
	cout << x;
	return 0;
}
```

Các kiến thức cần nắm

- Phương pháp tiếp cận bài toán và các phương pháp lập trình
  - Top-down & bottup-up
  - Lập trình tuần tự, cấu trúc, hướng đối tượng,....
- Các nguyên tắc cơ bản khi lập trình
  - Đặt tên, KISS, DRY, YAGN
- Lập trình bằng C++ giải quyết các bài toán cơ bản
  - Khai báo biến, hằng, kiểu
  - Mảng, struct
  - Đọc từ file và ghi lên file

Q&A