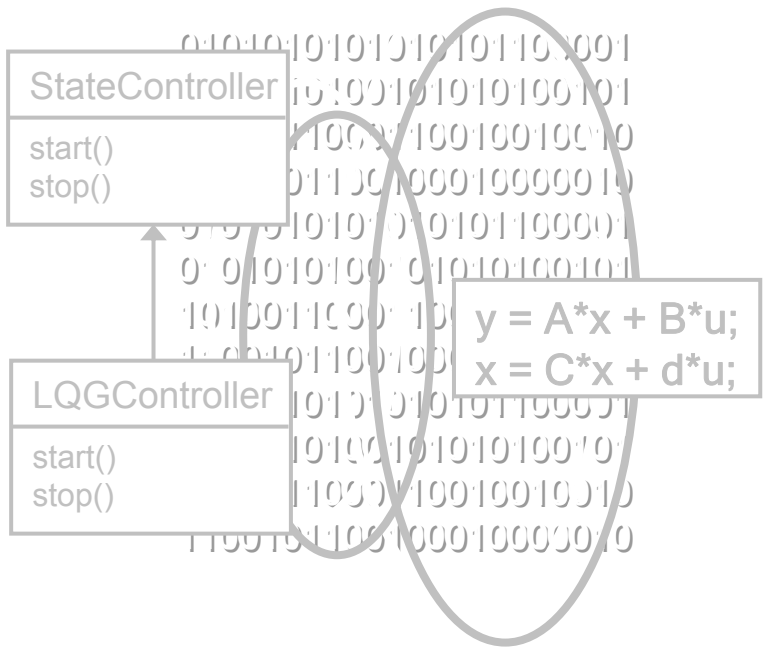
Kỹ thuật lập trình

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 Qui 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 hoạ 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 **tư 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 vặn 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 khắc 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 (µC, 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 Funtion 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;
...
```

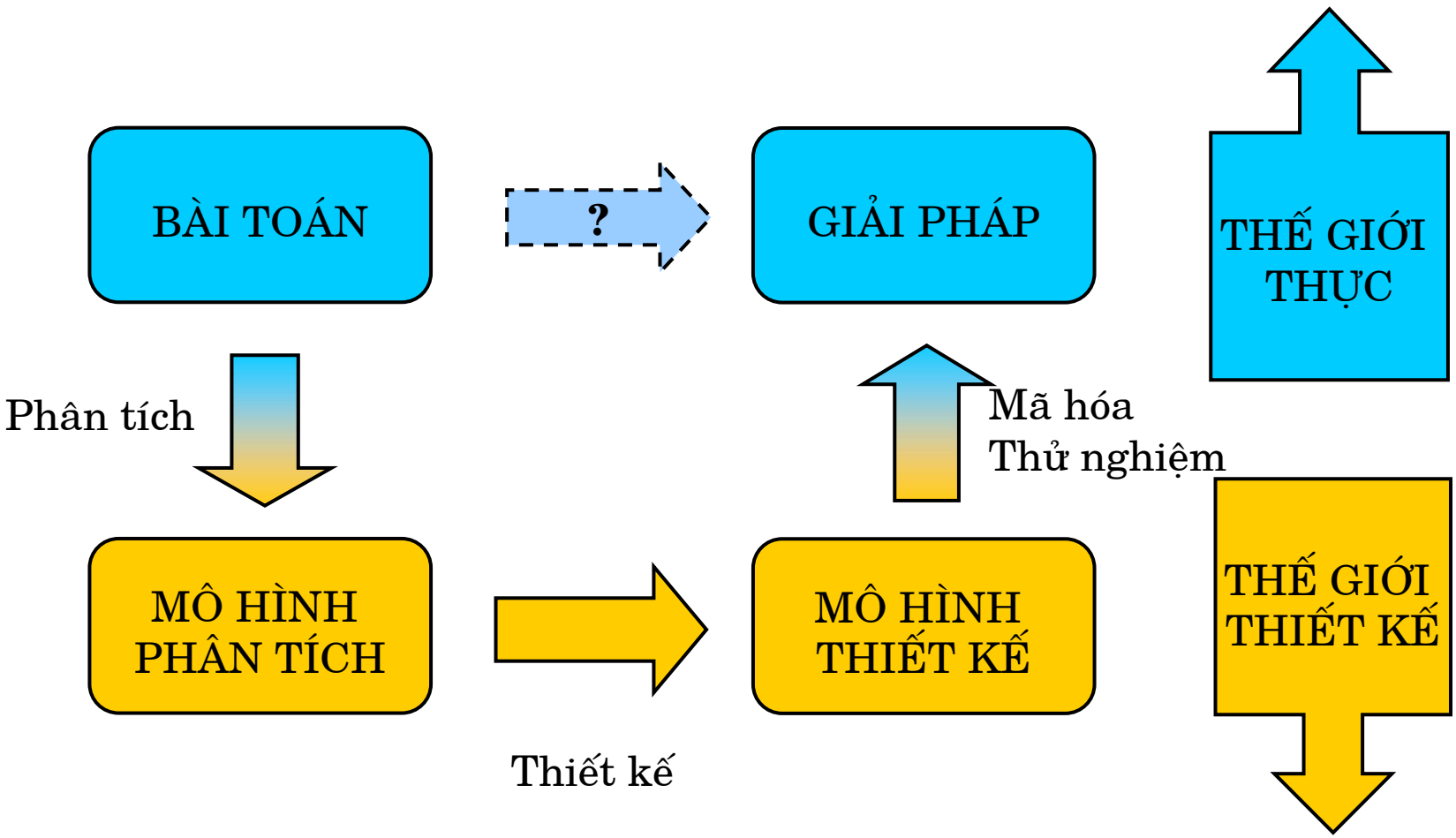
Lập trình thành phần (component-based programming)

- 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, HMTL, 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 Qui 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)

