

TRƯỜNG ĐẠI HỌC
CÔNG NGHỆ THÔNG TIN

TRUYỀN
CƠ SỞ VÀ CAO

11111111

AM VAN KYT

THUẬT LẬP TRÌNH

CƠ SỞ VÀ NÂNG CAO

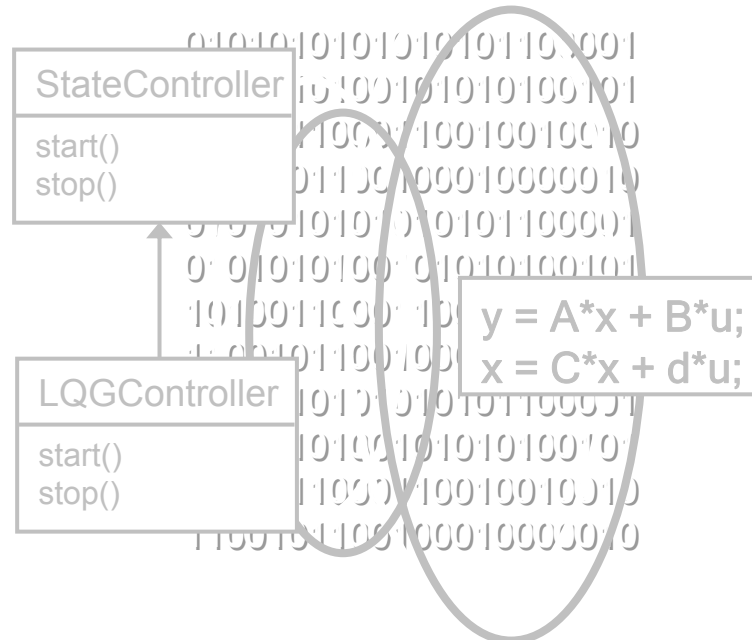


TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

www.beenvn.com

Kỹ thuật lập trình

Chương 1: Mở đầu



Nội dung bài giảng

- 1.1 Giới thiệu nội dung môn học
- 1.2 Giới thiệu chung về kỹ thuật lập trình
- 1.3 Phương pháp luận
- 1.4 Quy trình phát triển phần mềm
- 1.5 Sơ lược về ngôn ngữ C/C++

1.1 Nội dung môn học

- Các **kỹ thuật lập trình cơ bản**, thực hiện minh họa trên các **ngôn ngữ lập trình C và C++**:
 - Lập trình có cấu trúc (*structured programming*)
 - Lập trình hướng đối tượng (*object-oriented programming*)
 - Lập trình thời gian thực (*real-time programming*)
 - Lập trình tổng quát (*generic programming*)
- Tại sao chọn C/C++:
 - Hai ngôn ngữ lập trình tiêu biểu nhất, đủ để thực hiện các kỹ thuật lập trình quan trọng
 - Hai ngôn ngữ lập trình quan trọng nhất đối với kỹ sư điện/kỹ sư điều khiển

Quan điểm về môn học

- Đề cao kiến thức cơ bản, nền tảng:
 - Thiên về **tu duy** và **phương pháp** lập trình
 - Tạo khả năng dễ **thích ứng với các ứng dụng** khác nhau
 - Tạo khả năng dễ **thích ứng với các ngôn ngữ lập trình** khác (Java, Visual Basic, C#, MATLAB...)
 - Nhấn mạnh **tính chuyên nghiệp** trong lập trình: **hiệu quả + chất lượng**
- Những nội dung **không** có trong chương trình:
 - Lập trình hệ thống (low-level system programming)
 - Lập trình đồ họa
 - Lập trình giao tiếp với các thiết bị ngoại vi (cổng nối tiếp, song song...)
 - Lập trình cơ sở dữ liệu
 - Lập trình thành phần, lập trình phân tán (mạng, Internet)

Phương pháp học tập

- Cách thứ nhất: Nghe giảng → làm thử → đọc tài liệu → thảo luận → luyện tập
- Cách thứ hai: Đọc tài liệu → làm thử → nghe giảng → thảo luận → luyện tập
- Nguyên tắc cơ bản: **Chủ động học thường xuyên!**
- Những điều không nên làm:
 - Chép nhiều trên lớp
 - Học thuộc lòng, học chay
 - Mong đợi nhiều vào ôn tập
 - Dựa dẫm vào các bài tập mẫu trong sách

Công cụ học tập

- Máy tính PC
- Môi trường lập trình: **Visual C++ 6.0 (Visual Studio 6.0)**, Visual C++ .NET, Borland C++ Builder
- Nền ứng dụng: **Win32 Console Application**
- Tài liệu tham khảo.

1.2 Tổng quan về kỹ thuật lập trình

- Kỹ thuật lập trình là gì: *Kỹ thuật thực thi một giải pháp phần mềm (cấu trúc 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
 - = Cấu trúc dữ liệu + Giải thuật + Ngôn ngữ lập trình
- Kỹ thuật lập trình
 - ≠ Phương pháp phân tích & thiết kế (A&D)

Thế nào là lập trình?

~~Viết chương trình tính giai thừa của 100!~~

~~Viết chương trình in ra 100 số nguyên tố đầu tiên!~~

~~Lập trình giải bài toán:
"Vừa gà vừa chó,
ba mươi sáu con,
bó lại cho tròn,
một trăm chân chẵn"~~



KHÔNG PHẢI LÀ LẬP TRÌNH!

Viết một hàm tính giai thừa!

Viết chương trình in ra N số nguyên tố đầu tiên!

Lập trình giải bài toán:
"Vừa gà vừa chó,
vừa vạy X con,
bó lại cho tròn,
đủ Y chân chẵn"



ĐÂY LÀ LẬP TRÌNH!



Thế nào là lập trình tốt?

- Đúng/Chính xác
 - Thoả mãn đúng các nhiệm vụ bài toán lập trình đặt ra, được khách hàng chấp nhận
- Ổn định và bền vững
 - Chương trình chạy ổn định trong cả những trường hợp khác nghiệt
 - Chạy ít lỗi (số lượng lỗi ít, cường độ lỗi thấp)
 - Mức độ lỗi nhẹ có thể chấp nhận được
- Khả năng chỉnh sửa
 - Dễ dàng chỉnh sửa trong quá trình sử dụng và phát triển
 - Dễ dàng thay đổi hoặc nâng cấp để thích ứng với điều kiện bài toán lập trình thay đổi
- Khả năng tái sử dụng
 - Có thể được sử dụng hoặc được kế thừa cho các bài toán lập trình khác

Thế nào là lập trình tốt?

- Độ tương thích
 - Khả năng thích ứng và chạy tốt trong các điều kiện môi trường khác nhau
- Hiệu suất
 - Chương trình nhỏ gọn, sử dụng 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 minh họa: Tính giai thừa

- **Viết chương trình** hay **xây dựng hàm**?

- **Hàm tính giai thừa của một số nguyên**

```
int factorial(int N);
```

- **Giải thuật:**

- **Phương pháp đệ quy (*recursive*)**

```
if (N > 1)
    return N*factorial(N-1);
return 1;
```

- **Phương pháp lặp (*iterative*)**

```
int kq = 1;
while (N > 1)
    kq *= N--;
return kq;
```

☺ „to iterate is human,
to recurse is device!“

Làm thế nào để lập trình tốt?

- Học cách tư duy và phương pháp lập trình
 - Tư duy toán học, tư duy logic, tư duy có cấu trúc, tư duy hướng đối tượng, tư duy tổng quát
 - Tìm hiểu về cấu trúc dữ liệu và giải thuật
- Hiểu sâu 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
 - Hiểu sâu được các điểm nêu trên
 - Rèn luyện kỹ năng lập trình
 - Thúc đẩy sáng tạo

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

Nguyên tắc tối cao



*„Keep it simple:
as simple as possible,
but no simpler!“*

(Albert Einstein)

Các bài toán lập trình cho kỹ sư điện

- Lập trình phần mềm điều khiển (μ C, PC, PLC, DCS)
- Lập trình phần mềm thu thập/quản lý dữ liệu quá trình
- Lập trình phần mềm giao diện người-máy (đồ họa)
- Lập trình phần mềm tích hợp hệ thống (COM, OPC,...)
- Lập trình phần mềm tính toán, thiết kế
- Lập trình phần mềm mô phỏng
- Lập trình phần mềm tối ưu hóa
- ...

1.3 Phương pháp luận

- Phương pháp: *Cách thức tiến hành một công việc để có hiệu quả cao*
- Phương pháp luận: *Một tập hợp các phương pháp được sử dụng hoặc bộ môn khoa học nghiên cứu các phương pháp đó*
- Phương pháp luận phục vụ:
 - Phân tích hệ thống
 - Thiết kế hệ thống
 - Thực hiện
 - Thử nghiệm
 - ...

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)
- Ví dụ ngôn ngữ đặc thù:
 - Ngôn ngữ máy,
 - ASSEMBLY
 - BASIC
 - IL (Instruction List), STL (Statement List)
 - LD, LAD (Ladder Diagram)

Lập trình tuần tự: 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: Ưu điểm và nhược điểm

- 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 cấu trúc (structured programming)

- Cấu trúc hóa dữ liệu (xây dựng kiểu dữ liệu) và cấu trúc 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)
- Yêu cầu của chương trình có cấu trúc: chỉ sử dụng các cấu trúc điều khiển tuần tự, tuyển chọn (if then else), lặp (while) và thoát ra (exit).
- Ví dụ các ngôn ngữ đặc thù:
 - PASCAL, ALGO, FORTRAN, C,...
 - SFC (Sequential Function Charts)
 - ST (Structured Text)

Lập trình có cấu trúc: Ví dụ tính giai thừa (PASCAL)

```
FUNCTION Factorial(n: INTEGER) : INTEGER
VAR X: INTERGER;
BEGIN
  X := n;
  WHILE (n > 1) DO
    BEGIN
      DEC(n);
      X := X * n;
    END
    Factorial := X;
  END
END;
```

Lập trình có cấu trúc: Ví dụ 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)

- *Lập trình module là một dạng cải tiến của lập trình có cấu trúc. Chương trình được cấu trúc nghiêm ngặt hơn, dùng đơn vị cấu trúc là module.*
- **Module:**
 - Một đơn vị cấu trúc 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 (Object-Oriented Programming)

- *Xây dựng chương trình ứng dụng theo quan điểm dựa trên các cấu trúc dữ liệu trừu tượng (lớp), các thể nghiệm của các cấu trúc đó (đối tượng) và quan hệ giữa chúng (quan hệ lớp, quan hệ đối tượng).*
- Nguyên lý cơ bản:
 - Trừu tượng (*abstraction*)
 - 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,

Ví dụ minh họa: Quản lý sinh viên (C++)

```
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*);
    ...
};
```

Ví dụ minh họa: Tính toán kiểu MATLAB

```
Vector a(10, 1.0), b(10, 0.5);  
Vector c = a + b;  
...  
Vector d = a - b + 2*c;  
Matrix A(4,4), B(4,2), C(2,4), D(2,2);  
Vector x(4), u(2), y(2);  
...  
while (true) {  
    // đọc đầu vào u  
    y = C*x + D*u;  
    x = A*x + B*u;  
    // đưa đầu ra y  
}  
...  
CTFMatrix G = ss2tf(A,B,C,D);  
...
```

Lập trình tổng quát (generic programming)

- Một tư duy lập trình mở, trên quan điểm tổng quát hóa tất cả những gì có thể nhằm đưa ra một **khuôn mẫu giải pháp** cho nhiều bài toán lập trình cụ thể.
- Ưu điểm:
 - Giảm tối đa lượng mã nguồn
 - Tăng nhiều lần giá trị sử dụng lại của phần mềm
 - Có thể kết hợp tùy ý với các phương pháp luận khác
 - Tính khả chuyển cao
- Các hình thức tổng quát hóa:
 - Kiểu dữ liệu
 - Phép toán cơ bản
 - Cấu trúc dữ liệu
 - Quản lý bộ nhớ,...

Ví dụ minh họa: Các cấu trúc toán học

```
typedef TMatrix<double> Matrix;
typedef TMatrix<complex<double> > ComplexMatrix;
Matrix a(4,4), b(4,4);
Matrix c = a*b;
ComplexMatrix a1(4,4), b1(4,4);
ComplexMatrix c1 = a1*b1;

typedef TPoly<double> Poly;
typedef TMatrix<Poly> PolyMatrix;
typedef TPoly<ComplexMatrix> ComplexMatrixPoly;

TRational<int> IntRational;
TRational<Poly> PolyRational;
...
```

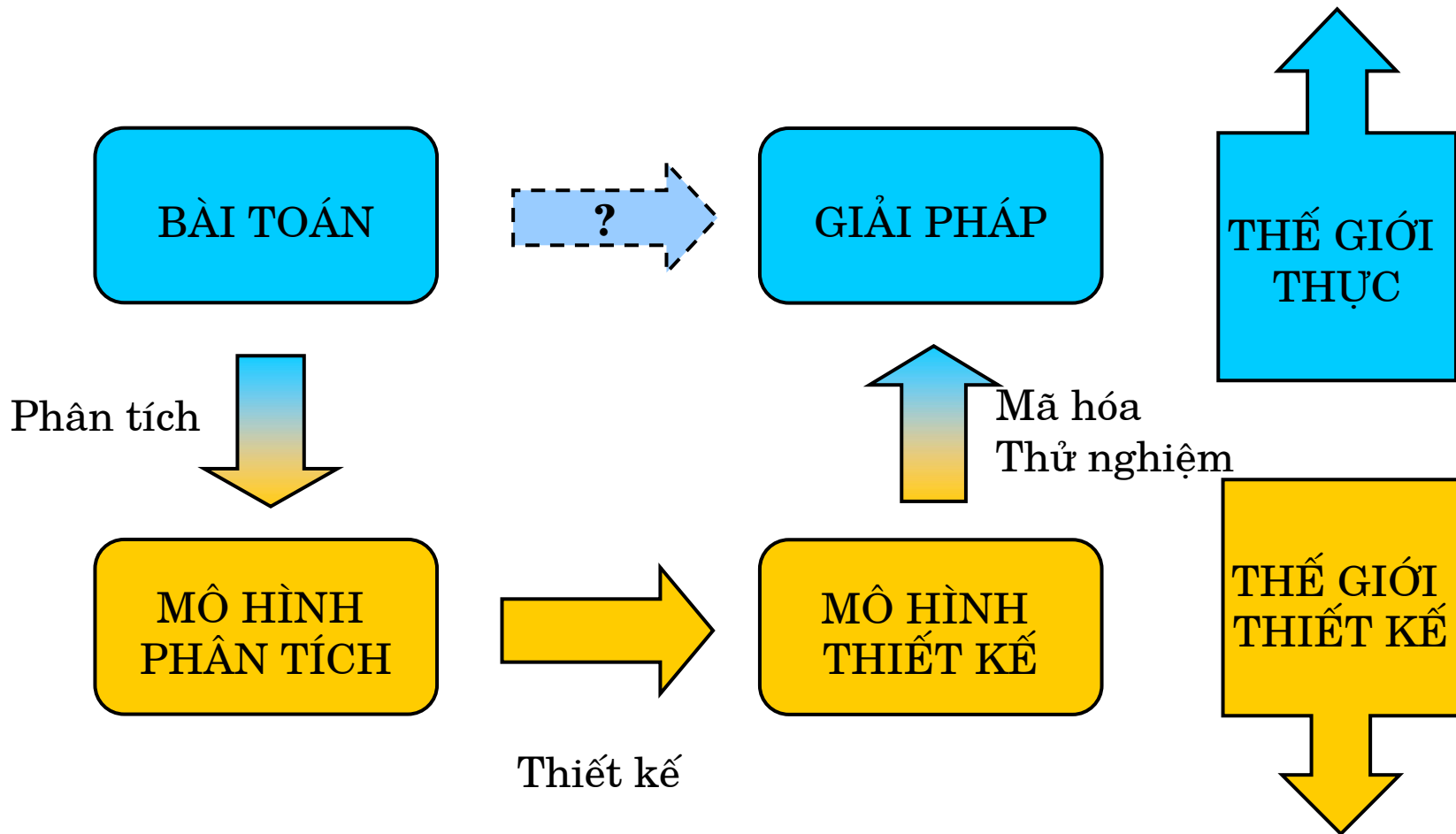
Lập trình thành phần (component-based programming)

- 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, HTML, XML,...
 - FBD

Lập trình thời gian thực (real-time programming)

- 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
 - Hệ điều hành
 - Nền phần cứng
 - Mạng truyền thông

1.4 Quy trình phát triển phần mềm



Tập hợp và phân tích yêu cầu

- Bởi vì: Khách hàng thường biết được là họ muốn gì, nhưng không biết lập hoạch các yêu cầu
 - Cho nên: Cần phải cùng với khách hàng phân hoạch và làm rõ những yêu cầu về phạm vi chức năng của bài toán
 - Kết quả: Mô hình đặc tả (*Specification Model*) ấn định và chỉ rõ yêu cầu của bài toán một cách tường minh theo một ngôn ngữ mô hình hóa rõ ràng, dễ hiểu để nhóm phân tích thiết kế lập trình thực hiện
- ⇒ Trả lời câu hỏi: **Khách hàng cần những gì và nên làm gì?**

Phân tích hệ thống (System analysis)

- Phân tích mối liên hệ của hệ thống với môi trường xung quanh
 - Tìm ra cấu trúc hệ thống và các thành phần quan trọng
 - Định nghĩa chức năng cụ thể của các thành phần
 - Nhận biết các đặc điểm của từng thành phần
 - Phân loại các thành phần, tổng quát hóa, đặc biệt hóa
 - Nhận biết mối liên hệ giữa các thành phần
 - Kết quả: Mô hình hệ thống (*System model*)
 - Cần một ngôn ngữ mô hình hóa để trao đổi giữa các thành viên trong nhóm phân tích và với nhóm thiết kế
- ⇒ Trả lời câu hỏi: **Những gì sẽ phải làm?**

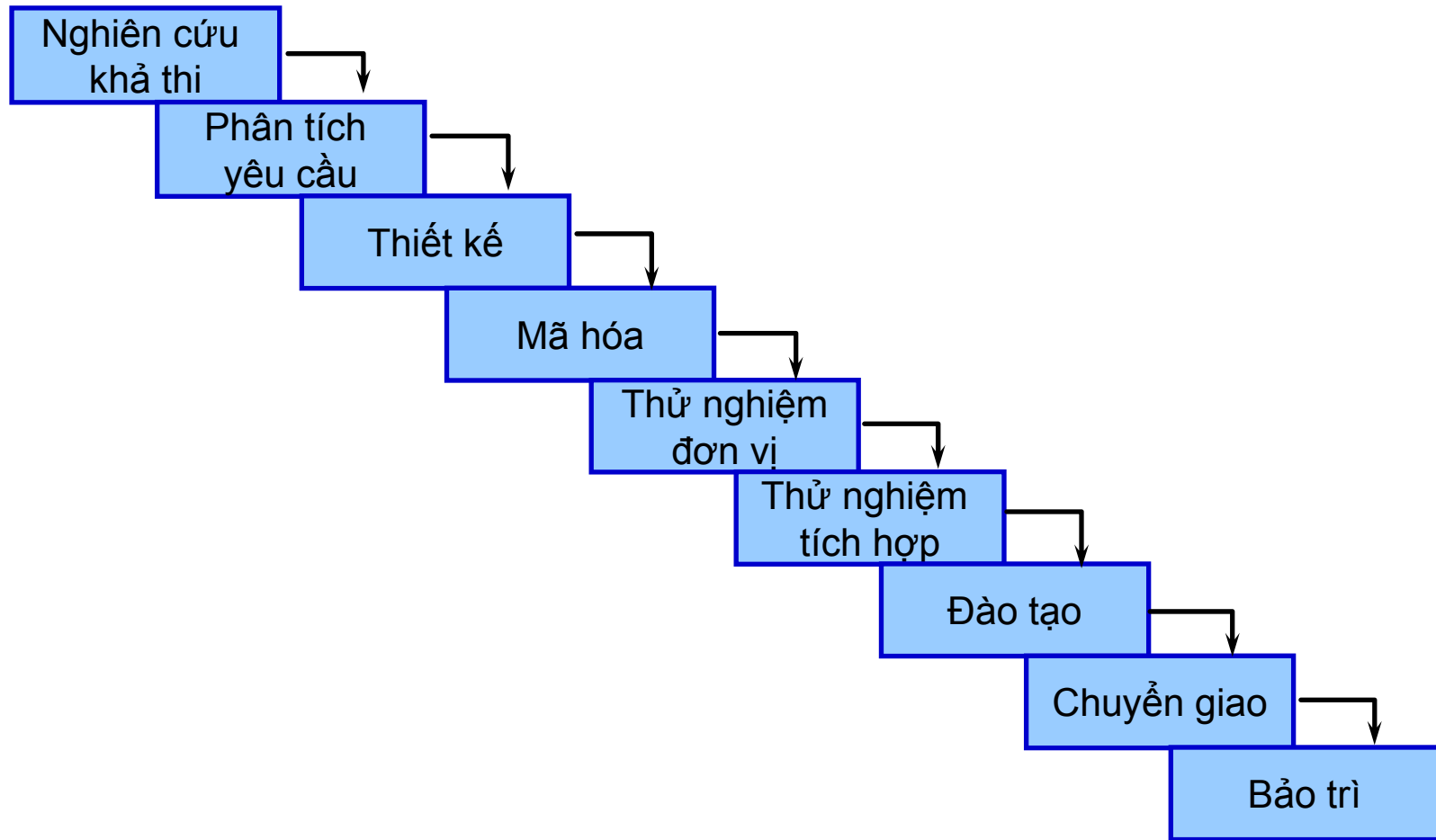
Thiết kế hệ thống (System Design)

- Dựa trên mô hình hệ thống, xây dựng các mô hình chi tiết phục vụ sẵn sàng mã hóa/cài đặt
 - Bao gồm:
 - Thiết kế cấu trúc (*structured design*): chương trình, kiểu dữ liệu, đối tượng, quan hệ cấu trúc giữa các đối tượng và kiểu)
 - Thiết kế tương tác (*interaction design*): quan hệ tương tác giữa các đối tượng
 - Thiết kế hành vi (*behaviour design*): sự kiện, trạng thái, phép toán, phản ứng
 - Thiết kế chức năng (*functional design*): tiến trình hành động, hàm, thủ tục)
 - Kết quả: Mô hình thiết kế (các bản vẽ và lời văn mô tả)
- ⇒ Trả lời câu hỏi: **Làm như thế nào?**