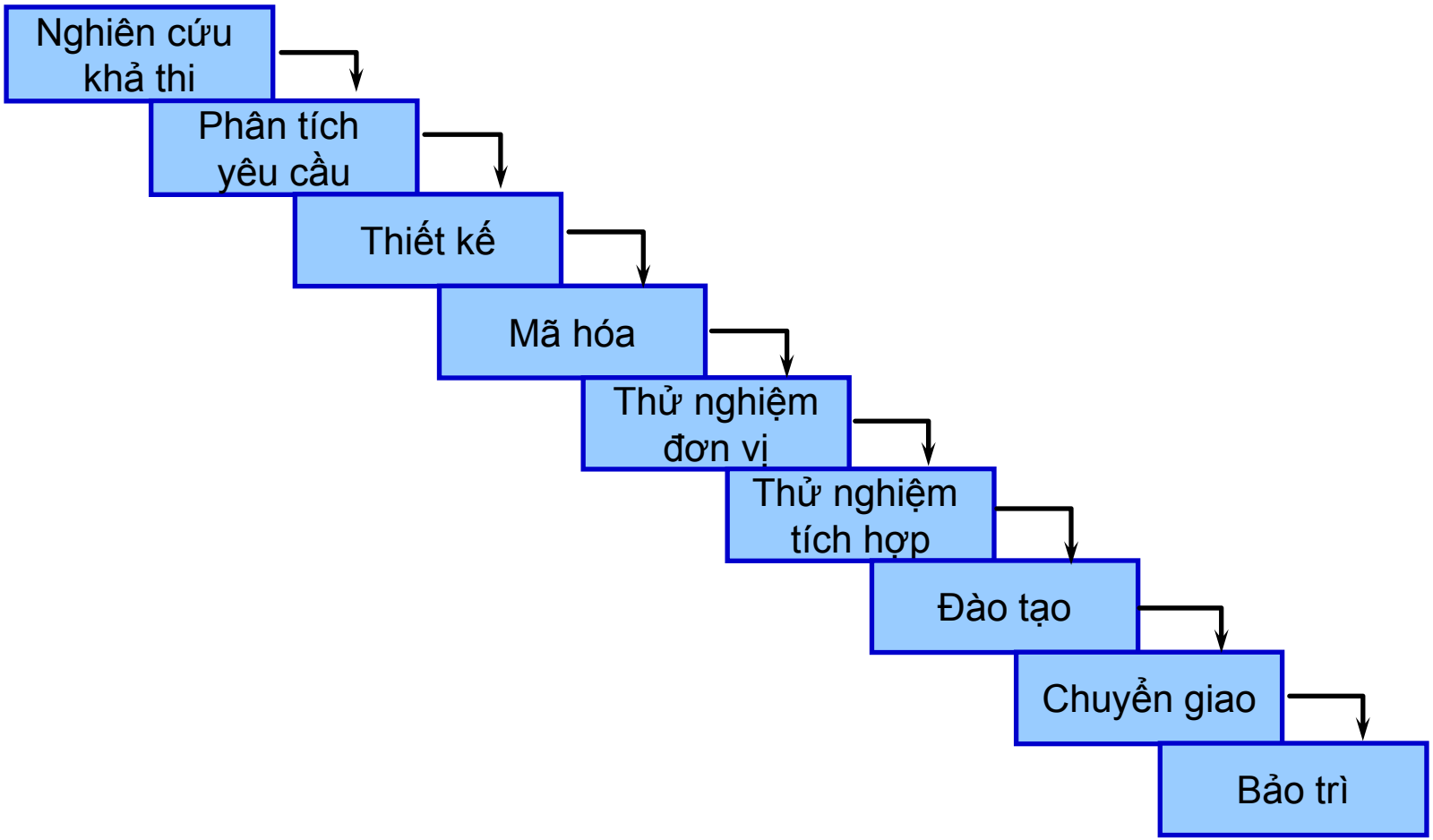
- Dựa trên mô hình hệ thống, xây dựng các mô hình chi tiết phục vụ sẵn sàng mã hóa/cài đặt
- Bao gồm:
  - Thiết kế cấu trúc (*structured design*): chương trình, kiểu dữ liệu, đối tượng, quan hệ cấu trúc giữa các đối tượng và kiểu)
  - Thiết kế tương tác (*interaction design*): quan hệ tương tác giữa các đối tượng