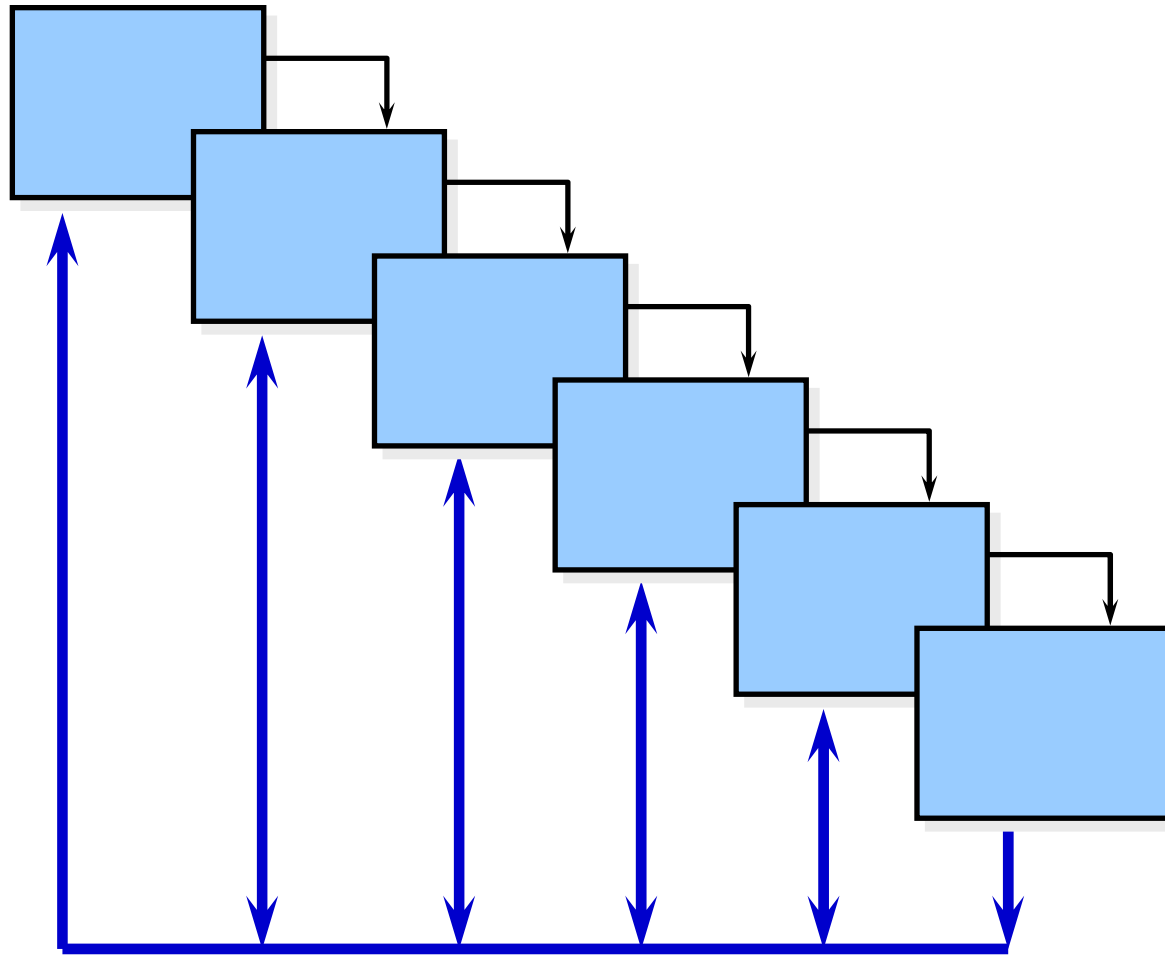
Các bước khác

- Mã hóa/cài đặt (*Coding/Implementation*): Thể hiện mô hình thiết kế với một ngôn ngữ lập trình cụ thể
- Thử nghiệm (*Testing, Verification*): Chạy thử, phân tích và kiểm chứng:
 - Thử đơn vị (*Unit Test*)
 - Thử tích hợp (*Integration Test*)
- Gỡ rối (*Debugging*): Tìm ra và sửa các lỗi chương trình chạy (các lỗi logic)
- Xây dựng tài liệu (*Documenting*): Xây dựng tài liệu phát triển, tài liệu hướng dẫn sử dụng
- Đào tạo, chuyển giao
- Bảo trì, bảo dưỡng

Chu trình cổ điển: “Waterfall Model”



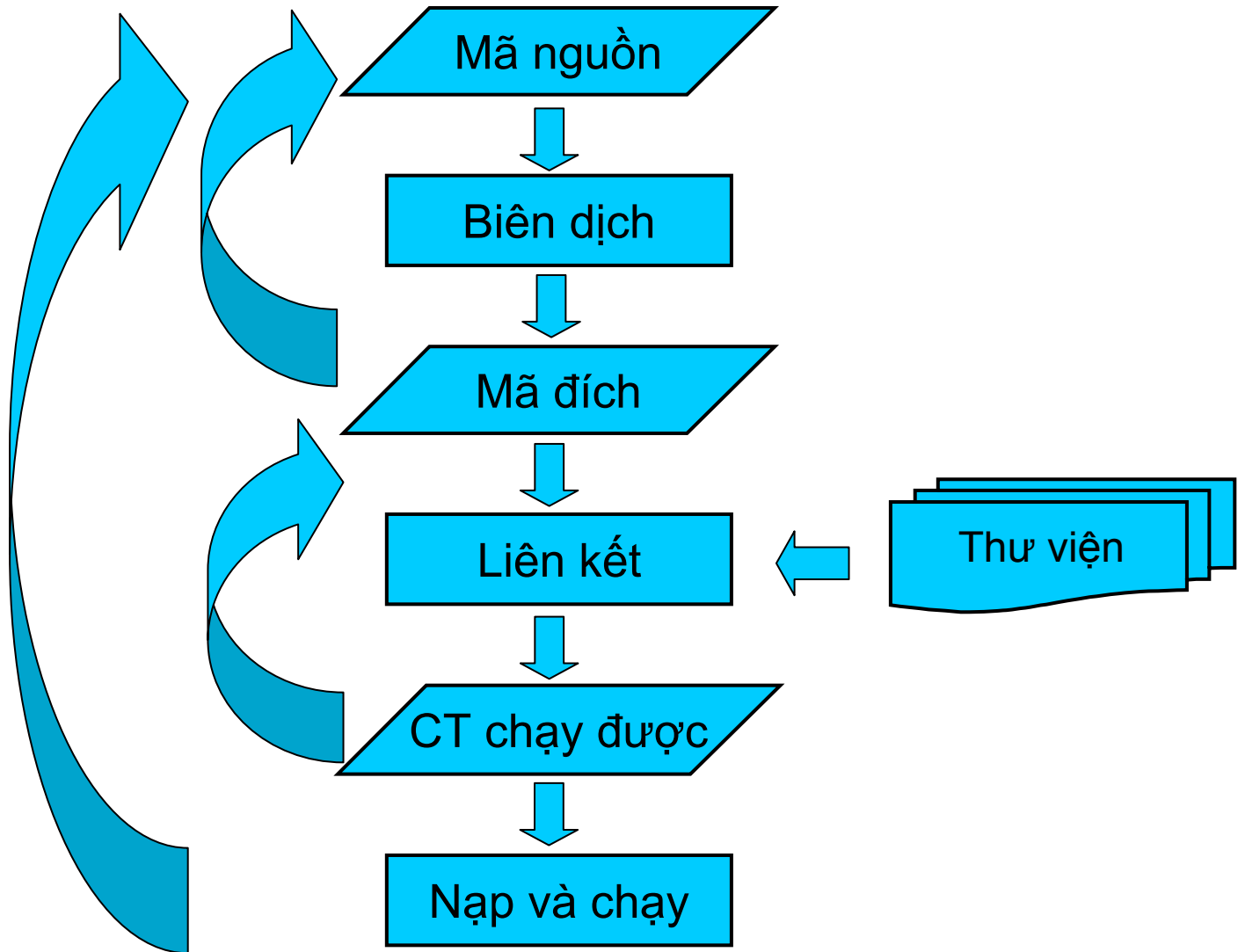
Xu thế hiện nay: Song song và lặp



Lập trình là gì, nằm ở đâu?

- Lập trình > Mã hóa
- Lập trình \approx Tư tưởng thiết kế + Mã hóa + Thử nghiệm + Gỡ rối

Các bước phát triển chương trình



Môi trường/công cụ phát triển

- IDE (*Integrated Development Environment*)
 - Hỗ trợ toàn bộ các bước phát triển chương trình
 - Ví dụ: MS Visual C++, Borland C++ (Builder), Keil-C
- Các công cụ tiêu biểu
 - 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)

Workspace 'ConnectDots': 1 project(

- ConnectDots files
 - ApiMain.cpp
 - Coordinate.cpp
 - Coordinate.h
 - EdgeStatus.h
 - Game.cpp
 - Game.h
 - Mover.h
 - position.h
 - Square.cpp
 - Square.h
 - Ezwinvc50.lib
 - External Dependencies

```
#include "Game.h"

using namespace std;

Game game;

int ApiMain() {
    game.start();
    return 0;
}

int processClick(const Position & p) {
    game.makePlayerMove(p);
    return 0;
}
```

1.5 Sơ lược về C/C++

Lược sử ngôn ngữ C

- Tiến hóa từ hai ngôn ngữ lập trình
 - BCPL và B: Các ngôn ngữ “phi kiểu”
- Dennis Ritchie (Bell Laboratories, AT&T)
 - Bổ sung kiểu hóa dữ liệu và các yếu tố khác
- Ngôn ngữ phát triển hệ điều hành UNIX
- Không phụ thuộc phần cứng
 - Tính khả chuyển
- 1989: ANSI chuẩn hóa (ANSI-C)
- 1990: Công bố chuẩn ANSI và ISO
 - ANSI/ISO 9899: 1990

Lược sử ngôn ngữ C++

- Mở rộng, tiến hóa từ C
- Bjarne Stroustrup (Bell Laboratories)
 - Đầu những năm 1980: “C with classes”
 - 1984: Tên C++
 - 1987: “The C++ Programming Language” 1st Edition
 - 1997: “The C++ Programming Language” 3rd Edition
 - Chuẩn hóa quốc tế: ANSI/ISO 1996
- Bổ sung các đặc tính hỗ trợ:
 - Lập trình hướng đối tượng
 - Lập trình tổng quát
 - Lập trình toán học,...
- Ngôn ngữ “lai”

Tại sao chọn C/C++

- Đáp ứng các yêu cầu:
 - Gần gũi với phần cứng
 - Hiệu suất cao
 - Tương đối thân thiện với người lập trình
 - Khả chuyển
 - Chuẩn hóa quốc tế (tương lai vững chắc)
- Thế mạnh tuyệt đối của ANSI-C:
 - Phổ biến cho hầu hết các nền vi xử lý, vi điều khiển, DSP
 - Phổ biến cho “mỗi người lập trình” trên thế giới
- Thế mạnh tuyệt đối của ANSI/ISO C++:
 - Lập trình hướng đối tượng
 - Lập trình tổng quát (template)
 - Lập trình toán học (dữ liệu trừu tượng và nạp chồng toán tử)

Visual C++, .NET & C#

- Visual C++:
 - Môi trường/công cụ lập trình C++ của Microsoft
 - Mở rộng một số yếu tố
 - Thư viện lập trình Windows: Microsoft Foundation Classes (MFC), Active Template Library (ATL)
 - Các thư viện chung: GUI, graphics, networking, multithreading, ...
- .NET (“dot net”)
 - Kiến trúc nền tảng phần mềm lập trình phân tán
 - Hướng tới các ứng dụng Web, phân tán trên nhiều chủng loại thiết bị khác nhau
 - Các ứng dụng trên nhiều ngôn ngữ khác nhau có thể giao tiếp một cách đơn giản trên một nền chung
 - Phương pháp luận: Lập trình thành phần

Visual C++, .NET & C#

- C#
 - Anders Hejlsberg và Scott Wiltamuth (Microsoft)
 - Thiết kế riêng cho nền .NET
 - Nguồn gốc từ C, C++ và Java
 - Điều khiển theo sự kiện, hoàn toàn hướng đối tượng, ngôn ngữ lập trình hiển thị
 - Integrated Development Environment (IDE)
 - Tương tác giữa các ngôn ngữ

Chúng ta đã học được những gì?



- Biết được những gì sẽ phải học, học để làm gì và phải học như thế nào
- Hàng loạt khái niệm mới xung quanh kỹ thuật lập trình và qui trình công nghệ phần mềm
- Tổng quan về các kỹ thuật lập trình
- Lược sử ngôn ngữ C/C++, thế mạnh của chúng so với các ngôn ngữ khác

Chủ đề tiếp theo: C/C++ cơ sở

- Tổ chức chương trình/bộ nhớ
- Dữ liệu và biến
- Toán tử, biểu thức và câu lệnh
- Điều khiển chương trình: vòng lặp, rẽ nhánh
- Mảng và con trỏ
- Cấu trúc

Kỹ thuật lập trình

Phần III: Lập trình hướng đối tượng

Chương 5: Lớp và đối tượng



Nội dung chương 5



- 5.1 Khái niệm
- 5.2 Từ cấu trúc sang lớp
- 5.3 Biến thành viên
- 5.4 Hàm thành viên
- 5.5 Kiểm soát truy nhập

5.1 Khái niệm

Đối tượng là gì?

- Thực thể phần mềm
- Mô hình/đại diện của một đối tượng vật lý:
 - Tank, Heater, Furnace
 - Motor, Pump, Valve
 - Sensor, Thermometer, Flowmeter
 - Control Loop, Control System
- Hoặc một đối tượng logic ("conceptual object):
 - Trend, Report, Button, Window
 - Matrix, Vector, Polynomial

Một đối tượng có...

- ➔ Các thuộc tính (attributes)
- ➔ Trạng thái (state)
 - Dữ liệu
 - Quan hệ
- ➔ Hành vi (behavior)
 - Các phép toán
 - Đặc tính phản ứng
- ➔ Căn cước (identity)
- ➔ Ngữ nghĩa/trách nhiệm (semantic/responsibilities)



Lớp là gì?

- Một lớp là thực thi của các đối tượng có chung
 - Ngữ nghĩa
 - Thuộc tính
 - Quan hệ
 - Hành vi
- Lớp = Đóng gói [Cấu trúc dữ liệu + hàm thao tác]
 - Lớp các vector, lớp các ma trận (dữ liệu phần tử + các phép truy nhập và phép toán cơ bản)
 - Lớp các hình chữ nhật (các dữ liệu tọa độ + phép vẽ, xóa,...)
 - Lớp các mô hình hàm truyền (các hệ số đa thức tử/mẫu, các phép toán xác định tính ổn định, xác định các điểm cực,...)
- Các dữ liệu của một lớp => biến thành viên
- Các hàm của một lớp => hàm thành viên
- Các biến của một lớp => một đối tượng, một thể nghiệm

Lập trình hướng đối tượng (object-oriented programming, OOP)

- Trừu tượng hóa (*abstraction*): giúp đơn giản hóa vấn đề, dễ sử dụng lại
- Đóng gói dữ liệu/che dấu thông tin (*data encapsulation/information hiding*): nâng cao giá trị sử dụng lại và độ tin cậy của phần mềm
- Dẫn xuất/thừa kế (*subtyping/inheritance*): giúp dễ sử dụng lại mã phần mềm và thiết kế
- Đa hình/đa xạ (*polymorphism*): giúp phản ánh trung thực thế giới thực và nâng cao tính linh hoạt của phần mềm

Phương pháp luận hướng đối tượng cho phép tư duy ở mức trừu tượng cao nhưng gần với thế giới thực!

5.2 Từ cấu trúc sang lớp

```
struct Time {
    int hour; // gio
    int min;  // phut
    int sec;  // giay
};

void addHour(Time& t, int h) {
    t.hour += h;
}

void addMin(Time& t, int m) {
    t.min += m;
    if (t.min > 59) {
        t.hour += t.min/60;
        t.min %= 60;
    }
    else if (t.min < 0) {
        t.hour += (t.min/60 - 1);
        t.min = (t.min % 60) + 60;
    }
}
```

```
void addSec(Time& t, int s) {
    t.sec += s;
    if (t.sec > 59) {
        addMin(t, t.sec/60);
        t.sec %= 60;
    }
    else if (t.sec < 0) {
        addMin(t, t.sec/60 - 1);
        t.sec = (t.sec % 60) + 60;
    }
}

void main() {
    Time t = {1, 0, 0};
    addMin(t, 60);
    addMin(t, -5);
    addSec(t, 25);
    ...
}
```


Một số vấn đề của cấu trúc

- Truy nhập dữ liệu trực tiếp, không có kiểm soát có thể dẫn đến không an toàn

```
Time t1 = {1, 61, -3}; // ??!  
Time t2; // Uncertain values  
int h = t2.hour; // ??!  
int m = 50;  
t2.min = m + 15; // ??!
```

- Không phân biệt giữa “chi tiết bên trong” và “giao diện bên ngoài”, một thay đổi nhỏ ở chi tiết bên trong cũng bắt người sử dụng phải thay đổi mã sử dụng theo!

Ví dụ: cấu trúc Time được sửa lại tên biến thành viên:

```
struct Time {  
    int h, m, s;  
};
```

Đoạn mã cũ sẽ không biên dịch được:

```
Time t;  
t.hour = 5;
```

Đóng gói hay "lớp hóa"

```
class Time {  
    int hour; // gio  
    int min;  // phut  
    int sec;  // giay  
public:  
    Time() {hour=min=sec=0;}  
  
    void setTime(int h, int m, int s)  
    {  
        hour = h;  
        min = sec = 0;  
        addSec(s);  
        addMin(m);  
    }  
  
    int getHour() { return hour; }  
    int getMin()  { return min; }  
    int getSec()  { return sec; }  
  
    void addHour(int h) { hour += h; }  
    ...  
}
```

Biến thành viên
(member variable)

Hàm tạo (constructor)

Hàm thành viên
(member functions)

```

void addMin(int m) {
    min += m;
    if (min > 59) {
        hour += min/60;
        min %= 60;
    }
    else if (min < 0) {
        hour += (min/60 - 1);
        min = (min % 60) + 60;
    }
}

void addSec(int s) {
    sec += s;
    if (sec > 59) {
        addMin(sec/60);
        sec %= 60;
    }
    else if (sec < 0) {
        addMin(sec/60 - 1);
        sec = (sec % 60) + 60;
    }
}
};

```

```

void main()
{
    Time t;
    t.addHour(1);
    t.addMin(60);
    t.addMin(-5);
    t.addSec(25);
    t.hour = 1; // error
    t.min = 65; // error
    t.sec = -3; // error
    t.setTime(1, 65, -3);
    int h = t.getHour();
    int m = t.getMin();
    int s = t.getSec();
}

```

5.3 Biến thành viên

- Khai báo biến thành viên của một lớp tương tự như cấu trúc

```
class Time {  
    int hour, min, sec;  
    ...  
};
```

- Mặc định, các biến thành viên của một lớp không truy nhập được từ bên ngoài (biến riêng), đương nhiên cũng không khởi tạo được theo cách cổ điển:

```
Time t = {1, 0, 0};    // error!  
t.hour = 2;           // error!
```

- Có thể làm cho một biến thành viên truy nhập được từ bên ngoài (biến công cộng), tuy nhiên ít khi có lý do cần làm như thế:

```
class Point {  
    public:  
    int x,y;  
};
```

- Kiểm soát việc truy nhập các biến riêng thông qua các hàm thành viên
- Cách duy nhất để khởi tạo giá trị cho các biến thành viên là sử dụng hàm tạo:

```
class Time {  
    ...  
public:  
    Time() {hour=min=sec=0;}  
};  
Time t;    // t.hour = t.min = t.sec = 0;
```

- Một số biến thành viên có vai trò lưu trữ trạng thái bên trong của đối tượng, không nên cho truy nhập từ bên ngoài (ngay cả gián tiếp qua các hàm)

```
class PID {  
    double Kp, Ti, Td;    // controller parameters  
    double I;           // internal state  
    ...  
};
```

5.4 Hàm thành viên

Định nghĩa cấu trúc & hàm

```
struct Time {  
    int hour, min, sec  
};  
void addHour(Time& t, int h) {  
    t.hour += h;  
}  
...
```

Gọi hàm với biến cấu trúc

```
Time t;  
...  
addHour(t, 5);
```

Định nghĩa lớp

```
class Time {  
    int hour, min, sec;  
public:  
    void addHour(int h) {  
        hour += h;  
    }  
    ...  
};
```

Gọi hàm thành viên của DT

```
Time t;  
...  
t.addHour(5);
```

Ở đây có sự khác nhau về cách viết, nhưng chưa có sự khác nhau cơ bản

Khai báo và định nghĩa hàm thành viên

- Thông thường, lớp cùng các hàm thành viên được **khai báo** trong tệp tin đầu (*.h). Ví dụ trong tệp có tên “mytime.h”:

```
class Time {  
    int hour, min, sec;  
public:  
    void addHour(int h);  
    void addMin(int m);  
    void addSec(int s);  
    ...  
};
```

- Các hàm thường được **định nghĩa** trong tệp tin nguồn (*.cpp):

```
#include "mytime.h"  
...  
void Time::addHour(int h) {  
    hour += h;  
}
```

- Có thể định nghĩa một hàm thành viên trong tệp tin đầu dưới dạng một hàm inline (chỉ nên áp dụng với hàm đơn giản), ví dụ:

```
inline void Time::addHour(int h) { hour += h; }
```

- Một hàm thành viên cũng có thể được định nghĩa trong phần khai báo lớp => mặc định trở thành hàm inline, ví dụ

```
class Time {  
    int hour, min, sec;  
public:  
    void addHour(int h) { hour += h; }  
};
```

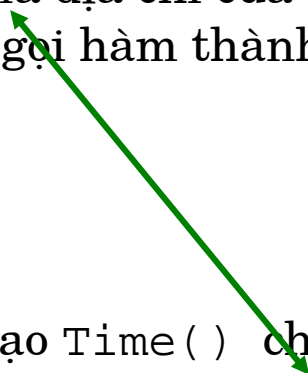
- Khi định nghĩa hàm thành viên, có thể sử dụng các biến thành viên và gọi hàm thành viên khác mà **không cần** (thậm chí không thể được) đưa tên biến đối tượng, ví dụ:

```
void Time::addSec(int s) {  
    ...  
    addMin(sec/60);  
    ...  
}
```


Bản chất của hàm thành viên?

```
class Time {
    int hour,min,sec;
public:
    Time() { hour=min=sec=0; }
    void addHour(int h) {
        this->hour += h; // con trỏ this chính là địa chỉ của
    } // đối tượng gọi hàm thành viên
    ...
};

void main() {
    Time t1,t2; // Tự động gọi hàm tạo Time() cho t1 và t2
    t1.addHour(5); // Có thể hiểu như là addHour(&t1,5);
    t2 = t1; // OK
    t2.addHour(5); // Có thể hiểu như là addHour(&t2,5);
    ...
}
```



5.5 Kiểm soát truy nhập

- **public:** Các thành viên công cộng, có thể sử dụng được từ bên ngoài
- **private:** Các thành viên riêng, không thể truy nhập được từ bên ngoài, ngay cả trong lớp dẫn xuất (sẽ đề cập sau)

```
class Time {  
    private:  
        int hour, min, sec;  
        ...  
};
```
- Mặc định, khi đã khai báo **class** thì các thành viên là **private**.
- **protected:** Các thành viên được bảo vệ, không thể truy nhập được từ bên ngoài, nhưng truy nhập được các lớp dẫn xuất (sẽ đề cập sau)

5.6 Con trỏ đối tượng

```
#include "mytime.h"
void main() {
    Time t;           // call constructor Time()
    t.addHour(5);
    Time *pt = &t;   // pt is identical to this pointer
    pt->addSec(70);
    pt = new Time;   // call constructor Time()
    pt->addMin(25);
    ...
    delete pt;
    pt = new Time[5]; // call constructor 5 times
    for (int i=0; i < 5; ++ i)
        pt[i].addSec(10);
    ...
    delete [] pt;
}
```

Bài tập về nhà

- Dựa trên cấu trúc Vector và các hàm liên quan đã thực hiện trong chương 4, hãy xây dựng lớp đối tượng Vector với các hàm thành viên cần thiết.
- Khai báo một lớp thực hiện lưu trữ thông tin của một lớp sinh viên gồm những thành phần thuộc tính như sau:
 - Số hiệu sinh viên : Kiểu số nguyên
 - Họ và tên: Chuỗi ký tự
 - Năm sinh: Kiểu số nguyên
- Khai báo và định nghĩa mở rộng lớp quản lý sinh viên bằng các hàm thành viên thực hiện các chức năng như sau:
 - Nhập họ tên sinh viên
 - Nhập số hiệu sinh viên
 - Nhập năm sinh
 - Tìm và hiển thị thông tin sinh viên khi biết mã số

Kỹ thuật lập trình

Chương 4: Khái quát về cấu trúc dữ liệu



Nội dung chương 4

- 4.1 Cấu trúc dữ liệu là gì?
- 4.2 Mảng và quản lý bộ nhớ động
- 4.2 Xây dựng cấu trúc Vector
- 4.3 Xây dựng cấu trúc List

4.1 Giới thiệu chung

- Phần lớn các bài toán trong thực tế liên quan tới các dữ liệu phức hợp, những kiểu dữ liệu cơ bản trong ngôn ngữ lập trình không đủ biểu diễn
- Ví dụ:
 - Dữ liệu sinh viên: Họ tên, ngày sinh, quê quán, mã số SV,...
 - Mô hình hàm truyền: Đa thức tử số, đa thức mẫu số
 - Mô hình trạng thái: Các ma trận A, B, C, D
 - Dữ liệu quá trình: Tên đại lượng, dải đo, giá trị, đơn vị, thời gian, cấp sai số, ngưỡng giá trị,...
 - Đối tượng đồ họa: Kích thước, màu sắc, đường nét, phong chữ, ...
- Phương pháp biểu diễn dữ liệu: định nghĩa kiểu dữ liệu mới sử dụng cấu trúc (struct, class, union, ...)

Vấn đề: Biểu diễn tập hợp dữ liệu

- Đa số những dữ liệu thuộc một ứng dụng có liên quan với nhau => cần biểu diễn trong một tập hợp có cấu trúc, ví dụ:
 - Danh sách sinh viên: Các dữ liệu sinh viên được sắp xếp theo thứ tự Alphabet
 - Mô hình tổng thể cho hệ thống điều khiển: Bao gồm nhiều thành phần tương tác
 - Dữ liệu quá trình: Một tập dữ liệu có thể mang giá trị của một đại lượng vào các thời điểm gián đoạn, các dữ liệu đầu vào liên quan tới dữ liệu đầu ra
 - Đối tượng đồ họa: Một cửa sổ bao gồm nhiều đối tượng đồ họa, một bản vẽ cũng bao gồm nhiều đối tượng đồ họa
- Thông thường, các dữ liệu trong một tập hợp có cùng kiểu, hoặc ít ra là tương thích kiểu với nhau
- Kiểu mảng không phải bao giờ cũng phù hợp!

Vấn đề: Quản lý (tập hợp) dữ liệu

- Sử dụng kết hợp một cách khéo léo kiểu cấu trúc và kiểu mảng đủ để biểu diễn các tập hợp dữ liệu bất kỳ
- Các giải thuật (hàm) thao tác với dữ liệu, nhằm quản lý dữ liệu một cách hiệu quả:
 - Bổ sung một mục dữ liệu mới vào một danh sách, một bảng, một tập hợp, ...
 - Xóa một mục dữ liệu trong một danh sách, bảng, tập hợp, ..
 - Tìm một mục dữ liệu trong một danh sách, bảng tập hợp, ... theo một tiêu chuẩn cụ thể
 - Sắp xếp một danh sách theo một tiêu chuẩn nào đó
 -

Quản lý DL thế nào là hiệu quả?

- Tiết kiệm bộ nhớ: Phần "overhead" không đáng kể so với phần dữ liệu thực
- Truy nhập nhanh, thuận tiện: Thời gian cần cho bổ sung, tìm kiếm và xóa bỏ các mục dữ liệu phải ngắn
- Linh hoạt: Số lượng các mục dữ liệu không (hoặc ít) bị hạn chế cố định, không cần biết trước khi tạo cấu trúc, phù hợp với cả bài toán nhỏ và lớn
- Hiệu quả quản lý dữ liệu phụ thuộc vào
 - Cấu trúc dữ liệu được sử dụng
 - Giải thuật được áp dụng cho bổ sung, tìm kiếm, sắp xếp, xóa bỏ

Các cấu trúc dữ liệu thông dụng

- Mảng (nghĩa rộng): Tập hợp các dữ liệu có thể truy nhập tùy ý theo chỉ số
- Danh sách (list): Tập hợp các dữ liệu được móc nối đôi một với nhau và có thể truy nhập tuần tự
- Cây (tree): Tập hợp các dữ liệu được móc nối với nhau theo cấu trúc cây, có thể truy nhập tuần tự từ gốc
 - Nếu mỗi nút có tối đa hai nhánh: cây nhị phân (binary tree)
- Bìa, bảng (map): Tập hợp các dữ liệu có sắp xếp, có thể truy nhập rất nhanh theo mã khóa (key)
- Hàng đợi (queue): Tập hợp các dữ liệu có sắp xếp tuần tự, chỉ bổ sung vào từ một đầu và lấy ra từ đầu còn lại

Các cấu trúc dữ liệu thông dụng (tiếp)

- Tập hợp (set): Tập hợp các dữ liệu được sắp xếp tùy ý nhưng có thể truy nhập một cách hiệu quả
- Ngăn xếp (stack): Tập hợp các dữ liệu được sắp xếp tuần tự, chỉ truy nhập được từ một đầu
- Bảng hash (hash table): Tập hợp các dữ liệu được sắp xếp dựa theo một mã số nguyên tạo ra từ một hàm đặc biệt
- Bộ nhớ vòng (ring buffer): Tương tự như hàng đợi, nhưng dung lượng có hạn, nếu hết chỗ sẽ được ghi quay vòng
- Trong toán học và trong điều khiển: vector, ma trận, đa thức, phân thức, hàm truyền, ...

4.2 Mảng và quản lý bộ nhớ động

- Mảng cho phép biểu diễn và quản lý dữ liệu một cách khá hiệu quả:
 - Đọc và ghi dữ liệu rất nhanh qua chỉ số hoặc qua địa chỉ
 - Tiết kiệm bộ nhớ
- Các vấn đề của mảng tĩnh:
 - VD: `Student student_list[100];`
 - Số phần tử phải là hằng số (biết trước khi biên dịch, người sử dụng không thể nhập số phần tử, không thể cho số phần tử là một biến) => kém linh hoạt
 - Chiếm chỗ cứng trong ngăn xếp (đối với biến cục bộ) hoặc trong bộ nhớ dữ liệu chương trình (đối với biến toàn cục) => sử dụng bộ nhớ kém hiệu quả, kém linh hoạt

Mảng động

- Mảng động là một mảng được cấp phát bộ nhớ theo yêu cầu, trong khi chương trình chạy

```
#include <stdlib.h>      /* C */
int n = 50;
...
float* p1= (float*) malloc(n*sizeof(float)); /* C */
double* p2= new double[n];    // C++
```

- Sử dụng con trỏ để quản lý mảng động: Cách sử dụng không khác so với mảng tĩnh

```
p1[0] = 1.0f;
p2[0] = 2.0;
```

- Sau khi sử dụng xong => giải phóng bộ nhớ:

```
free(p1); /* C */
delete [] p2;    // C++
```

Cấp phát và giải phóng bộ nhớ động

- C:
 - Hàm `malloc()` yêu cầu tham số là **số byte**, trả về con trỏ không kiểu (`void*`) mang địa chỉ vùng nhớ mới được cấp phát (nằm trong heap), trả về 0 nếu không thành công.
 - Hàm `free()` yêu cầu tham số là con trỏ không kiểu (`void*`), giải phóng vùng nhớ có địa chỉ đưa vào
- C++:
 - Toán tử `new` chấp nhận kiểu dữ liệu phần tử kèm theo số lượng phần tử của mảng cần cấp phát bộ nhớ (trong vùng heap), trả về con trỏ có kiểu, trả về 0 nếu không thành công.
 - Toán tử `delete[]` yêu cầu tham số là con trỏ có kiểu.
 - Toán tử `new` và `delete` còn có thể áp dụng cho cấp phát và giải phóng bộ nhớ cho một biến đơn, một đối tượng chứ không nhất thiết phải một mảng.

Một số điều cần lưu ý

- Con trỏ có vai trò quản lý mảng (động), chứ con trỏ không phải là mảng (động)
- Cấp phát bộ nhớ và giải phóng bộ nhớ chứ không phải cấp phát con trỏ và giải phóng con trỏ
- Chỉ giải phóng bộ nhớ một lần

```
int* p;  
p[0] = 1;           // never do it  
new(p);            // access violation!  
p = new int[100];  // OK  
p[0] = 1;         // OK  
int* p2=p;        // OK  
delete[] p2;      // OK  
p[0] = 1;         // access violation!  
delete[] p;       // very bad!  
p = new int[50];  // OK, new array  
...
```


Cấp phát bộ nhớ động cho biến đơn

- **Ý nghĩa:** Các đối tượng có thể được tạo ra động, trong khi chương trình chạy (bổ sung sinh viên vào danh sách, vẽ thêm một hình trong bản vẽ, bổ sung một khâu trong hệ thống,...)

- Cú pháp

```
int* p = new int;  
*p = 1;  
p[0]= 2;           // the same as above  
p[1]= 1;           // access violation!  
int* p2 = new int(1); // with initialization  
delete p;  
delete p2;  
Student* ps = new Student;  
ps->code = 1000;  
...  
delete ps;
```

- Một biến đơn **khác** với mảng một phần tử!

Ý nghĩa của sử dụng bộ nhớ động

- Hiệu suất:
 - Bộ nhớ được cấp phát đủ dung lượng theo yêu cầu và khi được yêu cầu trong khi chương trình đã chạy
 - Bộ nhớ được cấp phát nằm trong vùng nhớ tự do còn lại của máy tính (heap), chỉ phụ thuộc vào dung lượng bộ nhớ của máy tính
 - Bộ nhớ có thể được giải phóng khi không sử dụng tiếp.
- Linh hoạt:
 - Thời gian "sống" của bộ nhớ được cấp phát động có thể kéo dài hơn thời gian "sống" của thực thể cấp phát nó.
 - Có thể một hàm gọi lệnh cấp phát bộ nhớ, nhưng một hàm khác giải phóng bộ nhớ.
 - Sự linh hoạt cũng dễ dẫn đến những lỗi "rò rỉ bộ nhớ".

Ví dụ sử dụng bộ nhớ động trong hàm

```
Date* createDateList(int n) {
    Date* p = new Date[n];
    return p;
}

void main() {
    int n;
    cout << "Enter the number of your national holidays:";
    cin >> n;
    Date* date_list = createDateList(n);
    for (int i=0; i < n; ++i) {
        ...
    }
    for (... ) { cout << ... }
    delete [] date_list;
}
```

Tham số đầu ra là con trỏ?

```
void createDateList(int n, Date* p) {
    p = new Date[n];
}

void main() {
    int n;
    cout << "Enter the number of your national holidays:";
    cin >> n;
    Date* date_list;
    createDateList(n, date_list);
    for (int i=0; i < n; ++i) {
        ...
    }
    for (... ) { cout << ... }
    delete [] date_list;
}
```

4.3 Xây dựng cấu trúc Vector

- Vấn đề: Biểu diễn một vector toán học trong C/C++?
- Giải pháp chân phương: mảng động thông thường, nhưng...
 - Sử dụng không thuận tiện: Người sử dụng tự gọi các lệnh cấp phát và giải phóng bộ nhớ, trong các hàm luôn phải đưa tham số là số chiều.
 - Sử dụng không an toàn: Nhầm lẫn nhỏ dẫn đến hậu quả nghiêm trọng

```
int n = 10;
double *v1,*v2, d;
v1 = (double*) malloc(n*sizeof(double));
v2 = (double*) malloc(n*sizeof(double));
d = scalarProd(v1,v2,n); // scalar_prod đã có
d = v1 * v2;           // OOPS!
v1.data[10] = 0;      // OOPS!
free(v1);
free(v2);
```

Định nghĩa cấu trúc Vector

- Tên file: `vector.h`

- Cấu trúc dữ liệu:

```
struct Vector {  
    double *data;  
    int     nelem;  
};
```

- Khai báo các hàm cơ bản:

```
Vector createVector(int n, double init);  
void    destroyVector(Vector);  
double  getElem(Vector, int i);  
void    putElem(Vector, int i, double d);  
Vector  addVector(Vector, Vector);  
Vector  subVector(Vector, Vector);  
double  scalarProd(Vector, Vector);  
...
```

Định nghĩa các hàm cơ bản

- **Tên file:** vector.cpp

```
#include <stdlib.h>
#include "vector.h"
Vector createVector(int n, double init) {
    Vector v;
    v.nelem = n;
    v.data = (double*) malloc(n*sizeof(double));
    while (n--) v.data[n] = init;
    return v;
}
void destroyVector(Vector v) {
    free(v.data);
}
double getElem(Vector v, int i) {
    if (i < v.nelem && i >= 0) return v.data[i];
    return 0;
}
```

```

void putElem(Vector v, int i, double d) {
    if (i >=0 && i < v.nelem) v.data[i] = d;
}

Vector addVector(Vector a, Vector b) {
    Vector c = {0,0};
    if (a.nelem == b.nelem) {
        c = createVector(a.nelem,0.0);
        for (int i=0; i < a.nelem; ++i)
            c.data[i] = a.data[i] + b.data[i];
    }
    return c;
}

Vector subVector(Vector a, Vector b) {
    Vector c = {0,0};
    ...
    return c;
}

```


Ví dụ sử dụng

```
#include "vector.h"

void main() {
    int n = 10;
    Vector a,b,c;
    a = createVector(10,1.0);
    b = createVector(10,2.0);
    c = addVector(a,b);
    //...
    destroyVector(a);
    destroyVector(b);
    destroyVector(c);
}
```

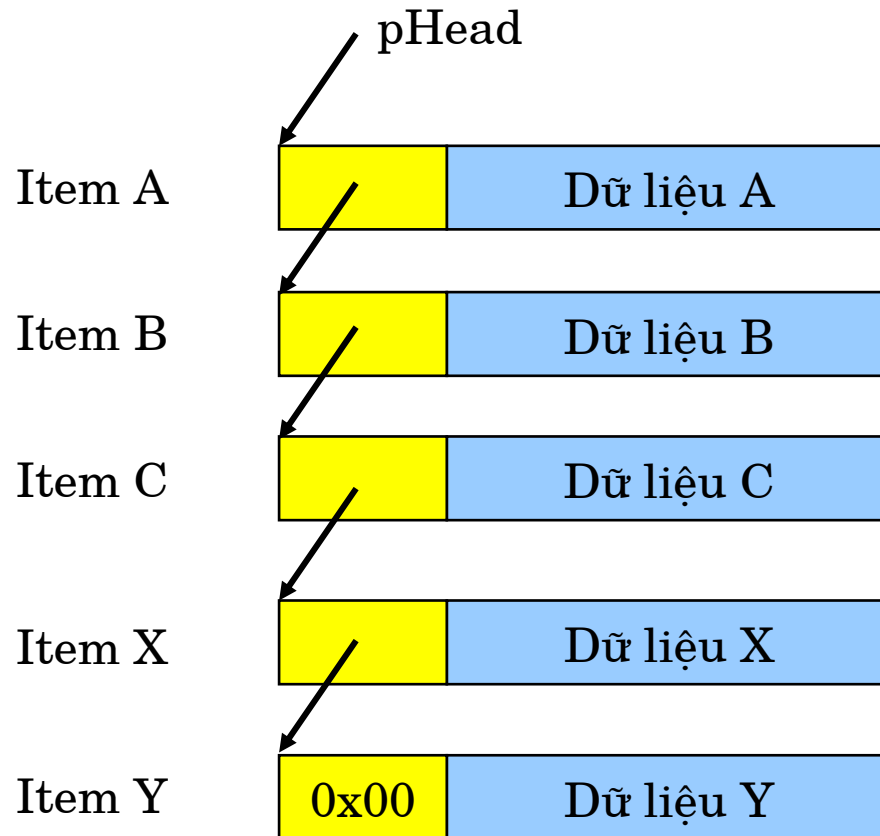
4.4 Xây dựng cấu trúc List

- Vấn đề: Xây dựng một cấu trúc để quản lý một cách hiệu quả và linh hoạt các dữ liệu động, ví dụ:
 - Hộp thư điện tử
 - Danh sách những việc cần làm
 - Các đối tượng đồ họa trên hình vẽ
 - Các khâu động học trong sơ đồ mô phỏng hệ thống (tương tự trong SIMULINK)
- Các yêu cầu đặc thù:
 - Số lượng mục dữ liệu trong danh sách có thể thay đổi thường xuyên
 - Các thao tác bổ sung hoặc xóa dữ liệu cần được thực hiện nhanh, đơn giản
 - Sử dụng tiết kiệm bộ nhớ

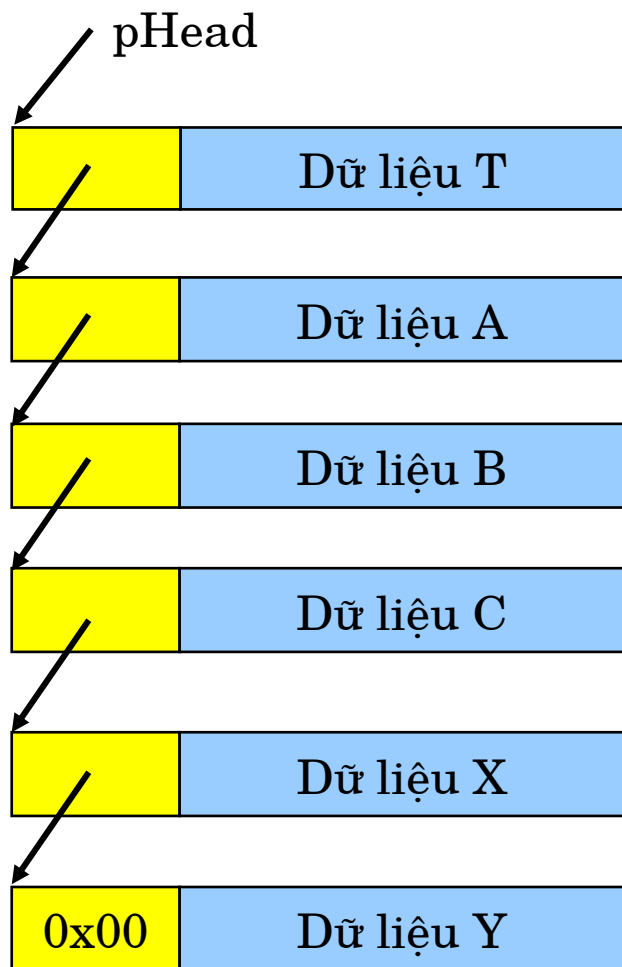
Sử dụng kiểu mảng?

- Số phần tử trong một mảng thực chất không bao giờ thay đổi được. Dung lượng bộ nhớ vào thời điểm cấp phát phải biết trước, không thực sự co giãn được.
- Nếu không thực sự sử dụng hết dung lượng đã cấp phát => lãng phí bộ nhớ
- Nếu đã sử dụng hết dung lượng và muốn bổ sung phần tử thì phải cấp phát lại và sao chép toàn bộ dữ liệu sang mảng mới => cần nhiều thời gian nếu số phần tử lớn
- Nếu muốn chèn một phần tử/xóa một phần tử ở đầu hoặc giữa mảng thì phải sao chép và dịch toàn bộ phần dữ liệu còn lại => rất mất thời gian

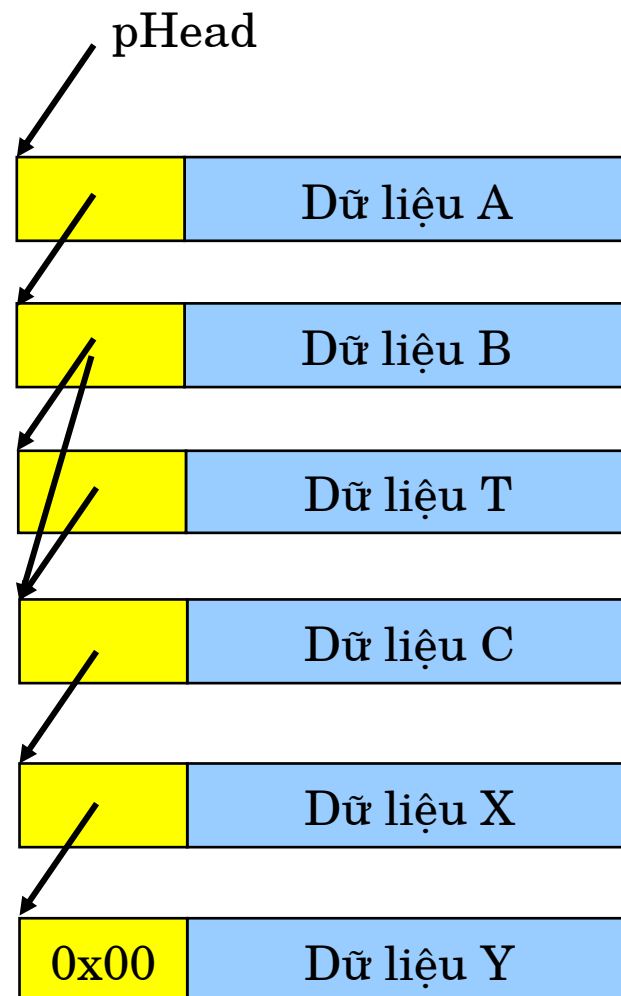
Danh sách móc nối (linked list)



Bổ sung dữ liệu

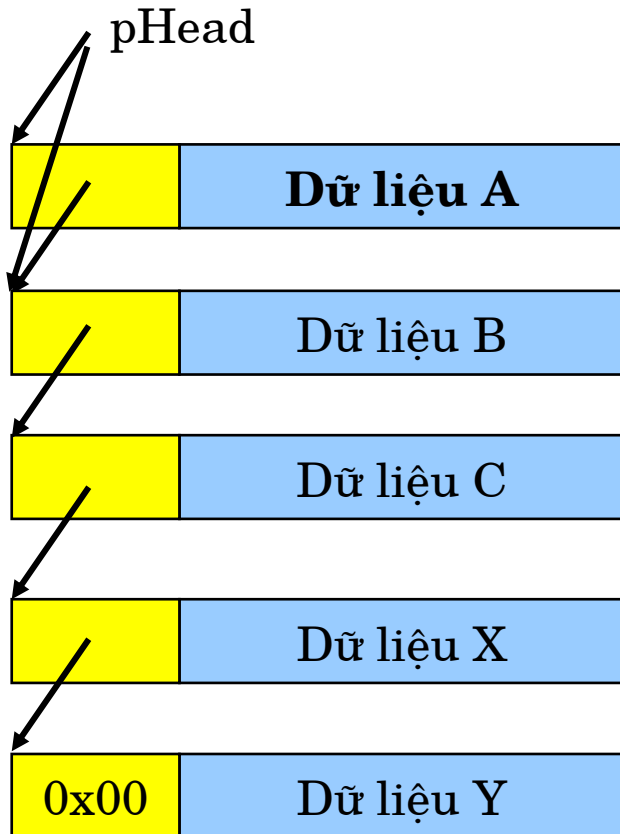


Bổ sung vào đầu danh sách

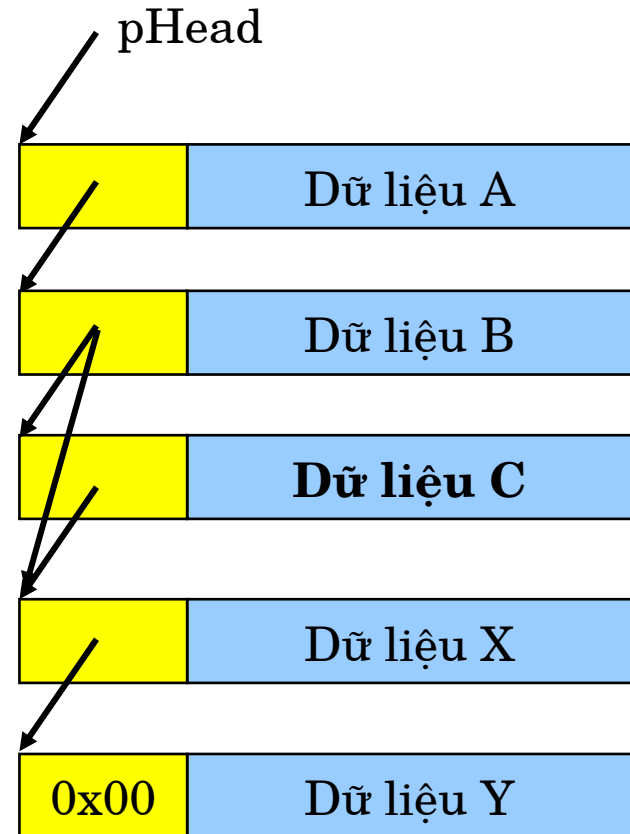


Bổ sung vào giữa danh sách

Xóa bớt dữ liệu



Xóa dữ liệu đầu danh sách



Xóa dữ liệu giữa danh sách

Các đặc điểm chính

- Ưu điểm:
 - Sử dụng rất linh hoạt, cấp phát bộ nhớ khi cần và xóa khi không cần
 - Bổ sung và xóa bỏ một dữ liệu được thực hiện thông qua chuyển con trỏ, thời gian thực hiện là hằng số, không phụ thuộc vào chiều dài và vị trí
 - Có thể truy nhập và duyệt các phần tử theo kiểu tuần tự
- Nhược điểm:
 - Mỗi dữ liệu bổ sung mới đều phải được cấp phát bộ nhớ động
 - Mỗi dữ liệu xóa bỏ đi đều phải được giải phóng bộ nhớ tương ứng
 - Nếu kiểu dữ liệu không lớn thì phần overhead chiếm tỉ lệ lớn
 - Tìm kiếm dữ liệu theo kiểu tuyến tính, mất thời gian

Ví dụ: Danh sách thông báo (hộp thư)

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

struct MessageItem {
    string subject;
    string content;
    MessageItem* pNext;
};

struct MessageList {
    MessageItem* pHead;
};

void initMessageList(MessageList& l);
void addMessage(MessageList&, const string& sj,
               const string& ct);
bool removeMessageBySubject(MessageList& l,
                           const string& sj);
void removeAllMessages(MessageList&);
```



```
#include "List.h"
void initMessageList(MessageList& l) {
    l.pHead = 0;
}
void addMessage(MessageList& l, const string& sj,
               const string& ct) {
    MessageItem* pItem = new MessageItem;
    pItem->content = ct;
    pItem->subject = sj;
    pItem->pNext = l.pHead;
    l.pHead = pItem;
}
void removeAllMessages(MessageList& l) {
    MessageItem *pItem = l.pHead;
    while (pItem != 0) {
        MessageItem* pItemNext = pItem->pNext;
        delete pItem;
        pItem = pItemNext;
    }
    l.pHead = 0;
}
```

```

bool removeMessageBySubject (MessageList& l,
                             const string& sj) {
    MessageItem* pItem = l.pHead;
    MessageItem* pItemBefore;
    while (pItem != 0 && pItem->subject != sj) {
        pItemBefore = pItem;
        pItem = pItem->pNext;
    }
    if (pItem != 0) {
        if (pItem == l.pHead)
            l.pHead = 0;
        else
            pItemBefore->pNext = pItem->pNext;
        delete pItem;
    }
    return pItem != 0;
}

```

Chương trình minh họa

```
#include <iostream>
#include "list.h"
using namespace std;
void main() {
    MessageList myMailBox;
    initMessageList(myMailBox);
    addMessage(myMailBox, "Hi", "Welcome, my friend!");
    addMessage(myMailBox, "Test", "Test my mailbox");
    addMessage(myMailBox, "Lecture Notes", "Programming Techniques");
    removeMessageBySubject(myMailBox, "Test");
    MessageItem* pItem = myMailBox.pHead;
    while (pItem != 0) {
        cout << pItem->subject << ":" << pItem->content << '\n';
        pItem = pItem->pNext;
    }
    char c;
    cin >> c;
    removeAllMessages(myMailBox);
}
```

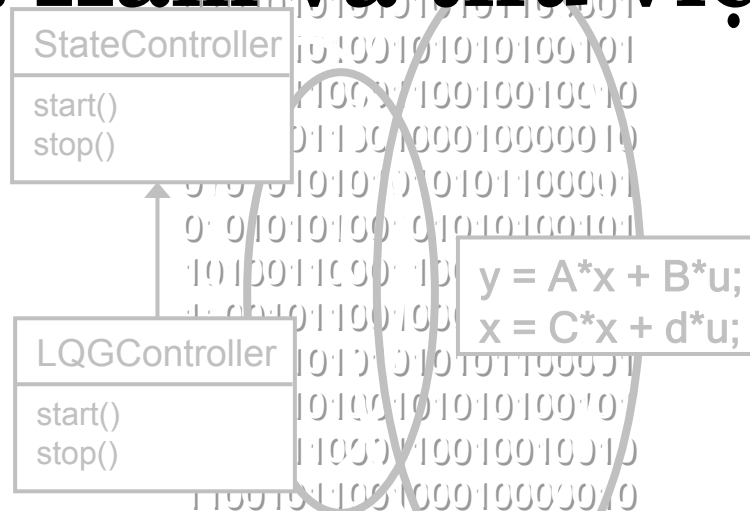
Bài tập về nhà

- Xây dựng kiểu danh sách móc nối chứa các ngày lễ trong năm và ý nghĩa của mỗi ngày (string), cho phép:
 - Bổ sung một ngày lễ vào đầu danh sách
 - Tìm ý nghĩa của một ngày (đưa ngày tháng là tham số)
 - Xóa bỏ đi một ngày lễ ở đầu danh sách
 - Xóa bỏ đi một ngày lễ ở giữa danh sách (đưa ngày tháng là tham số)
 - Xóa bỏ đi toàn bộ danh sách
- Viết chương trình minh họa cách sử dụng

Kỹ thuật lập trình

Phần II: Lập trình có cấu trúc

Chương 3: Hàm và thư viện



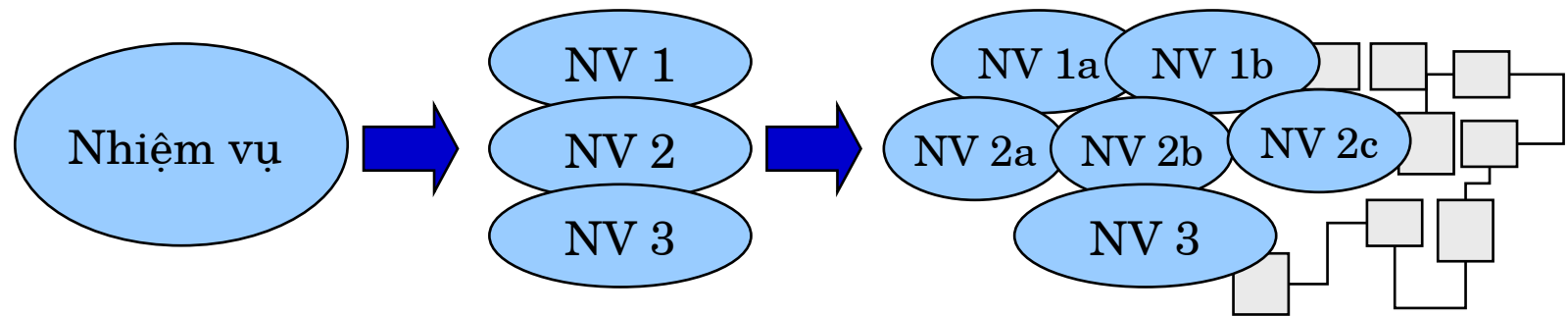
Nội dung chương 3

- 3.1 Hàm và lập trình hướng hàm
- 3.2 Khai báo, định nghĩa hàm
- 3.3 Truyền tham số và trả về kết quả
- 3.4 Thiết kế hàm và thư viện
- 3.5 Thư viện chuẩn ANSI-C
- 3.6 Làm việc với tệp tin sử dụng thư viện C++
- 3.7 Nạp chồng tên hàm C++
- 3.8 Hàm inline trong C++

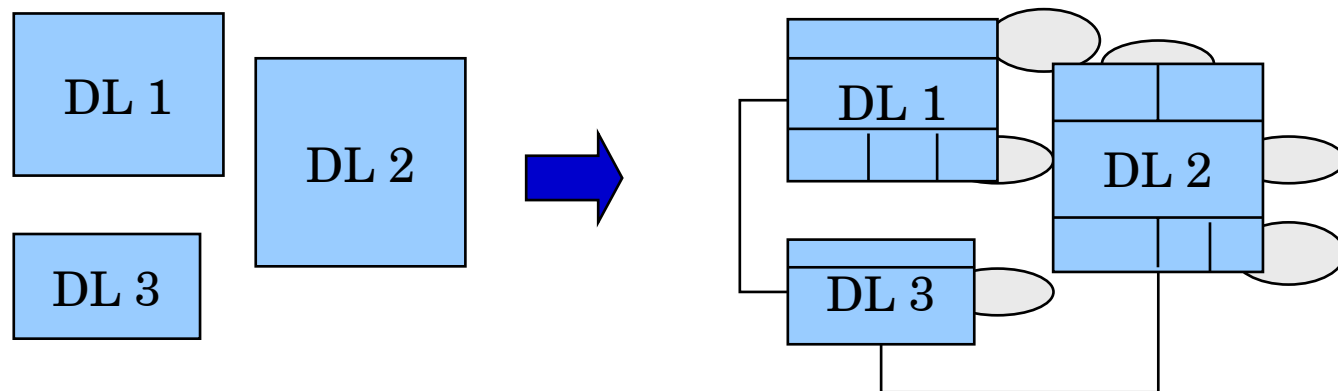
3.1 Hàm và lập trình hướng hàm

Lập trình có cấu trúc có thể dựa trên một trong hai phương pháp:

- Lập trình hướng hàm (*function-oriented*), còn gọi là hướng nhiệm vụ (*task-oriented*), hướng thủ tục (*procedure-oriented*)



- Lập trình hướng dữ liệu (*data-oriented*)



Hàm là gì?

- Tiếng Anh: function -> hàm, chức năng
- Một đơn vị tổ chức chương trình, một đoạn mã chương trình có cấu trúc để thực hiện một **chức năng** nhất định, có **giá trị sử dụng lại**
- Các hàm có quan hệ với nhau thông qua lời gọi, các biến tham số (đầu vào, đầu ra) và giá trị trả về
- Cách thực hiện cụ thể một hàm phụ thuộc nhiều vào dữ kiện (tham số, đối số của hàm):
 - Thông thường, kết quả thực hiện hàm mỗi lần đều giống nhau nếu các tham số đầu vào như nhau
 - Một hàm không có tham số thì giá trị sử dụng lại rất thấp
- Trong C/C++: Không phân biệt giữa thủ tục và hàm, cả đoạn mã chương trình chính cũng là hàm

Ví dụ phân tích

- Yêu cầu bài toán: Tính tổng một dãy số nguyên (liên tục) trong phạm vi do người sử dụng nhập. In kết quả ra màn hình.
- Các nhiệm vụ:
 - Nhập số nguyên thứ nhất:
 - Yêu cầu người sử dụng nhập
 - Nhập số vào một biến
 - Nhập số nguyên thứ hai
 - Yêu cầu người sử dụng nhập
 - Nhập số vào một biến
 - Tính tổng với vòng lặp
 - Hiển thị kết quả ra màn hình

Phương án 4 trong 1

```
#include <iostream.h>
void main() {
    int a, b;
    char c;
    do {
        cout << "Enter the first integer number: ";
        cin >> a;
        cout << "Enter the second integer number: ";
        cin >> b;
        int Total = 0;
        for (int i = a; i <= b; ++i)
            Total += i;
        cout << "The sum from " << a << " to " << b
            << " is " << Total << endl;
        cout << "Do you want to continue? (Y/N):";
        cin >> c;
    } while (c == 'y' || c == 'Y');
}
```

Phương án phân hoạch hàm (1)

```
#include <iostream.h>

int  ReadInt();
int  SumInt(int,int);
void WriteResult(int a, int b, int kq);

void main() {
    char c;
    do {
        int a = ReadInt();
        int b = ReadInt();
        int T = SumInt(a,b);
        WriteResult(a,b,T);
        cout << "Do you want to continue? (Y/N):";
        cin  >> c;
    } while (c == 'y' || c == 'Y');
}
```

Phương án phân hoạch hàm (1)

```
int ReadInt() {  
    cout << "Enter an integer number: ";  
    int N;  
    cin >> N;  
    return N;  
}
```

Không có tham số,
Giá trị sử dụng lại?

OK,
Không thể tốt hơn!

```
int SumInt(int a, int b) {  
    int Total = 0;  
    for (int i = a; i <= b; ++i)  
        Total += i;  
    return Total;  
}
```

Quá nhiều tham số,
Hiệu năng?

```
void WriteResult(int a, int b, int kq) {  
    cout << "The sum from " << a << " to " << b  
        << " is " << kq << endl;  
}
```

Phương án phân hoạch hàm (1)

- Chương trình dễ đọc hơn => dễ phát hiện lỗi
 - Chương trình dễ mở rộng hơn
 - Hàm SumInt có thể sử dụng lại tốt
 - Mã nguồn dài hơn
 - Mã chạy lớn hơn
 - Chạy chậm hơn
- ➔ Không phải cứ phân hoạch thành nhiều hàm là tốt, mà vấn đề nằm ở cách phân hoạch và thiết kế hàm làm sao cho **tối ưu!**

Phương án phân hoạch hàm (2)

```
#include <iostream.h>

int  ReadInt(const char*);
int  SumInt(int,int);

void main() {
    char c;
    do {
        int a = ReadInt("Enter the first integer number :");
        int b = ReadInt("Enter the second integer number:");
        cout << "The sum from " << a << " to " << b
              << " is " << SumInt(a,b) << endl;
        cout << "Do you want to continue? (Y/N):";
        cin  >> c;
    } while (c == 'y' || c == 'Y');
}
```

Phương án phân hoạch hàm (2)

```
int ReadInt(const char* userPrompt) {  
    cout << userPrompt;  
    int N;  
    cin >> N;  
    return N;  
}
```

OK,
Đã tốt hơn!

```
int SumInt(int a, int b) {  
    int Total = 0;  
    for (int i = a; i <= b; ++i)  
        Total += i;  
    return Total;  
}
```

3.2 Khai báo và định nghĩa hàm

- Định nghĩa hàm: tạo mã thực thi hàm

```
Kiểu trả về  Tên hàm  Tham biến (hình thức)
   ↓           ↓           ↓           ↓
int  SumInt(int a, int b) {
    int Total = 0;
    for (int i = a; i <= b; ++i)
        Total += i;
    return Total;
}
```

- Khai báo hàm thuần túy: không tạo mã hàm

```
int SumInt(int a, int b);
   ↑       ↑       ↑       ↑
Kiểu trả về Tên hàm Kiểu tham biến
```

- Tại sao và khi nào cần khai báo hàm?

Khái báo hàm và lời gọi hàm

- Ý nghĩa của khai báo hàm:
 - Khi cần sử dụng hàm (gọi hàm)
 - Trình biên dịch cần lời khai báo hàm để kiểm tra lời gọi hàm đúng hay sai về cú pháp, về số lượng các tham số, kiểu các tham số và cách sử dụng giá trị trả về.

```
int SumInt(int a, int b);
```

- Có thể khai báo hàm độc lập với việc định nghĩa hàm (tất nhiên phải đảm bảo nhất quán)
- Gọi hàm: yêu cầu thực thi mã hàm với tham số thực tế (tham trị)

```
int x = 5;
```

```
int k = SumInt(x, 10);
```

↑ ↑ ↑
Tên hàm Tham số (gọi hàm)

Khi biên dịch chưa cần phải có định nghĩa hàm, nhưng phải có khai báo hàm!

Khai báo hàm C/C++ ở đâu?

- Ở phạm vi toàn cục (ngoài bất cứ hàm nào)
- Một hàm phải được khai báo trước lời gọi đầu tiên trong một tệp tin mã nguồn
- Nếu sử dụng nhiều hàm thì sẽ cần rất nhiều dòng mã khai báo (mất công viết, dễ sai và mã chương trình lớn lên?):
 - Nếu người xây dựng hàm (định nghĩa hàm) đưa sẵn tất cả phần khai báo vào trong một tệp tin => **Header file** (*.h, *.hx,...) thì người sử dụng chỉ cần bổ sung dòng lệnh
`#include <filename>`
 - Mã chương trình không lớn lên, bởi khai báo không sinh mã!
- Một hàm có thể khai báo nhiều lần tùy ý!

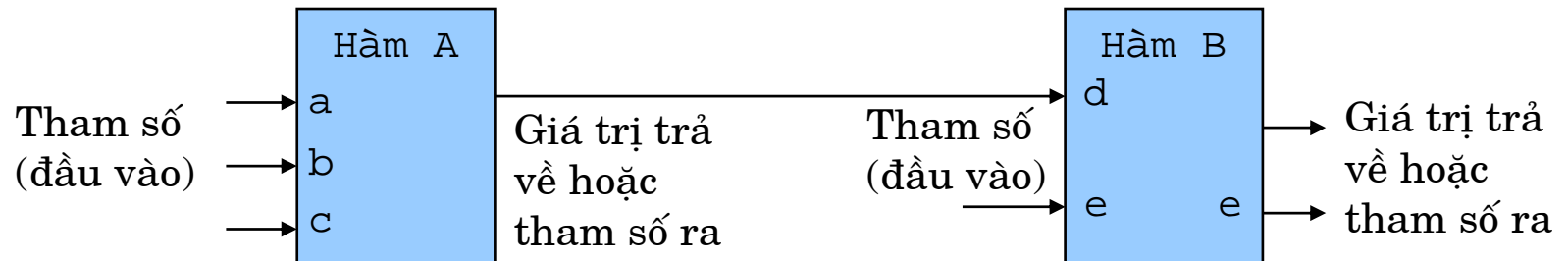
Định nghĩa hàm ở đâu?

- Ở phạm vi toàn cục (ngoài bất cứ hàm nào)
- Có thể định nghĩa trong cùng tệp tin với mã chương trình chính, hoặc tách ra một tệp tin riêng. Trong Visual C++:
 - * .c => C compiler,
 - * .cpp => C++ compiler
- Một hàm đã có lời gọi thì phải được định nghĩa chính xác 1 lần trong toàn bộ (dự án) chương trình, trước khi gọi trình liên kết (lệnh Build trong Visual C++)
- Đưa tệp tin mã nguồn vào dự án, không nên:

```
#include "xxx.cpp"
```
- Một hàm có được định nghĩa bằng C, C++, hợp ngữ hoặc bằng một ngôn ngữ khác và dùng trong C/C++ => Sử dụng hàm không cần mã nguồn!
- Một thư viện cho C/C++ bao gồm:
 - Header file (thường đuôi *.h, *.hxx, ..., nhưng không bắt buộc)
 - Tệp tin mã nguồn (*.c, *.cpp, *.cxx, ...) hoặc mã đích (*.obj, *.o, *.lib, *.dll, ...)

3.3 Truyền tham số và trả về kết quả

- Truyền tham số và trả về kết quả là phương pháp cơ bản để tổ chức quan hệ giữa các hàm (giữa các chức năng trong hệ thống)



- Ngoài ra, còn có các cách khác:
 - Sử dụng biến toàn cục: nói chung là không nên!
 - Sử dụng các tệp tin, streams: dù sao vẫn phải sử dụng tham số để nói rõ tệp tin nào, streams nào
 - Các cơ chế giao tiếp hệ thống khác (phụ thuộc vào hệ điều hành, nền tảng và giao thức truyền thông) => nói chung vẫn cần các tham số bổ sung
- Truyền tham số & trả về kết quả là một vấn đề cốt lõi trong xây dựng và sử dụng hàm, một trong những yếu tố ảnh hưởng quyết định tới chất lượng phần mềm!

Tham biến hình thức và tham số thực tế

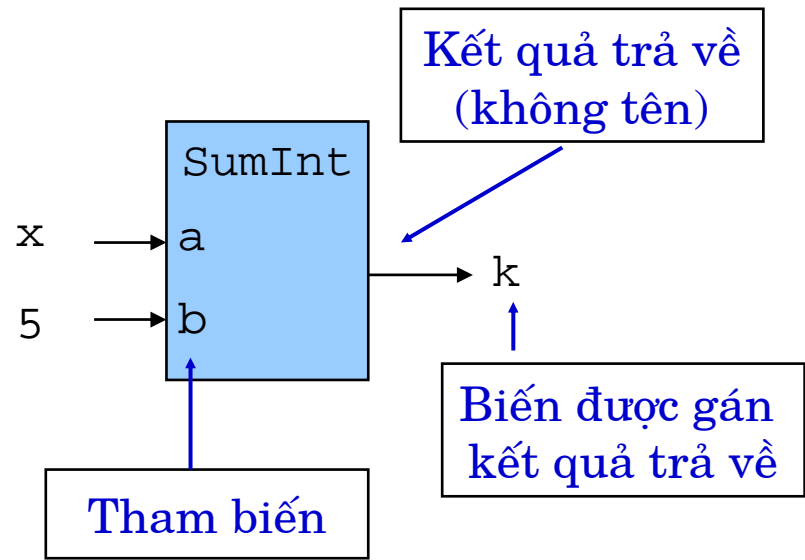
```
int SumInt(int a, int b) {  
    ...  
}
```

Tham biến
(hình thức)

```
int x = 5;  
int k = SumInt(x, 10);  
...
```

Tham số
(thực tế)

```
int a = 2;  
k = SumInt(a, x);
```



3.3.1 Truyền giá trị

```
int SumInt(int, int);
```

```
// Function call
```

```
void main() {
```

```
    int x = 5;
```

```
    int k = SumInt(x, 10);
```

```
    ...
```

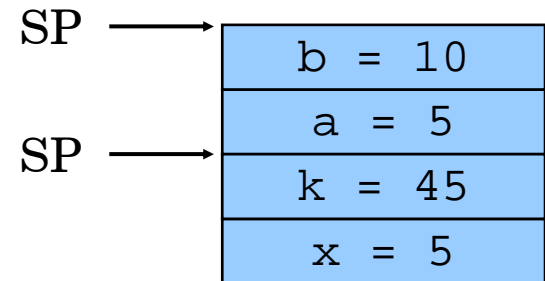
```
}
```

```
// Function definition
```

```
int SumInt(int a, int b) {
```

```
    ...
```

```
}
```



Ngăn xếp

Thử ví dụ đọc từ bàn phím

```
#include <iostream.h>
void ReadInt(const char* userPrompt, int N) {
    cout << userPrompt;
    cin >> N;
}

void main() {
    int x = 5;
    ReadInt("Input an integer number:", x);
    cout << "Now x is " << x;
    ...
}
```

- Kết quả: x không hề thay đổi sau đó.

Truyền giá trị

- Truyền giá trị là cách thông thường trong C
- Tham biến chỉ nhận được bản sao của biến đầu vào (tham số thực tế)
- Thay đổi tham biến chỉ làm thay đổi vùng nhớ cục bộ, không làm thay đổi biến đầu vào
- Tham biến chỉ có thể mang tham số đầu vào, không chứa được kết quả (tham số ra)
- Truyền giá trị khá an toàn, tránh được một số hiệu ứng phụ
- Truyền giá trị trong nhiều trường hợp kém hiệu quả do mất công sao chép dữ liệu

3.3.2 Truyền địa chỉ

```
int SumInt(int* p, int N);  
// Function call
```

```
void main() {  
    int a[] = {1, 2, 3, 4};  
    int k = SumInt(a,4);
```

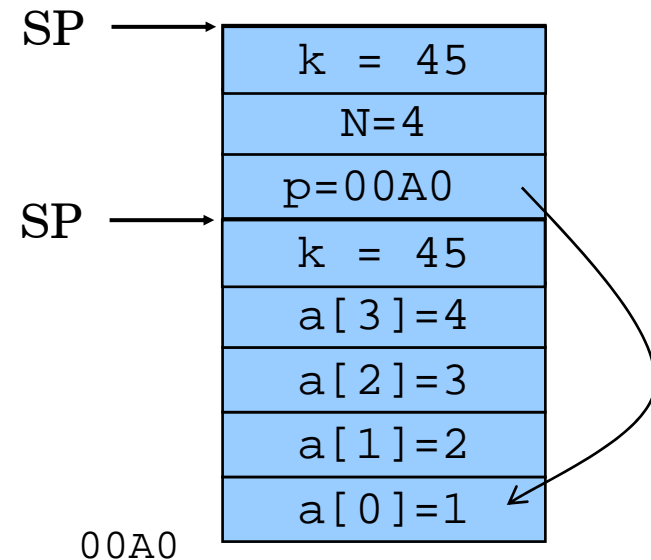
....

```
}
```

```
// Function definition
```

```
int SumInt(int* p, int N) {  
    int *p2 = p + N, k = 0;  
    while (p < p2)  
        k += *p++;  
    return k;
```

```
}
```



Truyền mảng tham số?

```
int SumInt(int p[4], int N);  
    // Function call  
void main() {  
    int a[] = {1, 2, 3, 4};  
    int k = SumInt(a,4);  
    ...  
}  
  
// Function definition  
int SumInt(int p[4] int N) {  
    int *p2 = p + N, k = 0;  
    while (p < p2)  
        k += *p++;  
    return k;  
}
```

Bản chất như
trong ví dụ trước:
Truyền địa chỉ!

Thử lại ví dụ đọc từ bàn phím

```
#include <iostream.h>

void ReadInt(const char* userPrompt, int* pN) {
    cout << userPrompt;
    cin >> *pN;
}
```

```
void main() {
    int x = 5;
    ReadInt("Input an integer number:", &x);
    cout << "Now x is " << x;
    ...
}
```

- Kết quả: x thay đổi giá trị sau đó (cũng là lý do tại sao hàm scanf() lại yêu cầu kiểu tham biến là con trỏ!)

Khi nào sử dụng truyền địa chỉ?

- Khi cần thay đổi "biến đầu vào" (truy nhập trực tiếp vào ô nhớ, không qua bản sao)
- Khi kích cỡ kiểu dữ liệu lớn => tránh sao chép dữ liệu vào ngăn xếp
- Truyền tham số là một mảng => bắt buộc truyền địa chỉ
- Lưu ý: Sử dụng con trỏ để truyền địa chỉ của vùng nhớ dữ liệu đầu vào. Bản thân con trỏ có thể thay đổi được trong hàm nhưng địa chỉ vùng nhớ không thay đổi (nội dung của vùng nhớ đó thay đổi được): xem ví dụ biến p trong hàm SumInt trang 21.

3.3.3 Truyền tham chiếu (C++)

```
#include <iostream.h>

void ReadInt(const char* userPrompt, int& N) {
    cout << userPrompt;
    cin >> N;
}

void main() {
    int x = 5;
    ReadInt("Input an integer number:", x);
    cout << "Now x is " << x;
    ...
}

▪ Kết quả: x thay đổi giá trị sau đó
```

Thử ví dụ hàm swap

```
#include <iostream.h>
void swap(int& a, int& b) {
    int temp = a;
    a = b;
    b = temp;
}

void main() {
    int x = 5, y = 10;
    swap(x,y);
    cout << "Now x is " << x << ", y is " << y;
    ...
}
```

Khi nào sử dụng truyền tham chiếu?

- Chỉ trong C++
- Khi cần thay đổi "biến đầu vào" (truy nhập trực tiếp vào ô nhớ, không qua bản sao)
- Một tham biến tham chiếu có thể đóng vai trò là đầu ra (chứa kết quả), hoặc có thể vừa là đầu vào và đầu ra
- Khi kích cỡ kiểu dữ liệu lớn => tránh sao chép dữ liệu vào ngăn xếp, ví dụ:

```
void copyData(const Student& sv1, Student& sv2) {  
    sv2.birthday = sv1.birthday;  
    ...  
}
```

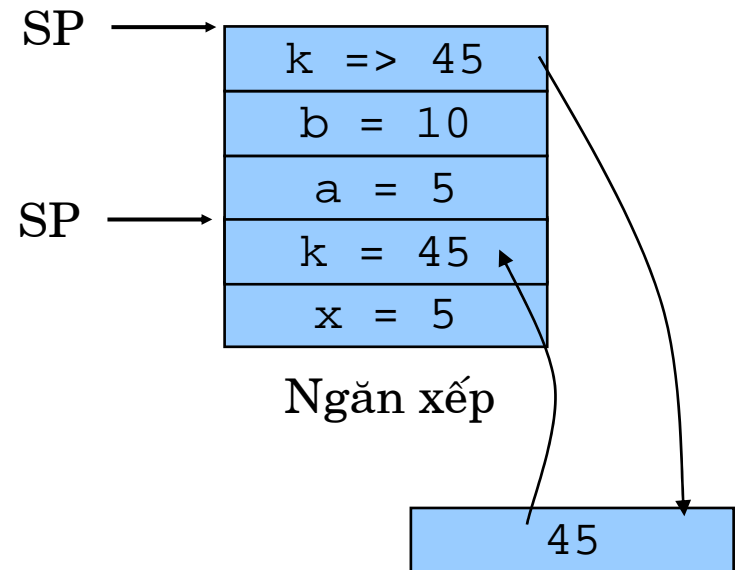
3.3.4 Kiểu trả về

- Kiểu trả về: gần như tùy ý, chỉ không thể trả về trực tiếp một mảng
- Về nguyên tắc, có thể trả về kiểu:
 - Giá trị
 - Con trỏ
 - Tham chiếu
- Tuy nhiên, cần rất thận trọng với trả về địa chỉ hoặc tham chiếu:
 - Không bao giờ trả về con trỏ hoặc tham chiếu vào biến cục bộ
 - Không bao giờ trả về con trỏ hoặc tham chiếu vào tham biến truyền qua giá trị
- Với người lập trình ít có kinh nghiệm: chỉ nên trả về kiểu giá trị

Cơ chế trả về

```
int SumInt(int a, int b) {  
    int k = 0;  
    for (int i=a; i <= b; ++i)  
        k +=i;  
    return k;  
}
```

```
void main() {  
    int x = 5, k = 0;  
    k = SumInt(x,10);  
    ...  
}
```



Trả về con trỏ

- Viết hàm trả về địa chỉ của phần tử lớn nhất trong một mảng:

```
int* FindMax(int* p, int n) {
    int *pMax = p;
    int *p2 = p + n;
    while (p < p2) {
        if (*p > *pMax)
            pMax = p;
        ++p;
    }
    return pMax;
}

void main() {
    int s[5] = { 1, 2, 3, 4, 5};
    int *p = FindMax(s,5);
}
```

Lý do trả về con trỏ hoặc tham chiếu

- Tương tự như lý do truyền địa chỉ hoặc truyền tham chiếu:
 - Tránh sao chép dữ liệu lớn không cần thiết
 - Để có thể truy cập trực tiếp và thay đổi giá trị đầu ra
- Có thể trả về con trỏ hoặc tham chiếu vào đâu?
 - Vào biến toàn cục
 - Vào tham số truyền cho hàm qua địa chỉ hoặc qua tham chiếu
 - Nói chung: vào vùng nhớ mà còn tiếp tục tồn tại sau khi kết thúc hàm
- Con trỏ lại phức tạp thêm một chút?

Phản ví dụ: trả về con trỏ

```
int* f(int* p, int n) {
    int Max = *p;
    int *p2 = p + n;
    while (p < p2) {
        if (*p > Max)
            Max = *p;
        ++p;
    }
    return &Max;
}

void main() {
    int s[5] = { 1, 2, 3, 4, 5};
    int *p = FindMax(s,5); // get invalid address
}
```

Các ví dụ nghiên cứu: **Đúng** / **sai**?

```
int* f1(int a) {  
    ...  
    return &a;  
}
```

```
int& f2(int &a) {  
    ...  
    return a;  
}
```

```
int f3(int &a) {  
    ...  
    return a;  
}
```

```
int* f4(int *pa) {  
    ...  
    return pa;  
}
```

```
int f5(int *pa) {  
    ...  
    return *pa;  
}
```

```
int& f6(int *pa) {  
    ...  
    return *pa;  
}
```

```
int& f7(int a) {  
    ...  
    return a;  
}
```

```
int *pa;  
int* f8() {  
    ...  
    return pa;  
}
```

3.4 Thiết kế hàm và thư viện

- Viết một chương trình chạy tốt đã khó, viết một thư viện hàm tốt còn khó hơn!
- Một thư viện hàm định nghĩa:
 - một tập hợp các hàm (có liên quan theo một chủ đề chức năng)
 - những kiểu dữ liệu sử dụng trong các hàm
 - một số biến toàn cục (rất hạn chế)
- Một thư viện hàm tốt cần phải:
 - Thực hiện những chức năng hữu ích
 - Đơn giản, dễ sử dụng
 - Hiệu suất và độ tin cậy cao
 - Trọn vẹn, nhất quán và đồng bộ

Thiết kế hàm

- Phân tích yêu cầu:
 - Làm rõ các dữ kiện (đầu vào) và kết quả (đầu ra)
 - Tìm ra các chức năng cần thực hiện
- Đặt tên hàm: ngắn gọn, ý nghĩa xác đáng, tự miêu tả
 - Hàm chỉ hành động: Chọn tên hàm là một động từ kết hợp với kiểu đối tượng chủ thể, ví dụ `printVector`, `displayMatrix`, `addComplex`, `sortEventQueue`, `filterAnalogSignal`,...
 - Hàm truy nhập thuộc tính: Có thể chọn là động từ hoặc danh từ kết hợp kiểu đối tượng chủ thể, ví dụ `length`, `size`, `numberOfColumns`, `getMatrixElem`, `putShapeColor`
 - Trong C++ nhiều hàm có thể giống tên (nạp chồng tên hàm), có thể chọn tên ngắn, ví dụ `sort`, `print`, `display`, `add`, `putColor`, `getColor` => nguyên tắc đa hình/đa xạ theo quan điểm hướng đối tượng
 - Trong C++ còn có thể định nghĩa hàm toán tử để có thể sử dụng các ký hiệu toán tử định nghĩa sẵn như `*`, `/`, `+`, `-` thay cho lời gọi hàm.

- Chọn tham số đầu vào (\Rightarrow tham biến)
 - Đặc tả ý nghĩa: Thể hiện rõ vai trò tham số
 - Đặt tên: Ngắn gọn, tự mô tả
 - Chọn kiểu: Kiểu nhỏ nhất mà đủ biểu diễn
 - Chọn cách truyền tham số: cân nhắc giữa truyền giá trị hay truyền địa chỉ/tham chiếu vào kiểu hằng
- Chọn tham số đầu ra (\Rightarrow tham biến truyền qua địa chỉ/qua tham chiếu hoặc sử dụng giá trị trả về)
 - Đặc tả ý nghĩa, đặt tên, chọn kiểu tương tự như tham số đầu vào
- Định nghĩa bổ sung các kiểu dữ liệu mới như cần thiết
- Mô tả rõ tiền trạng (*pre-condition*): điều kiện biên cho các tham số đầu vào và các điều kiện ngoại cảnh cho việc gọi hàm
- Mô tả rõ hậu trạng (*post-condition*): tác động của việc sử dụng hàm tới ngoại cảnh, các thao tác bắt buộc sau này,...
- Thiết kế thân hàm dựa vào các chức năng đã phân tích, sử dụng lưu đồ thuật toán với các cấu trúc điều kiện/rẽ nhánh (kể cả vòng lặp) \Rightarrow có thể phân chia thành các hàm con nếu cần

Ví dụ minh họa: Tìm số nguyên tố

Bài toán: Xây dựng hàm tìm N số nguyên tố đầu tiên!

- Phân tích:
 - Dữ kiện: N - số số nguyên tố đầu tiên cần tìm
 - Kết quả: Một dãy N số nguyên tố đầu tiên
 - Các chức năng cần thực hiện:
 - Nhập dữ liệu? KHÔNG!
 - Kiểm tra dữ kiện vào (N)? Có/không (Nếu kiểm tra mà N nhỏ hơn 0 thì hàm làm gì?)
 - Cho biết k số nguyên tố đầu tiên, xác định số nguyên tố tiếp theo
 - Lưu trữ kết quả mỗi lần tìm ra vào một cấu trúc dữ liệu phù hợp (dãy số cần tìm)
 - In kết quả ra màn hình? KHÔNG!

- Đặt tên hàm: `findPrimeSequence`
- Tham số vào: 1
 - Ý nghĩa: số các số nguyên tố cần tìm
 - Tên: `N`
 - Kiểu: số nguyên đủ lớn (`int/long`)
 - Truyền tham số: qua giá trị
- Tham số ra: 1
 - Ý nghĩa: dãy `N` số nguyên tố đầu tiên tính từ 1
 - Giá trị trả về hay tham biến? **Tham biến!**
 - Tên: `primes`
 - Kiểu: mảng số nguyên (của `int/long`)
 - Truyền tham số: qua địa chỉ (`int*` hoặc `long*`)
- Tiền trạng:
 - Tham số `N` phải là số không âm (có nên chọn kiểu `unsigned`?)
 - `primes` phải mang địa chỉ của mảng số nguyên có ít nhất `N` phần tử
- Hậu trạng: không có gì đặc biệt

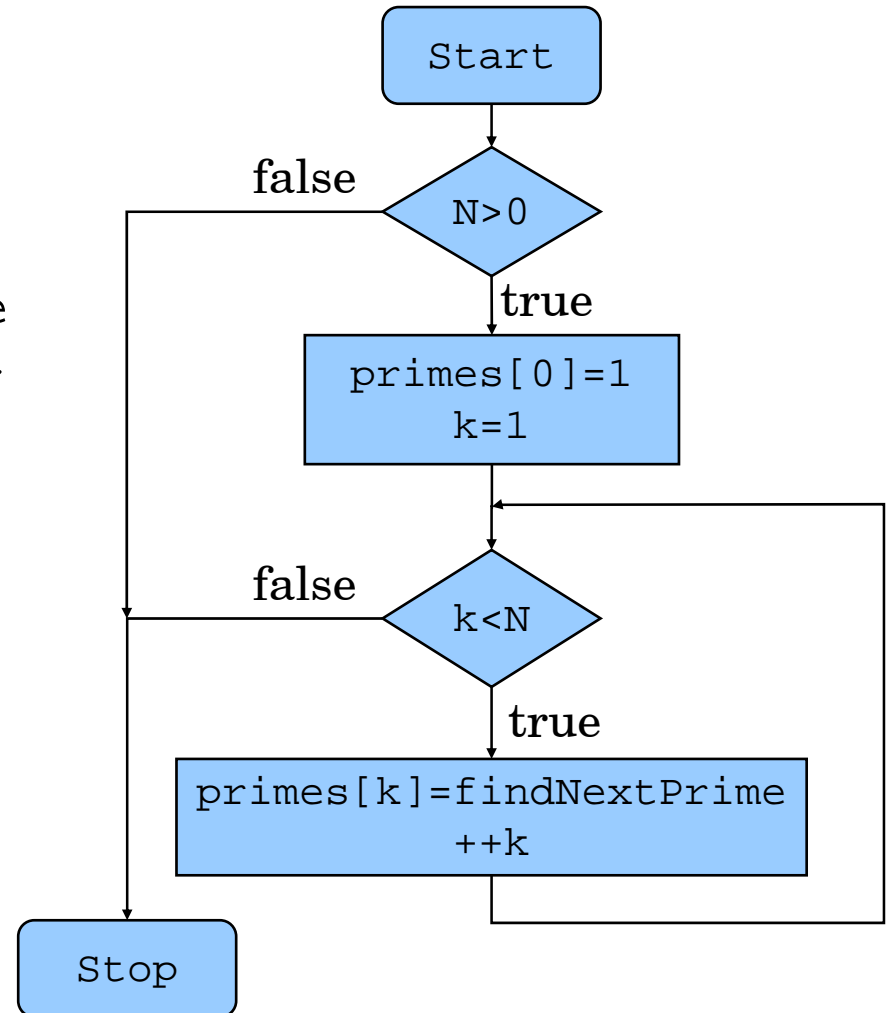
- Khai báo hàm:

```
void findPrimeSequence(int N, int* primes);
```

- Thiết kế thân hàm

- Lưu đồ thuật toán như hình vẽ
- Phân chia, bổ sung một hàm mới: `findNextPrime`

- Lặp lại qui trình thiết kế hàm cho `findNextPrime`
(Bài tập về nhà!)



3.5 Thư viện chuẩn ANSI-C

- Thư viện vào/ra (nhập/xuất) `<stdio.h>`
- Xử lý ký tự và chuỗi ký tự `<string.h>`, `<ctype.h>`
- Thư viện hàm toán `<math.h>`, `<float.h>`
- Thời gian, ngày tháng `<time.h>`, `<locale.h>`
- Cấp phát bộ nhớ động `<stdlib.h>`
- Các hàm ký tự rộng `<wchar.h>`, `<wctype.h>`
- Các hàm khác `<stdlib.h>`, ...

3.6 Làm việc với tệp tin trong C++

```
#include <iostream.h>
```

```
#include <fstream.h>
```

- Khai báo một biến:

```
ifstream fin; // input
```

```
ofstream fout; // output
```

```
fstream fio; // input and output
```

- Mở/tạo một tệp tin:

```
fin.open("file1.txt");
```

```
fout.open("file2.dat");
```

```
fio.open("file3.inf");
```

- Kết hợp khai báo biến và mở/tạo một tệp tin

```
ifstream fin("file1.txt"); // input
```

```
ofstream fout("file2.inf"); // output
```

```
fstream fio("file3.dat"); // input and output
```

- Ghi dữ liệu ra tệp tin
 - Tương tự như sử dụng `cout`
 - Tệp tin có thể chứa dữ liệu kiểu hỗn hợp, ví dụ:

```
fout << "Nguyen Van A" << endl;  
fout << 21 << endl << false;
```
- Đọc dữ liệu từ một tệp tin
 - Tương tự như sử dụng `cin`

```
char name[32];  
int age, married;  
fin.getline(name, 32);  
fin >> age >> married;
```
- Đóng một tệp tin:
 - Tự động khi kết thúc phạm vi `{ }`,
 - Hoặc gọi hàm thành viên `close()`:

```
fin.close();  
fout.close();  
fio.close();
```

Ví dụ: làm việc với tệp tin

```
#include <iostream.h>
#include <fstream.h>
void main() {
    {
        ofstream fout("file1.dat");// output
        fout << "Nguyen Van A" << endl << 21 << endl << false;
    }
    {
        ifstream fin("file1.dat"); // input
        char name[32];
        int age;
        int married;
        fin.getline(name,32);
        fin >> age >> married;
        cout << "Name:\t" << name << endl;
        cout << "Age:\t" << age << endl;
        cout << "Married:" << (married ? "Yes" : "No");
    }
    char c;
    cin >> c;
}
```

3.7 Nạp chồng tên hàm trong C++

- Trong C++ có thể xây dựng nhiều hàm có cùng tên, ví dụ:

```
int    max(int a, int b);  
double max(double a, double b);  
double max(double a, double b, double c);  
double max(double *seq, int n);
```

- Mục đích của **nạp chồng tên hàm** là:
 - Đơn giản hóa cho người xây dựng hàm trong việc chọn tên (thay vì `maxInt`, `maxDouble`, `maxDouble3`, `maxDoubleSequence`, ...)
 - Đơn giản hóa cho người sử dụng hàm, chỉ cần nhớ 1 tên quen thuộc thay cho nhiều tên phức tạp

Ví dụ: định nghĩa các hàm max()

```
int max(int a, int b) { // (1)
    return (a > b)? a : b;
}
double max(double a, double b) { // (2)
    return (a > b)? a : b;
}
double max(double a, double b, double c); { // (3)
    if (a < b) a = b;
    if (a < c) a = c;
    return a;
}
double max(double *seq, int n) { // (4)
    int i = 0, kq = seq[0];
    while (i < n) {
        if (kq < seq[i])kq = seq[i];
        ++i;
    }
    return kq;
}
```

Ví dụ: sử dụng các hàm max()

```
int max(int a, int b); // (1)
double max(double a, double b); // (2)
double max(double a, double b, double c); // (3)
double max(double *seq, int n); // (4)
```

```
void main() {
    int k = max(5,7); // call (1)
    double d = max(5.0,7.0); // call (2)
    double a[] = {1,2,3,4,5,6};
    d = max(d, a[1], a[2]); // call (3)
    d = max(a, 5); // call (4)
    d = max(5,7); // ?
    d = max(d, 5); // ?
}
```

➔ Đẩy trách nhiệm kiểm tra và tìm hàm phù hợp cho compiler!

Một số qui tắc về nạp chồng tên hàm

- Các hàm cùng tên được **định nghĩa cùng trong một file/ trong một thư viện** hoặc **sử dụng trong cùng một chương trình** phải khác nhau ít nhất về:
 - Số lượng các tham số, hoặc
 - Kiểu của ít nhất một tham số (`int` khác `short`, `const int` khác `int`, `int` khác `int&`, ...)
- ➔ Không thể chỉ khác nhau ở kiểu trả về
- Tại sao vậy?
 - Compiler cần có cơ sở để quyết định gọi hàm nào
 - Dựa vào cú pháp trong lời gọi (số lượng và kiểu các tham số thực tế) compiler sẽ chọn hàm có cú pháp phù hợp nhất
 - Khi cần compiler có thể tự động chuyển đổi kiểu theo chiều hướng **hợp lý nhất** (vd `short=>int`, `int => double`)

3.8 Hàm inline trong C++

- Vấn đề: Hàm tiện dụng, nhưng nhiều khi hiệu suất không cao, đặc biệt khi mã thực thi hàm ngắn
 - Các thủ tục như nhớ lại trạng thái chương trình, cấp phát bộ nhớ ngăn xếp, sao chép tham số, sao chép giá trị trả về, khôi phục trạng thái chương trình **mất nhiều thời gian**
 - Nếu mã thực thi hàm ngắn thì **sự tiện dụng không bù so với sự lãng phí thời gian**
- Giải pháp trong C: Sử dụng macro, ví dụ

```
#define max(a,b) a>b?a:b
```

 - Vấn đề: Macro do tiền xử lý chạy (preprocessor), không có kiểm tra kiểu, không có phân biệt ngữ cảnh => gây ra các hiệu ứng phụ không mong muốn

Ví dụ dòng lệnh `l=max(k*5-2,1);`
sẽ được thay thế bằng `l=k*5-2>k?k*5-2:1; // OOPS!`

 - Những cách giải quyết như thêm dấu ngoặc chỉ làm mã khó đọc, không khắc phục triệt để các nhược điểm

Giải pháp hàm inline trong C++

- Điều duy nhất cần làm là thêm từ khóa `inline` vào đầu dòng khai báo và định nghĩa hàm

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

- Hàm inline khác gì hàm bình thường:
 - "Hàm inline" thực chất không phải là một hàm!
 - Khi gọi hàm thì lời gọi hàm được **thay thế một cách thông minh bởi mã nguồn định nghĩa hàm**, không thực hiện các thủ tục gọi hàm

Ví dụ:

```
l=max(k*5-2,1);
```

Được thay thế bằng các dòng lệnh kiểu như:

```
int x=k*5-2; // biến tạm trung gian  
l=(x>1)?x:1; // OK
```

Khi nào nên dùng hàm inline

- Ưu điểm của hàm inline:
 - Tiện dụng như hàm bình thường
 - Hiệu suất như viết thẳng mã, không gọi hàm
 - Tin cậy, an toàn hơn nhiều so với sử dụng Macro
- Nhược điểm của hàm inline:
 - Nếu gọi hàm nhiều lần trong chương trình, mã chương trình có thể lớn lên nhiều (mã thực hiện hàm xuất hiện nhiều lần trong chương trình)
 - Mã định nghĩa hàm phải để mở => đưa trong header file
- Lựa chọn xây dựng và sử dụng hàm inline khi:
 - Mã định nghĩa hàm nhỏ (một vài dòng lệnh, không chứa vòng lặp)
 - Yêu cầu về **tốc độ** đặt ra trước **dung lượng bộ nhớ**

Bài tập về nhà

- Xây dựng hàm tìm N số nguyên tố đầu tiên
 - Hoàn thiện thiết kế hàm
 - Định nghĩa hàm
- Viết chương trình minh họa cách sử dụng

Kỹ thuật lập trình

Chương 2: Các yếu tố cơ bản của C và C++



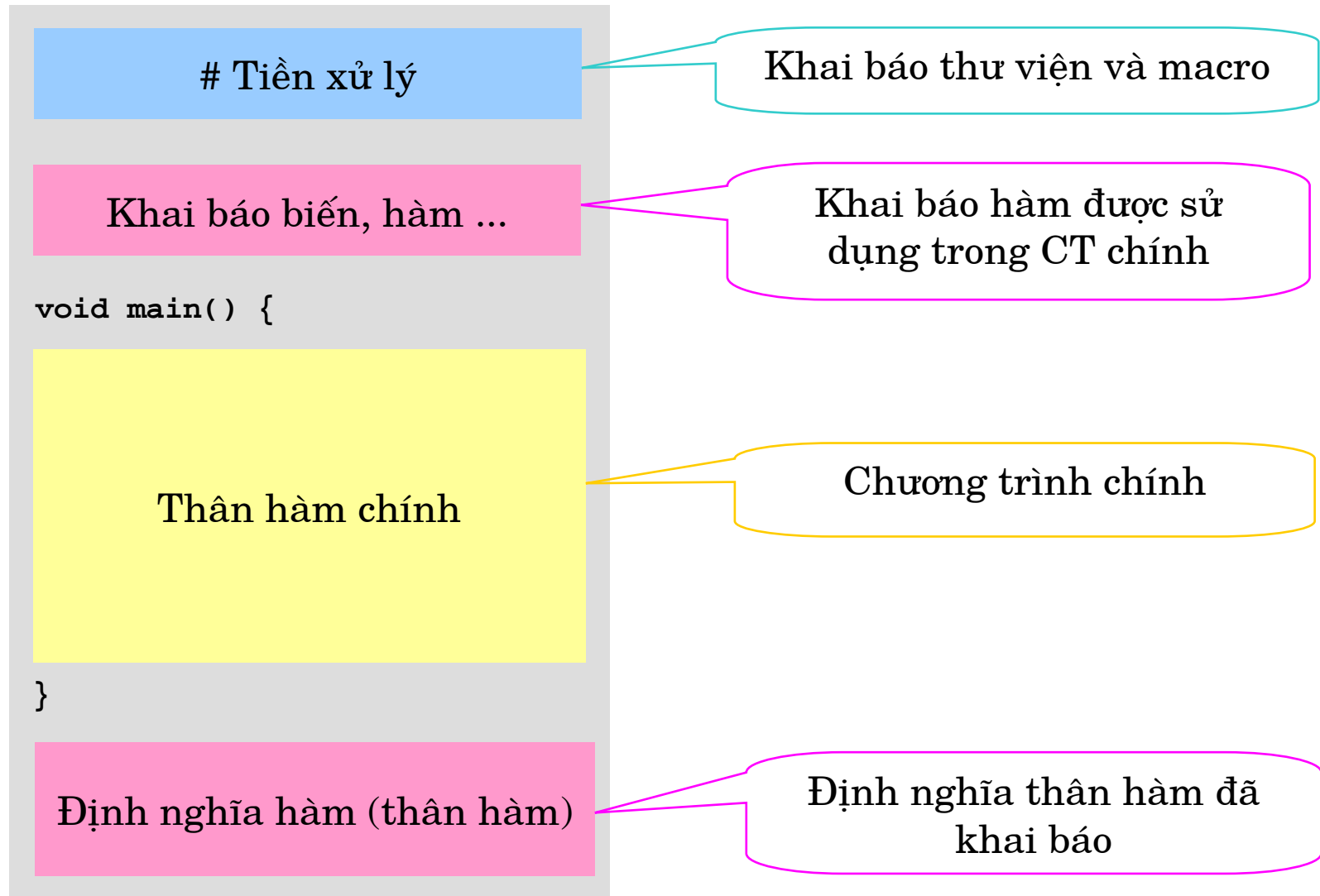
Nội dung chương 2

- 2.1 Tổ chức chương trình C/C++
- 2.2 Biến và các kiểu dữ liệu cơ bản
- 2.3 Các kiểu dữ liệu dẫn xuất trực tiếp
- 2.4 Định nghĩa kiểu dữ liệu mới
- 2.5 Điều khiển chương trình: phân nhánh
- 2.6 Điều khiển chương trình: vòng lặp
- 2.7 Một số lệnh điều khiển chương trình khác

2.1 Tổ chức chương trình C/C++

- Cấu trúc và các phần tử cơ bản của một chương trình viết trên C/C++
- Quy trình tạo ra một chương trình chạy được:
 - Vấn đề tạo dự án
 - Quy tắc soạn thảo mã nguồn
 - Biên dịch từng phần và sửa các loại lỗi biên dịch
 - Liên kết và sử dụng thư viện, sửa lỗi liên kết
 - Chạy thử và gỡ rối (Debug)
- Sơ lược về tổ chức bộ nhớ

2.1 Tổ chức chương trình C/C++



Chương trình tính giai thừa: Phiên bản C

```
#include <stdio.h>
#include <conio.h>
```

```
int factorial(int);
```

```
void main() {
```

```
    char c = 'N';
    int N = 1;
    int kq;
    do {
        printf("\nEnter a number > 0:");    /* writing on the screen */
        scanf("%d",&N);                    /* reading from keyboard to N */
        kq = factorial(N);                  /* calling function with argument N */
        printf("\nFactorial of %d is %d", N, kq); /*write result on screen */
        printf("\nPress 'Y' to continue or any other key to stop");
        c = getch();                        /* reading a character from keyboard*/
    } while (c=='y' || c=='Y');            /* checking loop condition */
```

```
    }
    int factorial(int n) {
        int kq = 1;
        while (n > 1)
            kq *= n--;
        return kq;
    }
```

Chương trình tính giai thừa: Phiên bản C++

```
#include <iostream.h>
#include <conio.h>
```

```
int factorial(int);
```

```
void main() {
```

```
    char c = 'N';
    int N = 1;
    do {
        cout << "\nEnter a number > 0:" // writing on the screen
        cin >> N; // reading from keyboard to N
        int kq = factorial(N); // calling function with argument
        cout << "\nFactorial of " << N << " is " << kq
        cout << "\nPress 'Y' to continue or any other key to stop";
        c = getch(); // reading a character from keyboard
    } while (c == 'y' || c == 'Y'); // checking loop condition
}
```

```
int factorial(int n) {
    int kq = 1;
    while (n > 1)
        kq *= n--;
    return kq;
}
```

Tạo dự án

The image shows the Microsoft Visual C++ IDE interface. The 'New' dialog is open, and the 'Win32 Console Application' option is selected in the list. The 'Project name' field is set to 'FactorialProg' and the 'Location' is 'C:\DOCS\LECTURES\PROGR'. The 'Win32 Console Application' wizard is also open, showing the first step where the user is asked to choose the type of console application to create. The 'An empty project' option is selected.

1 New... Ctrl+N

2 Win32 Console Application

3 Project name: FactorialProg
Location: C:\DOCS\LECTURES\PROGR

4 What kind of Console Application do you want to create?
 An empty project
 A simple application.
 A "Hello, World!" application.
 An application that supports MFC.

Build Debug Find in Files 1 Find in Files 2 Results SQL Deb

Creates a new document, project or workspace

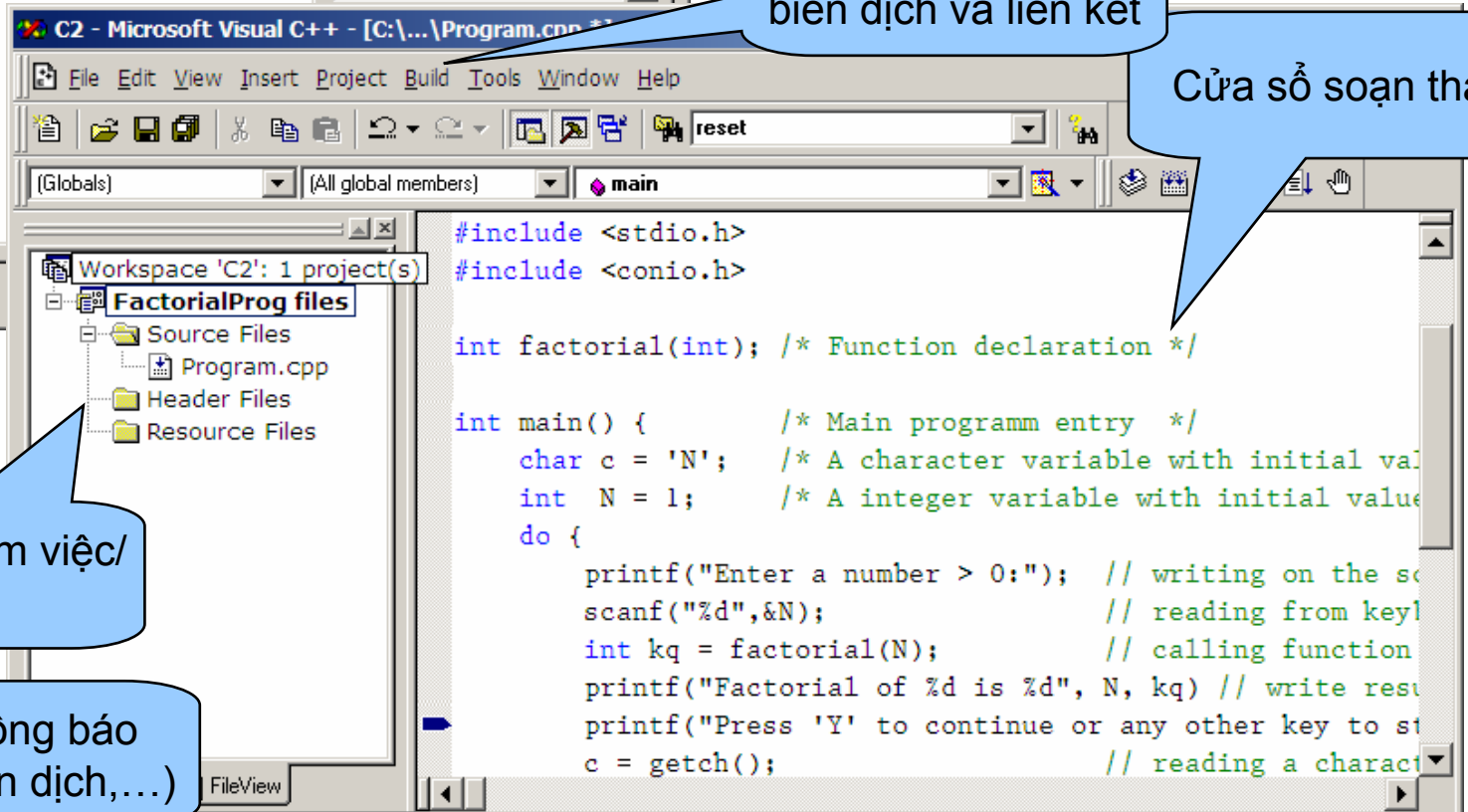
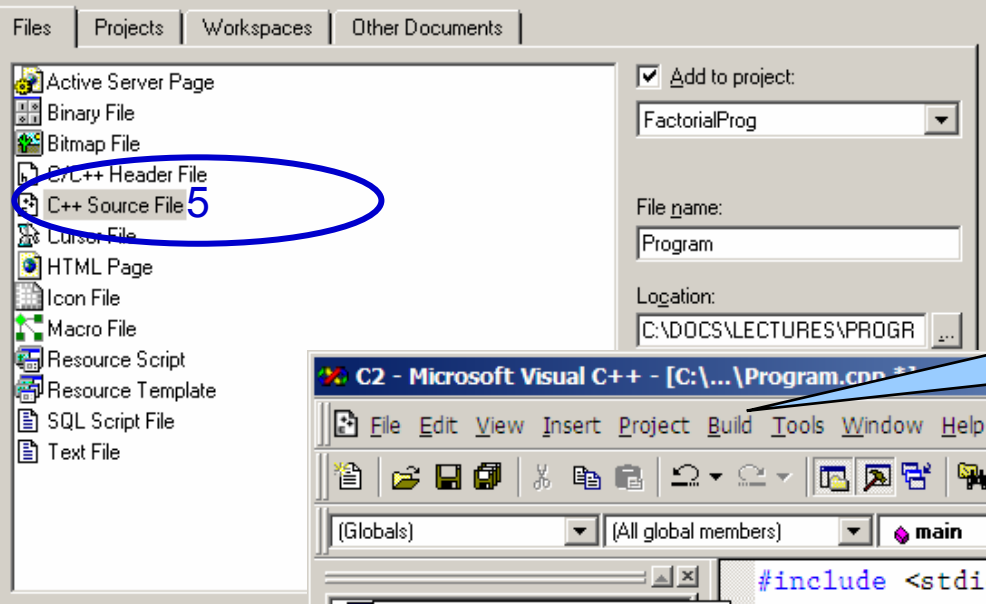
Bổ sung file mã nguồn và soạn thảo

Các công cụ biên dịch và liên kết

Cửa sổ soạn thảo

Cửa sổ bàn làm việc/ dự án

Cửa sổ thông báo kết quả (biên dịch,...)



Qui tắc soạn thảo mã nguồn

1. Tên biến, tên hàm, tên kiểu mới:
 - Tránh sử dụng các từ khóa và tên kiểu cơ sở
 - Các ký tự dùng được: 'A'..'Z', 'a'..'z', '0'..'9', '_'
 - Phân biệt giữa chữ hoa và chữ thường: **n** khác **N**
 - Ngắn nhưng đủ khả năng phân biệt, gợi nhớ để nhận biết
 - Sử dụng tiếng Anh hoặc tiếng Việt không dấu (kể cả dòng chú thích)
2. Sau mỗi câu lệnh có chấm phẩy;
3. Đoạn { ... } được coi là nhóm lệnh, không có dấu chấm phẩy sau đó, trừ trường hợp khai báo kiểu
4. Cấu trúc mã nguồn theo kiểu phân cấp => dễ đọc
5. Bổ sung chú thích đầy đủ, hợp lý (`/* ...*/` hoặc `//`)
6. Chia một file lớn thành nhiều file nhỏ

Các từ khóa trong C

<code>auto</code>	<code>double</code>	<code>int</code>	<code>struct</code>
<code>break</code>	<code>else</code>	<code>long</code>	<code>switch</code>
<code>case</code>	<code>enum</code>	<code>register</code>	<code>typedef</code>
<code>char</code>	<code>extern</code>	<code>return</code>	<code>union</code>
<code>const</code>	<code>float</code>	<code>short</code>	<code>unsigned</code>
<code>continue</code>	<code>for</code>	<code>signed</code>	<code>void</code>
<code>default</code>	<code>goto</code>	<code>sizeof</code>	<code>volatile</code>
<code>do</code>	<code>if</code>	<code>static</code>	<code>while</code>

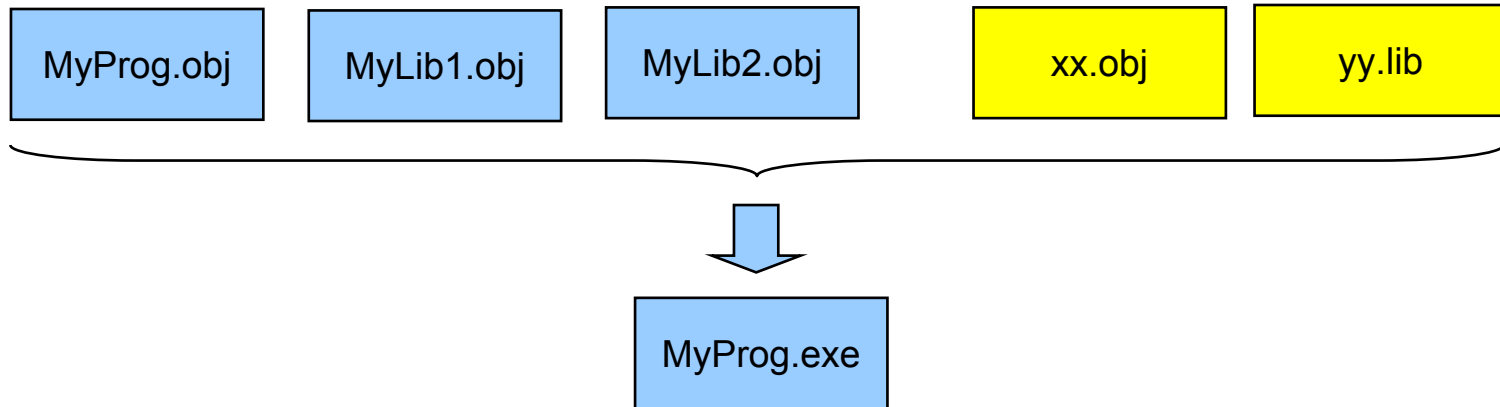
Từ khóa trong C++

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	

Biên dịch (compile)

- Biên dịch từng file nguồn riêng biệt (*.c: C compiler, *.cpp: C++ compiler), kết quả => *.obj
- Trong Visual C++: Gọi **Compile (Ctrl + F7)** để biên dịch riêng rẽ hoặc **Build (F7)** để kết hợp biên dịch và liên kết cho toàn bộ dự án
- Các kiểu lỗi biên dịch (compile error):
 - Lỗi cú pháp: Sử dụng tên sai qui định hoặc chưa khai báo, thiếu dấu chấm phẩy ;, dấu đóng }
 - Lỗi kiểu: Các số hạng trong biểu thức không tương thích kiểu, gọi hàm với tham số sai kiểu
 - ...
- Các kiểu cảnh báo biên dịch (warning):
 - Tự động chuyển đổi kiểu làm mất chính xác
 - Hàm khai báo có kiểu trả về nhưng không trả về
 - Sử dụng dấu = trong trường hợp nghi vấn là so sánh ==
 - ...

Liên kết (link)

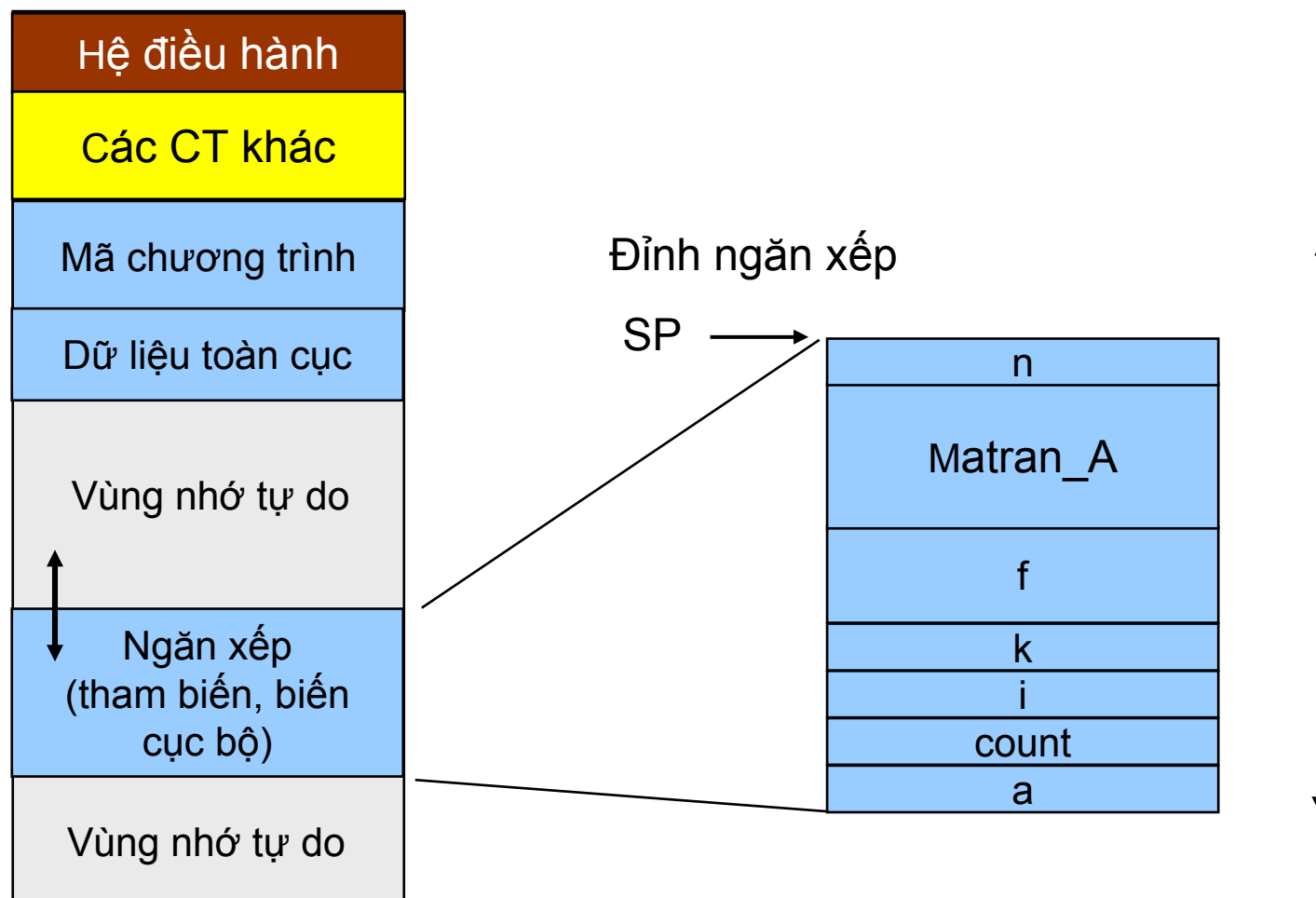


- Liên kết là quá trình ghép nhiều file đích (*.obj, *.lib) để tạo ra chương trình chạy cuối cùng *.exe
- Trong Visual C++: Gọi **Build (F7)**
- Lỗi liên kết có thể là do:
 - Sử dụng hàm nhưng không có định nghĩa hàm
 - Biến hoặc hàm được định nghĩa nhiều lần
 - ...

Chạy thử và gỡ rối (debug)

- Chạy thử trong Visual C++: **Execute** hoặc **Ctrl+F5**
- Tìm lỗi:
 - Lỗi khi chạy là lỗi thuộc về phương pháp, tư duy, thuật toán, không phải về cú pháp
 - Lỗi khi chạy bình thường không được báo
 - Lỗi khi chạy rất khó phát hiện, vì thế trong đa số trường hợp cần tiến hành **debug**.
- Chạy Debug trong Visual C++:
 - Chạy tới chỗ đặt cursor: **Ctrl+F10**
 - Chạy từng dòng lệnh: **F10**
 - Chạy vào trong hàm: **F11**
 - Chạy tiếp bình thường: **F5**
 - Xem kết quả dưới cửa sổ Output hoặc gọi QuickWatch

Tổ chức bộ nhớ



2.2 Biến và dữ liệu

- Biểu thức = dữ liệu + phép toán + ...
- Biểu diễn dữ liệu: Thông qua **biến** hoặc **hằng số**, kèm theo **kiểu**
- Nội dung trong phần này:
 - Các kiểu dữ liệu cơ bản
 - Các phép toán áp dụng
 - Tương thích và chuyển đổi kiểu
 - Khai báo biến, phân loại biến

2.2.1 Các kiểu dữ liệu cơ bản của C/C++

Kiểu	Kích cỡ thông dụng (tính bằng bit)	Phạm vi tối thiểu
<code>char</code>	8	-127 to 127
<code>signed char</code>	8	-127 .. 127
<code>unsigned char</code>	8	0 .. 255
<code>int</code>	16/32	-32767 .. 32767
<code>signed int</code>	16/32	-nt-
<code>unsigned int</code>	16/32	0 .. 65535
<code>short</code>	16	-32767 .. 32767
<code>signed short</code>	16	nt
<code>unsigned short</code>	16	0 .. 65535
<code>long</code>	32	-2147483647..2147483647
<code>signed long</code>	32	- nt-
<code>unsigned long</code>	32	0 .. 4294967295
<code>float</code>	32	Độ chính xác 6 chữ số
<code>double</code>	64	Độ chính xác 15 chữ số
<code>long double</code>	80	Độ chính xác 17 chữ số
<code>bool</code> (C++)	-	-
<code>wchar_t</code> (C++)	16	-32767 .. 32767

Các phép toán cơ bản

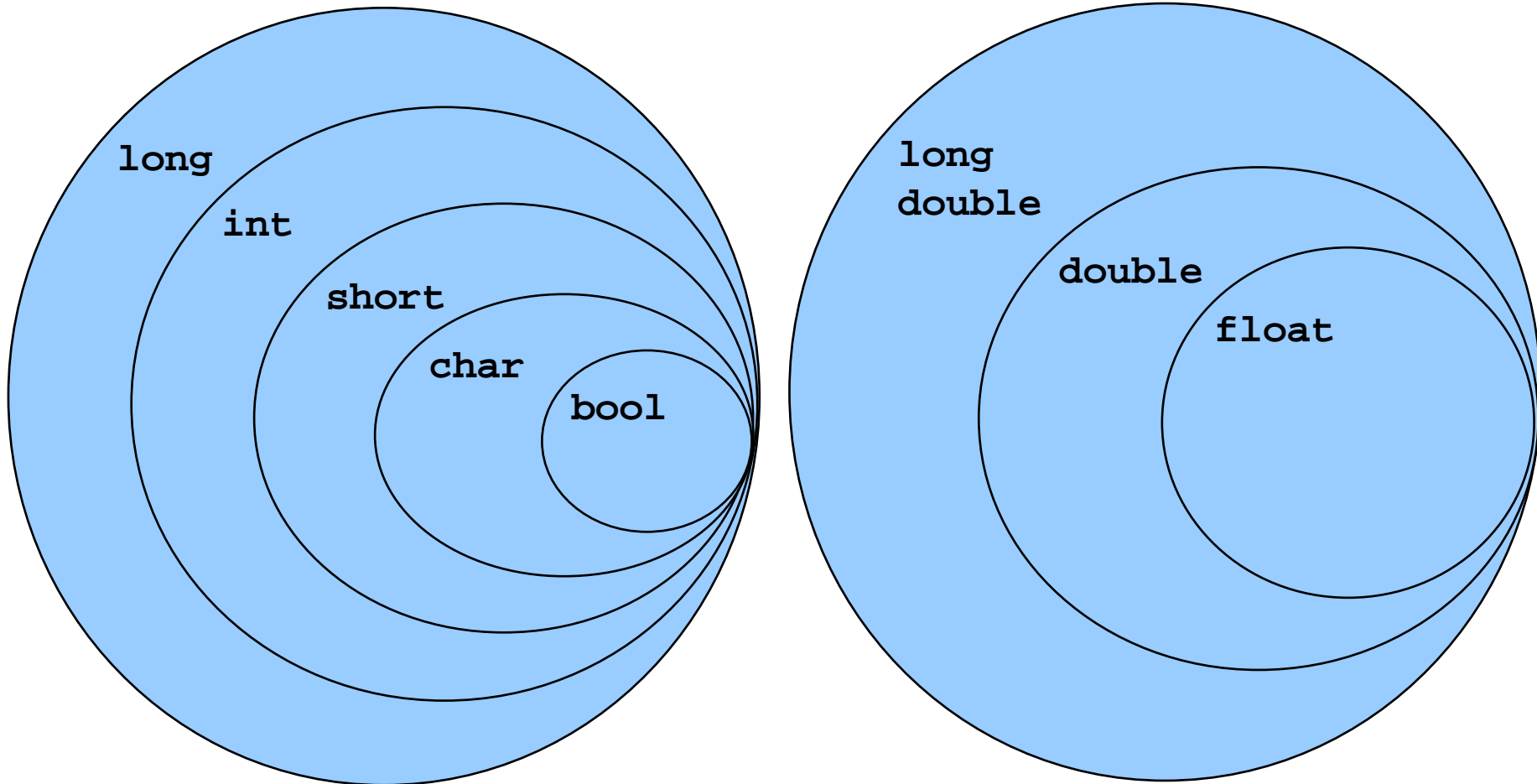
Phép toán	Ký hiệu	Kiểu nguyên	Kiểu số thực	Kiểu bool
Gán	=	X	X	X
Số học	+, -, *, /, +=, -=, *=, /=	X	X	x
	%, %=	X		x
	++, --	X		x
So sánh	>, <, >=, <=, ==, !=	X	X	X
Logic	&&, , !	X	X	X
Logic bit	&, , ^, ~ &=, =, ^=	X		x
Dịch bit	<<, >>, <<=, >>=	X		x
Lựa chọn	? :	X	X	X
Lũy thừa?	Không có!			

Tương thích và chuyển đổi kiểu

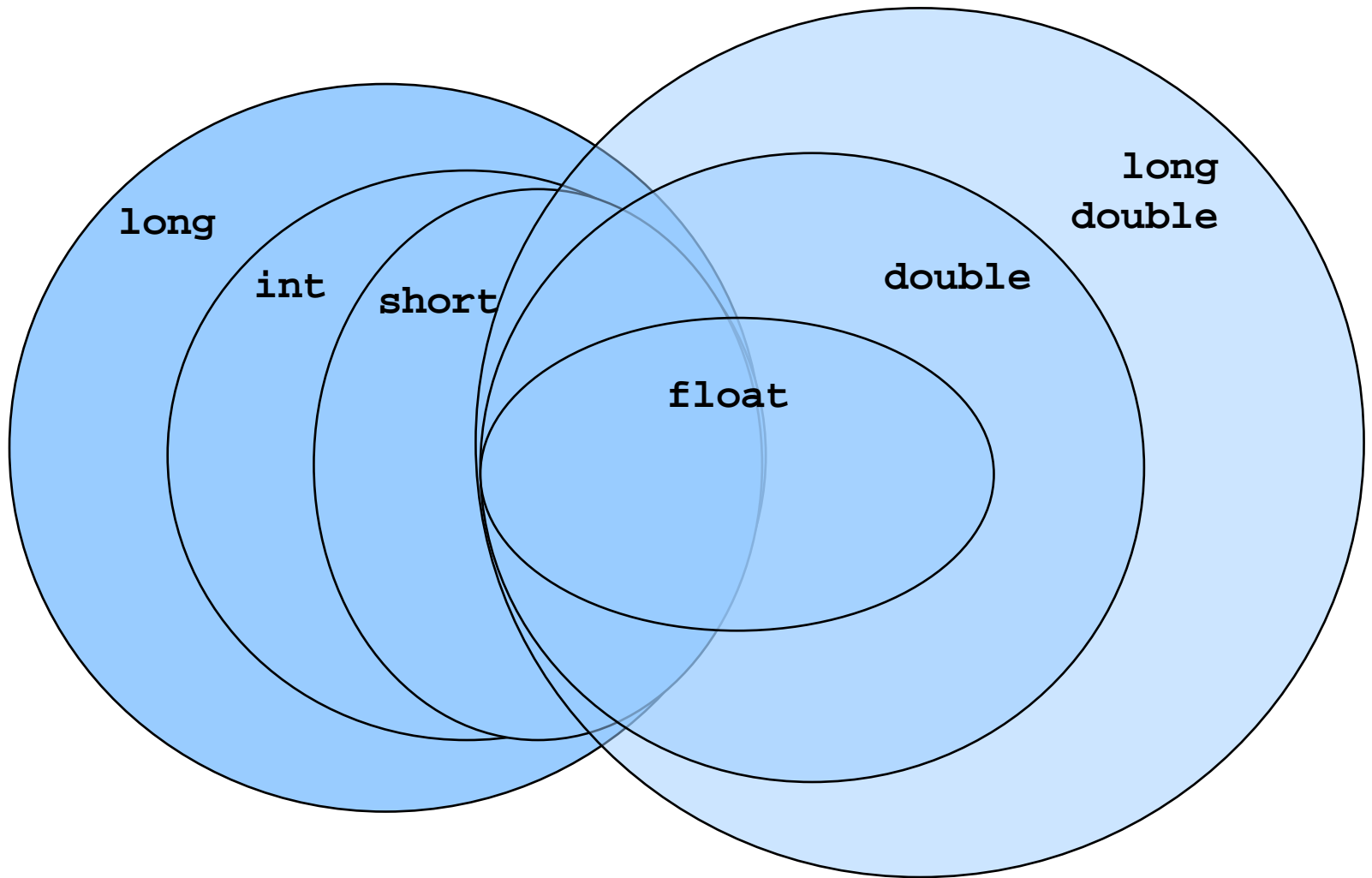
- Tương thích kiểu => Tự động chuyển đổi kiểu
 - Giữa các kiểu số nguyên với nhau (lưu ý phạm vi giá trị)
 - Giữa các kiểu số thực với nhau (lưu ý độ chính xác)
 - Giữa các kiểu số nguyên và số thực (lưu ý phạm vi giá trị và độ chính xác)
 - Kiểu bool sang số nguyên, số thực: true => 1, false => 0
 - Số nguyên, số thực sang kiểu bool: $\neq 0$ => 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++
```

Nhìn nhận về chuyển đổi kiểu



Nhìn nhận về chuyển đổi kiểu



2.2.2 Khai báo biến

`char` `c = 'N';` ← Khai báo và khởi tạo giá trị
`bool` `b = true;` ← Khai báo và khởi tạo giá trị
`int` `kq;` ← Chỉ khai báo, giá trị bất định
`double` `d;` ← Chỉ khai báo, giá trị bất định
`long` `count, i=0;` ← Khai báo kết hợp, chỉ `i=0`
`unsigned` `vhexa=0x00fa;` ← Đặt giá trị đầu hexa
`unsigned` `voctal=082;` ← Đặt giá trị đầu octal -> 66 chứ không phải 82

- C: Toàn bộ biến phải khai báo ngay đầu thân hàm
- C++: Có thể khai báo tại chỗ nào cần, trước khi sử dụng
- Phân loại biến:
 - Biến toàn cục: Khai báo ngoài hàm, lưu giữ trong vùng nhớ dữ liệu chương trình
 - Biến cục bộ: Khai báo trong thân hàm, lưu giữ trong ngăn xếp
 - Tham biến: Khai báo trên danh sách tham số của hàm, lưu giữ trong ngăn xếp

Ví dụ khai báo các loại biến

Biến toàn cục

```
int N = 1;
void main() {
    char c = 'N';
    do {
        printf("\nEnter a number > 0:");
        scanf("%d",&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;
}
```

Biến cục bộ

Hai biến cục bộ cùng tên ở hai phạm vi khác nhau, không liên quan gì đến nhau!

Tham biến

Đặc tính lưu giữ

- Biến **extern**: Khai báo sử dụng biến toàn cục đã được định nghĩa và gán giá trị trong một tập tin khác

```
/* file1.c */
int x, y;
char ch;
void main()
{
    /* ... */
}
void func1(void)
{
    x = 123;
}
```

```
/* file2.c */
extern int x, y;
extern char ch;
void func22()
{
    x = y / 10;
}
void func23()
{
    y = 10;
}
```

- Biến **static**: được lưu trữ trong bộ nhớ dữ liệu CT
 - Biến static cục bộ: hạn chế truy nhập từ bên ngoài hàm
 - Biến static toàn cục: hạn chế truy nhập từ file khác

2.2.3 Hằng số (trực kiện)

Kiểu	Ví dụ
int	1 123 21000 -234 0x0A 081
long int	35000L -341 -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
wchar_t	L'A' L'B'

2.3 Các kiểu dữ liệu dẫn xuất trực tiếp

- Kiểu liệt kê
- Kiểu hằng
- Kiểu con trỏ
- Kiểu mảng
- Kiểu tham chiếu (C++)

2.3.1 Kiểu liệt kê (enum)

- Mục đích sử dụng:
 - Định nghĩa một kiểu là tập các hằng số nguyên kí hiệu
 - 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
};
```

Sử dụng kiểu liệt kê

```
/* C version */  
void main() {  
    enum Color c = Red;           /* c = 0 */  
    enum WeekDay d = Tue;        /* d = 3 */  
    int i=c, j=d;                /* j=0, i=3 */  
    enum Color c2 = i+1;        /* c2 = 1 */  
    int di1 = 0x01;             /* OK, but... */  
    int di2 = DI_MOTOR1_STARTED; /* this is better */  
    ++c;                        /* c = 1 */  
}
```

C:
Như một kiểu số
nguyên 8 bit

```
// C++ version */  
void main() {  
    enum Color c = Red;           // c = Red  
    WeekDay d = Tue;            // OK, d = Tue  
    int i=c, j=d;              // i=0, j=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!  
}
```

C++
Không còn như
một kiểu số
nguyên!

2.3.2 Kiểu hằng (const)

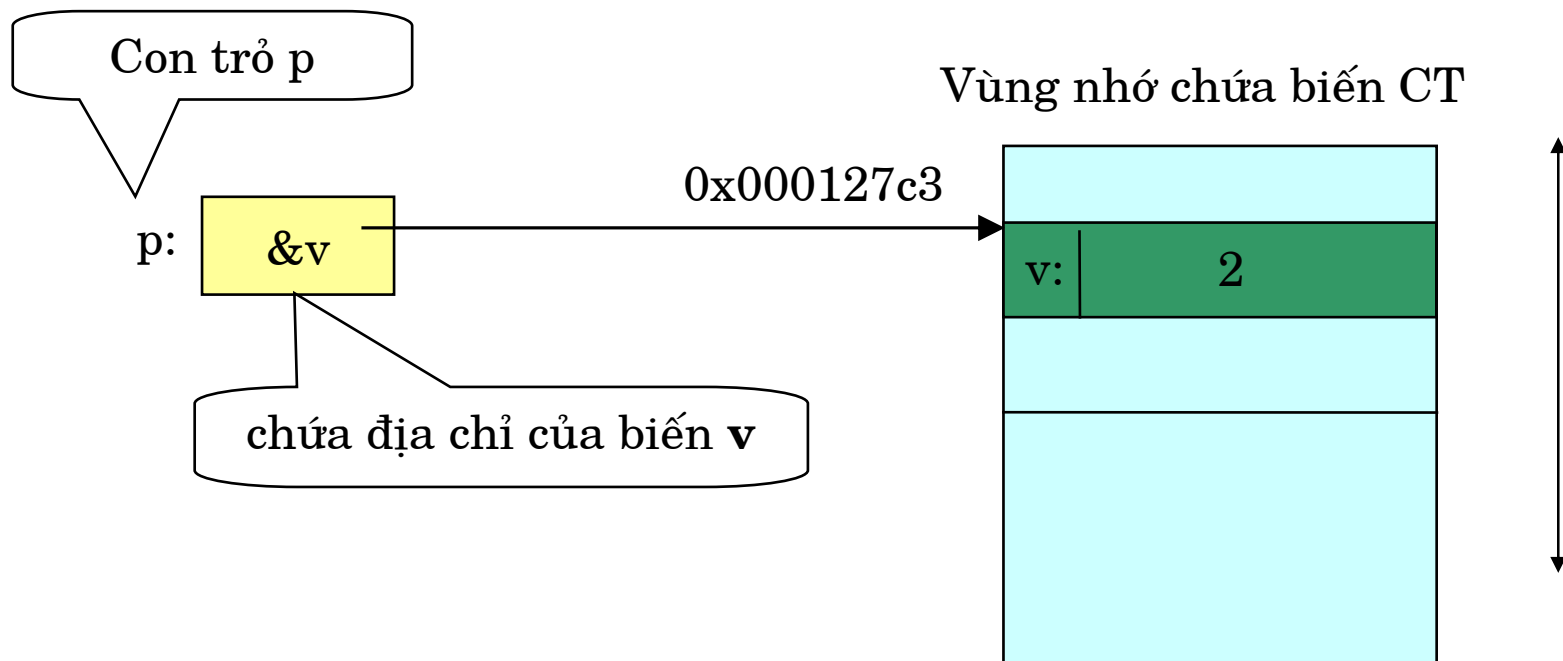
- Cho phép khai báo như biến số, nhưng được gán giá trị cố định bằng một hằng số và không thể được thay đổi => khai báo hằng số

```
void 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  
}
```

2.3.3 Kiểu con trỏ

- Con trỏ thực chất là một biến chứa địa chỉ của một đối tượng có thể là một biến hoặc một hàm.

```
int v = 2;  
int* p = &v; // p holds the address of v
```



2.3.3 Kiểu con trỏ

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

Name	Value
&i	0x0012ff7c
i	1

Name	Value
i	2
p	0x0012ff7c

Name	Value
j	3
i	2
p	0x0012ff74

Name	Value
&i	0x0012ff7c
i	1
p	0x0012ff7c
*p	1

Name	Value
j	-858993460
&j	0x0012ff74
p	0x0012ff74
*p	-858993460

Ví dụ sử dụng kiểu con trỏ

```
void main() {
    int i = 0;
    int* p = &i; // p refers to the address of i
    int j = *p; // j = 0
    *p = 2; // now i = 2
    p = &j; // now p contains the address of j
    *p = 3; // now j = 3, i remains 2
    double d = i; // OK, int is compatible to double
    p = &d; // error, int* isn't compatible to double*
    p = (*int)&d; // no compile error, but dangerous,
                // meaningles type conversion!
    double* pd=0; // p contains the address 0
    *pd = 0; // no compile error, but fatal error
    pd = &d; // OK
    double* pd2; // p refers to an uncertain address
    *pd2 = 0; // fatal error
    pd2 = &d; // OK, pd and pd2 refer to the same addr.
}
```

Tóm tắt sơ bộ về 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 (*) áp dụng cho con trỏ trả về biến mà con trỏ mang địa chỉ => 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 mà chương trình có quyền kiểm soát

Khai báo mảng

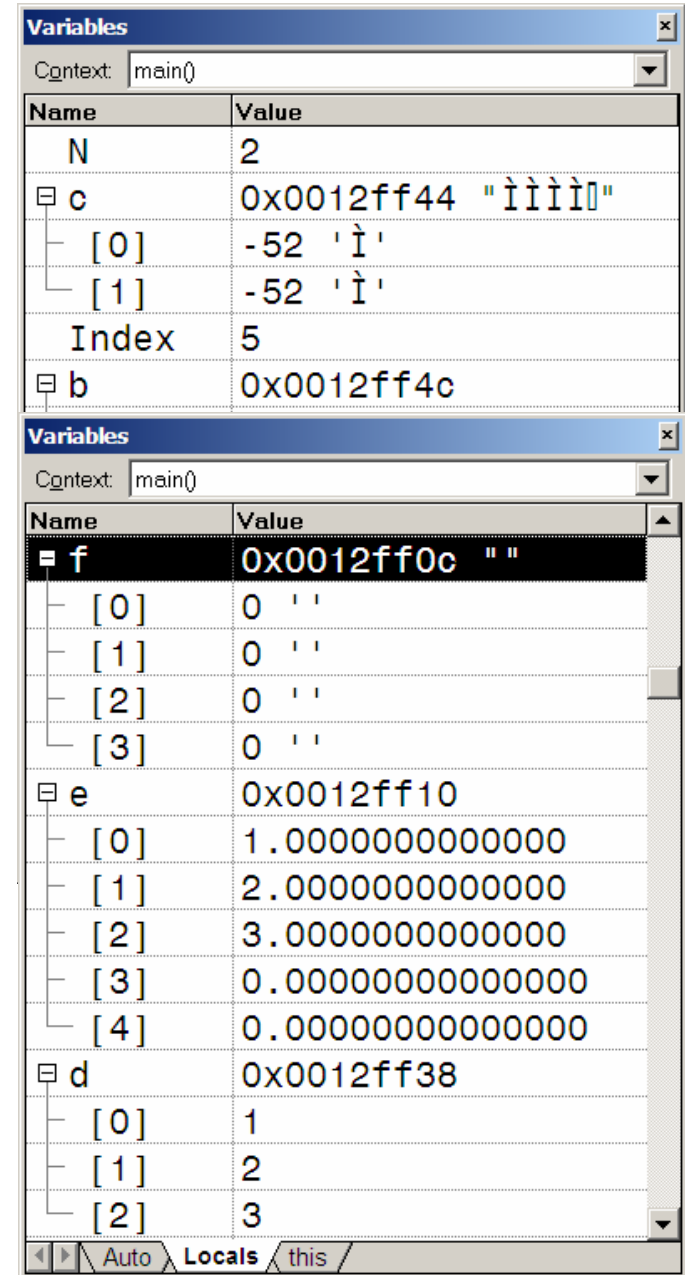
- Số phần tử của mảng phải là hằng số nguyên (trong C phải là một trực kiện, trong C++ có thể là kiểu **const** ...)

- 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};
```



Khai báo mảng (tiếp)

- 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};
```

Name	Value
M	0x0012ff50
[0]	0x0012ff50
[0]	-9.2559631349318e+061
[1]	-9.2559631349318e+061
[2]	-9.2559631349318e+061
[1]	0x0012ff68
[0]	-9.2559631349318e+061
[1]	-9.2559631349318e+061
[2]	-9.2559631349318e+061
X	0x0012ff38
[0]	0x0012ff38
[0]	1
[1]	2
[1]	0x0012ff40
[0]	3
[1]	4
[2]	0x0012ff48
[0]	5
[1]	6
T	0x0012ff30
[0]	0x0012ff30
[0]	1
[1]	2

Ví dụ sử dụng kiểu mảng

```
void main() {
    int a[5];           // a has 5 elements with uncertain values
    int b[5]= {1,3,5,7,9}; // 5 elements with initial values
    double c[];        // error, unspecified size
    double x = 1.0, y = 2.0;
    double d[]={x,y,3.0}; // 3 elements with initial values
    short n = 10;
    double v[n];       // error, array size must be a constant!
    const int m=10;    // C++ OK
    double v2[m];     // C++ OK
    a[0] = 1;
    int i= 1;
    a[i] = 2;
    a[5] = 6;         // no compile error, but fatal error
    int k = a[5];     // no compile error, but fatal error
    a = {1,2,3,4,5}; // error
    a = b;           // error, cannot assign array
    int M[2][3];
    M[0][1] = 0;
    M[0][2] = 1;
}
```

Mảng đặc biệt: Chuỗi ký tự

- Trong C/C++, chuỗi ký tự không phải là kiểu cơ bản, mà thực chất là một mảng
- Phân biệt chuỗi ký tự thường và **chuỗi ký tự kết 0**

```
char city1[] = {'H', 'A', 'N', ' ', 'O', 'I'};  
char city2[] = "HA NOI";  
wchar_t city3[] = L"HÀ NOI";  
city2[] = "HANOI"; // error
```

- Đa số các hàm trong thư viện C làm việc với chuỗi ký tự kết 0
- Với C++, chuỗi ký tự được định nghĩa bằng lớp **string** trong thư viện chuẩn, không sử dụng byte kết 0

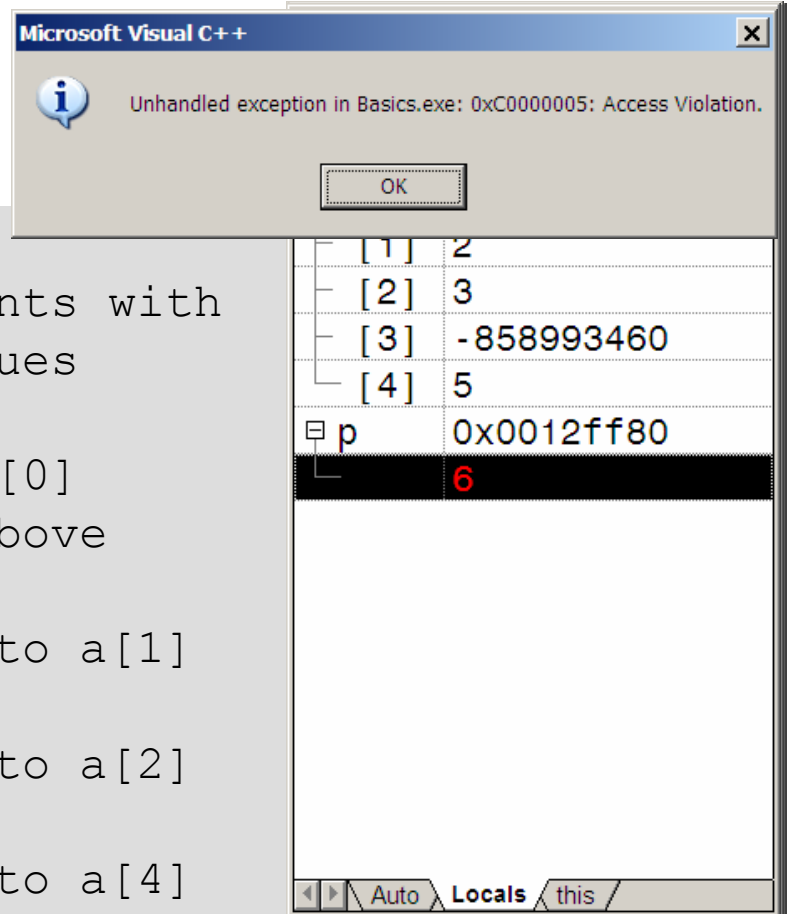
The screenshot shows a debugger's 'Variables' window with the context set to 'main()'. It displays three variables: city3, city2, and city1. Each variable is shown as an array of characters with their corresponding memory addresses and values.

Name	Value
city3	0x0012ff60
[0]	72
[1]	192
[2]	32
[3]	78
[4]	79
[5]	73
[6]	0
city2	0x0012ff70 "HA NOI"
[0]	72 'H'
[1]	65 'A'
[2]	32 ' '
[3]	78 'N'
[4]	79 'O'
[5]	73 'I'
[6]	0 ''
city1	0x0012ff78 "HAN OIììÀÿ"
[0]	72 'H'
[1]	65 'A'
[2]	78 'N'
[3]	32 ' '
[4]	79 'O'
[5]	73 'I'

Mảng và con trỏ

```
void 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; // error, a is like a constant pointer
}
```



Mảng và con trỏ (tiếp)

```
void main() {
    int a[5]; // a has 5 elements with
              // uncertain values
    int* p = a; // p points to a[0]
    p[0] = 1; // a[0]=1
    p[1] = 2; // a[1]=2
    p+= 2; // now p points to a[2]
    p[0] = 3; // a[2]=3
    p[1] = 4; // a[3]=4
    p[3] = 6; // a[5]=6, Now is a BIG BIG problem!
}
```

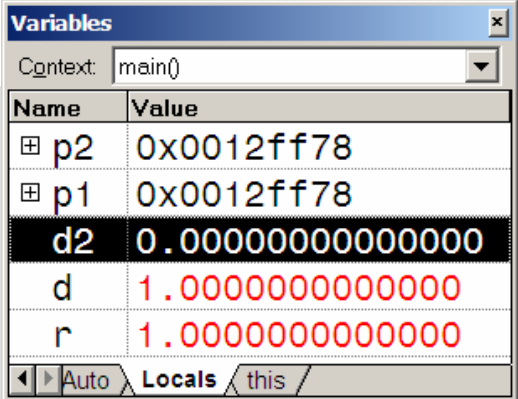
Tóm lược về mảng

- Mảng là một tập hợp các dữ liệu cùng kiểu, sắp xếp liên kề trong bộ nhớ => các phần tử của mảng
- 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 (khi khai báo phải là hằng số), không bao giờ thay đổi được
- 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 bao giờ gán được mảng cho nhau. Nếu cần sao chép hai mảng thì phải sử dụng hàm
- Không bao giờ được phép truy nhập với chỉ số nằm ngoài phạm vi, nếu N là số phần tử thì phạm vi cho phép là từ 0..N-1
- Con trỏ không bao giờ là một mảng, nó chỉ có thể mang địa chỉ của một mảng và sử dụng để quản lý mảng (dù là động hay tĩnh)

2.3.5 Kiểu tham chiếu (C++)

- Một biến tham chiếu là một biến đại diện trực tiếp cho một biến khác (thay cho con trỏ)
- Ý nghĩa sử dụng chủ yếu về sau trong truyền tham số cho hàm

```
void 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  
}
```



The screenshot shows a debugger window titled "Variables" with the context set to "main()". It displays a table of variables and their values:

Name	Value
p2	0x0012ff78
p1	0x0012ff78
d2	0.0000000000000000
d	1.0000000000000000
r	1.0000000000000000

At the bottom of the window, there are navigation buttons: "Auto", "Locals", and "this".

2.3.6 Typedef

- Từ khóa **typedef** tạo ra một tên mới cho một kiểu có sẵn, không định nghĩa một kiểu mới
- Ý nghĩa: đưa 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;
```

2.4 Định nghĩa kiểu dữ liệu mới

- Cấu trúc (**struct**): Tập hợp những dữ liệu hỗn hợp, truy nhập theo tên (biến thành viên). Thông dụng nhất trong C, ý nghĩa được mở rộng trong C++
- Hợp nhất (**union**): Một tên kiểu chung cho nhiều dữ liệu khác nhau (chiếm cùng chỗ trong bộ nhớ). Ít thông dụng trong cả C và C++
- Lớp (**class**): Chỉ có trong C++, mở rộng **struct** cũ thêm những hàm thành viên.

2.4.1 Cấu trúc (struct)

- Định nghĩa cấu trúc (bên trong hoặc ngoài các hàm)

```
struct Time  
{  
    int hour; // gio  
    int minute; // phut  
    int second; // giay  
};
```

Tên kiểu mới
(không trùng lặp)

```
struct Date {  
    int day, month, year;  
};
```

Các biến thành viên,
khai báo độc lập
hoặc chung kiểu

```
struct Student {  
    char name[32];  
struct Date birthday;  
    int id_number;  
};
```

Các biến thành viên
có thể cùng kiểu
hoặc khác kiểu

C++

Khai báo biến cấu trúc

```
void main() {  
    Time clasTime = {6,45,0};  
    Time lunchTime = {12};  
    Date myBirthday, yourBirthday = {30,4,1975};  
    Student I = {"Nguyen Van A", {2,9,1975}};  
    //...
```

The image displays four sequential screenshots of a debugger's Variables window, showing the state of variables in a C++ program. Each window has a 'Context' of 'main()'. The variables and their values are as follows:

Variable	Field	Value
clasTime	hour	6
	minute	45
	second	0
	I	{...}
lunchTime	hour	12
	minute	0
	second	0
	I	{...}
yourBirthday	day	-858993
	month	-858993
	year	-858993
	I	{...}
I	name	0x0012ff20 "Nguyen Van A"
	birthday	{...}
	day	2
	month	9

Sử dụng biến cấu trúc

```
/. . .
void main() {
    Time clasTime = {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 = clasTime.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
}
```

Phản ví dụ: khai báo và sử dụng cấu trúc

```
struct Time {
    int hour = 0;        // error, initialization not allowed
    int minute,         // error, use semicolon (;) instead
    int second          // error, missing semicolon (;)
} // error, missing semicolon (;)
//...
void main() {
    Date d;
    d = {11,9,2001}; // error, {...} is an initialization
                  // list, not a structure
    Date.hour = 0; // error, Date is a type, not a var.
    struct Date2 { int day, month, year; };
    Date2 d2 = d; // error, Date is not compatible to Date2
}
```

Mảng, con trỏ và cấu trúc

- Kết hợp mảng, con trỏ và cấu trúc cho phép xây dựng và sử dụng các cấu trúc dữ liệu phức tạp một cách rất linh hoạt

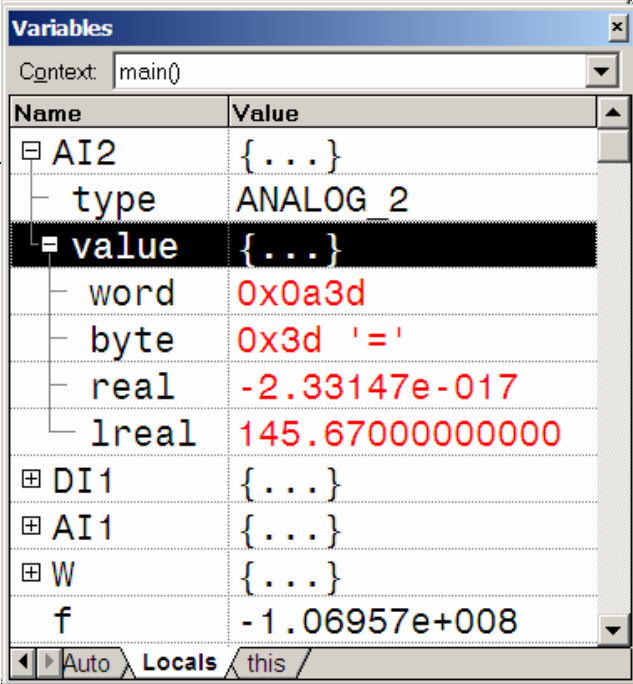
```
void main() {
//...
    Date victoryDays[]= {{19,8,1945},{7,5,1954},{30,4,1975}};
    Date saigonVictory= victoryDays[2];
    Date *p=&saigonVictory;
    (*p).year += 30;      // good
    p->year -=30;        // better
    Student studentList[45];
    for (int i=0; i < 45; ++i) {
        studentList[i].id_number= i;
        studentList[i].birthday=yourBirthday;
    }
    Student* pList = studentList;
    while (pList < studentList+45) {
        pList->id_number += 4800;
        ++pList;
    }
}
```


Tóm lược về cấu trúc (struct)

- Cấu trúc (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 kiểu cấu trúc bằng cách khai báo **tên các biến thành viên**. Định nghĩa kiểu cấu trúc chưa phải là định nghĩa các biến cụ thể, vì thế không được đặt giá trị đầu cho các biến
- Kích cỡ của cấu trúc \geq tổng kích cỡ các thành viên
- Truy cập một biến cấu trúc thông qua tên biến, toán tử (.) và tên biến thành viên
- Các kiểu cấu trúc có thể lồng vào nhau, trong cấu trúc có thể sử dụng mảng, một mảng có thể có các phần tử là cấu trúc, v.v...
- Các biến có cùng kiểu cấu trúc 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 cấu trúc thông qua toán tử (*) và toán tử (->)
- Hai kiểu cấu trúc có khai báo giống nhau hoàn toàn vẫn là hai kiểu cấu trúc khác nhau

2.4.2 Hợp nhất

```
enum SignalType {BINARY_8, BINARY_16, ANALOG_1, ANALOG_2};
union SignalValue {
    unsigned short word;
    unsigned char byte;
    float real;
    double lreal;
};
struct Signal {
    SignalType type;
    SignalValue value;
};
void main() {
    SignalValue B,W;
    B.byte = 0x01;
    W.word = 0x0101;
    unsigned char b = W.byte; // OK, the lower byte
    float f = W.real; // meaningless
    Signal DI1 = {BINARY_8, 0x11};
    Signal AI1 = {ANALOG_1, {0}};
    Signal AI2;
    AI2.type = ANALOG_2;
    AI2.value.lreal = 145.67;
}
```



The screenshot shows a debugger window titled "Variables" with the context set to "main()". It displays a list of variables and their values:

Name	Value
AI2	{...}
type	ANALOG_2
value	{...}
word	0x0a3d
byte	0x3d '='
real	-2.33147e-017
lreal	145.670000000000
DI1	{...}
AI1	{...}
W	{...}
f	-1.06957e+008

Tóm lược về hợp nhất

- Hợp nhất (union) là một tập hợp (không có cấu trúc chặt chẽ) chứa các biến sử dụng chung ô nhớ, ở mỗi ngữ cảnh chỉ sử dụng một biến riêng biệt
- Union thường được sử dụng khi dữ liệu đầu vào có thể có kiểu khác nhau
- Các thành viên của một union không liên quan đến nhau, không cùng nhau tạo thành một thực thể thống nhất
- Kích cỡ của union bằng kích cỡ của biến lớn nhất
- Khai báo kiểu union tương tự như khai báo struct, nhưng ý nghĩa khác hẳn
- Truy nhập biến thành viên cũng tương tự như struct, có thể qua biến trực tiếp hoặc qua biến con trỏ.
- Union có thể chứa struct, struct có thể chứa union, union có thể chứa mảng, các phần tử của mảng có thể là union.

2.5 Điều khiển CT: 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!**)

2.5.1 Cấu trúc `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: Hàm max()

```
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;  
}
```

2.5.2 Cấu trúc switch .. case

```
Signal input;
int i = 0;
while (i++ < 8) {
    input = readInput(i); // read from input module i
    switch (input.type) {
        case BINARY_8:
            cout << input.value.byte; break;
        case BINARY_16:
            cout << input.value.word; break;
        case ANALOG_1:
            cout << input.value.real; break;
        case ANALOG_2:
            cout << input.value.lreal; break;
        default:
            cout << "Unknown signal type";
    }
}
```

2.6 Điều khiển CT: 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**, song không bao giờ nên như vậy
- Ứng dụng 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)

2.6.1 Cấu trúc while..

```
#include <iostream.h>
void main() {
    char input[32];
    cout << "\nEnter your full name:";
    cin.getline(input,31);
    short nLetters=0, nSpaces=0;
    short i=0;
    while (input[i] != 0) {
        if (input[i] == ' ')
            ++nSpaces;
        else
            ++nLetters;
        ++i;
    }
    cout << "\nYour name has " << nLetters << " letters";
    cout << "\nYou have " << nSpaces - 1 << " middle name";
    cin >> i;
}
```

Cấu trúc while: Biểu thức điều kiện

```
#include <iostream.h>
void main() {
    char input[32], family_name[16]={0};
    cout << "\nEnter your full name:";
    cin.getline(input,31);
    short i=0;
    while (input[i] != 0) {
        if (input[i] == ' ') break;
        family_name[i]= input[i];
        ++i;
    }
    cout << "\nYour family name is " << family_name;
    cin >> i;
}
```

2.6.2 Cấu trúc do while...

```
#include <iostream.h>
void main() {
    char input[32], family_name[16]={0};
    short i;
    do {
        cout << "\nEnter your full name:";
        cin.getline(input,31);
        i = 0;
        while (input[i] != 0 && input[i] != ' ') {
            family_name[i]= input[i];
            ++i;
        }
        cout << "\nYour family name is " << family_name;
        cout << "\nDo you want to continue? (Y/N):";
        cin >> i;
    } while (i == 'Y' || i == 'N')
}
```

2.6.3 Cấu trúc for ..

```
short i =0;
while (input[i] != 0)
{
    if (input[i]==' ')
        ++nSpaces;
    else
        ++nLetters;
    ++i;
}
```

```
for (short i=0;input[i]!=0; ++i)
{
    if (input[i] == ' ')
        ++nSpaces;
    else
        ++nLetters;
}
```

```
short i=0;
for (;input[i] != 0;)
{
    if (input[i]==' ')
        ++nSpaces;
    else
        ++nLetters;
    ++i;
}
```

```
short i=0;
for (;input[i] !=0; ++i)
{
    if (input[i] == ' ')
        ++nSpaces;
    else
        ++nLetters;
}
```

Tóm lược các cấu trúc 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 tương đối tự do, 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, ...)

Luyện tập ở nhà theo sườn bài giảng

- Tập tạo dự án mới với Visual C++
- Tập viết một chương trình bằng C (đặt đuôi *.c):
 - tập khai báo các loại biến, sử dụng các kiểu dữ liệu cơ bản
 - tập sử dụng các phép toán đã học
 - sử dụng toán tử **sizeof** để tìm kích cỡ các kiểu dữ liệu, in kết quả ra màn hình
 - biên dịch, chạy thử và tìm lỗi
 - tập sử dụng công cụ debugger
 - đổi đuôi file thành *.cpp và thử lại
- Tập viết một chương trình bằng C/C++ khác để tìm hiểu:
 - Cách khai báo và sử dụng kiểu hằng, kiểu liệt kê, kiểu con trỏ, kiểu mảng, kiểu tham chiếu (C++), kiểu cấu trúc
 - bản chất của con trỏ và quan hệ với kiểu mảng

Bài tập về nhà cho chương 2

1. Viết một chương trình bằng C, thực hiện lần lượt các chức năng sau đây:
 - yêu cầu người sử dụng nhập một số nguyên lớn hơn 0
 - phân tích số nguyên đó thành hàng đơn vị, hàng chục, hàng trăm, v.v... và in kết quả lần lượt ra màn hình.
 - hỏi người sử dụng có yêu cầu tiếp tục hay không, nếu có yêu cầu thì lặp lại
2. Chuyển chương trình thành C++ và đơn giản hóa các câu lệnh vào-ra bằng cách sử dụng thư viện `<iostream.h>`
3. Dựa vào kiểu Date trong bài giảng, viết một chương trình cho phép người sử dụng nhập số liệu cho một ngày, và sau đó:
 - a) Kiểm tra các số liệu ngày, tháng và năm có hợp lệ hay không
 - b) Kiểm tra xem ngày đó có phải là một ngày lễ trong năm hay không
 - c) Xác định ngày tiếp theo là ngày nào
 - d) In các kết quả thông báo ra màn hình

Bài tập lớn 1 (tuần 1-6: Lập trình cấu trúc)

1. Xây dựng một chương trình có chức năng tạo tín hiệu theo yêu cầu người sử dụng về dạng tín hiệu (bậc thang, tín hiệu dốc, xung vuông, hình sin hoặc ồn trắng), tham số của tín hiệu (tùy theo dạng tín hiệu chọn như biên độ, tần số, độ dốc, độ rộng xung,...). Yêu cầu người sử dụng nhập khoảng thời gian cần tạo giá trị tín hiệu cùng thời gian trích mẫu, sau đó ghi các giá trị gián đoạn của tín hiệu ra một file với tên do người sử dụng nhập.

Gợi ý: sử dụng thư viện `<fstream.h>` cho việc thao tác với file.

2. Xây dựng một chương trình để tính tích phân của tín hiệu (hay tính diện tích dưới đường cong) bằng phương pháp xấp xỉ hình thang với các giá trị gián đoạn của tín hiệu đưa vào từ file tạo ra theo chương trình 1.
3. Suy nghĩ phân hoạch chương trình 1 và 2 thành các hàm đưa vào thư viện. Viết lại các chương trình đó theo thiết kế mới.

Chỉ dẫn về thực hiện bài tập lớn

- Bài tập lớn có thể thực hiện riêng hoặc theo nhóm tự chọn (tối đa 3 người/nhóm)
- Bài tập lớn 1 nộp vào cuối tuần 7, bao gồm:
 - Mô tả theo mẫu trên ít nhất 1 trang giấy về các tư tưởng phân tích, thiết kế và thực thi.
 - Toàn bộ thư mục dự án (file dự án, mã nguồn và chương trình chạy) cần nén lại dưới dạng *.zip và gửi về địa chỉ email của giáo viên: luuhongviet@gmail.com. Qui định tên file zip: bắt đầu bằng “P1_”, tiếp theo là tên đầy đủ của người đại diện nhóm, ví dụ “P1_NguyenVanA.zip”. Lưu ý trước khi nén cần xóa tất cả các file phụ trong thư mục “Debug”, chỉ trừ file *.exe.
- Hoàn thành bài tập lớn không những là điều kiện dự thi học kỳ, mà điểm bài tập lớn còn được tính vào điểm cuối học kỳ theo một hệ số thích hợp