  - Thiết kế hành vi (*behaviour design*): sự kiện, trạng thái, phép toán, phản ứng
  - Thiết kế chức năng (*funtional 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?**

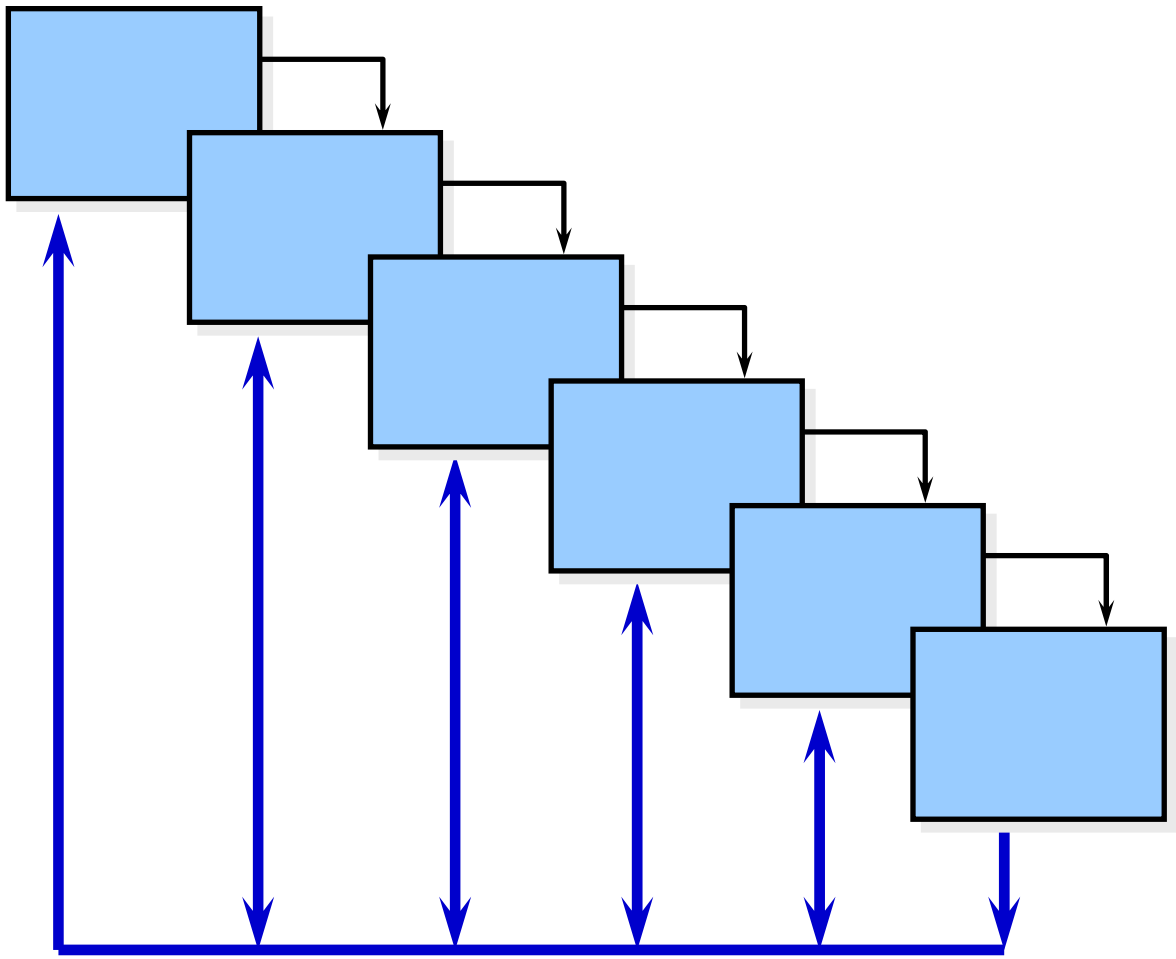
Các bước khác

- 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"



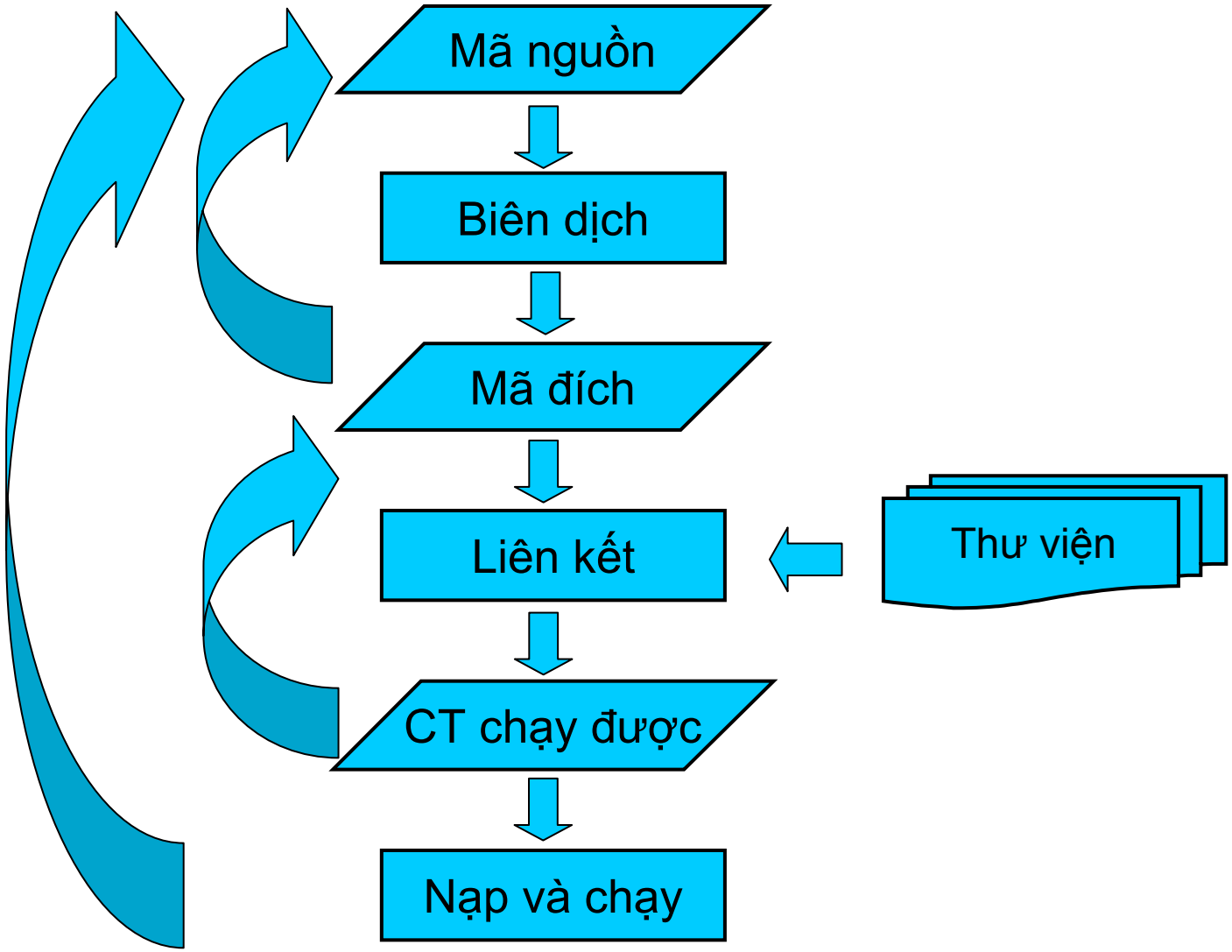
Xu thế hiện nay: Song song và lặp



Lập trình là gì, nằm ở đâu?

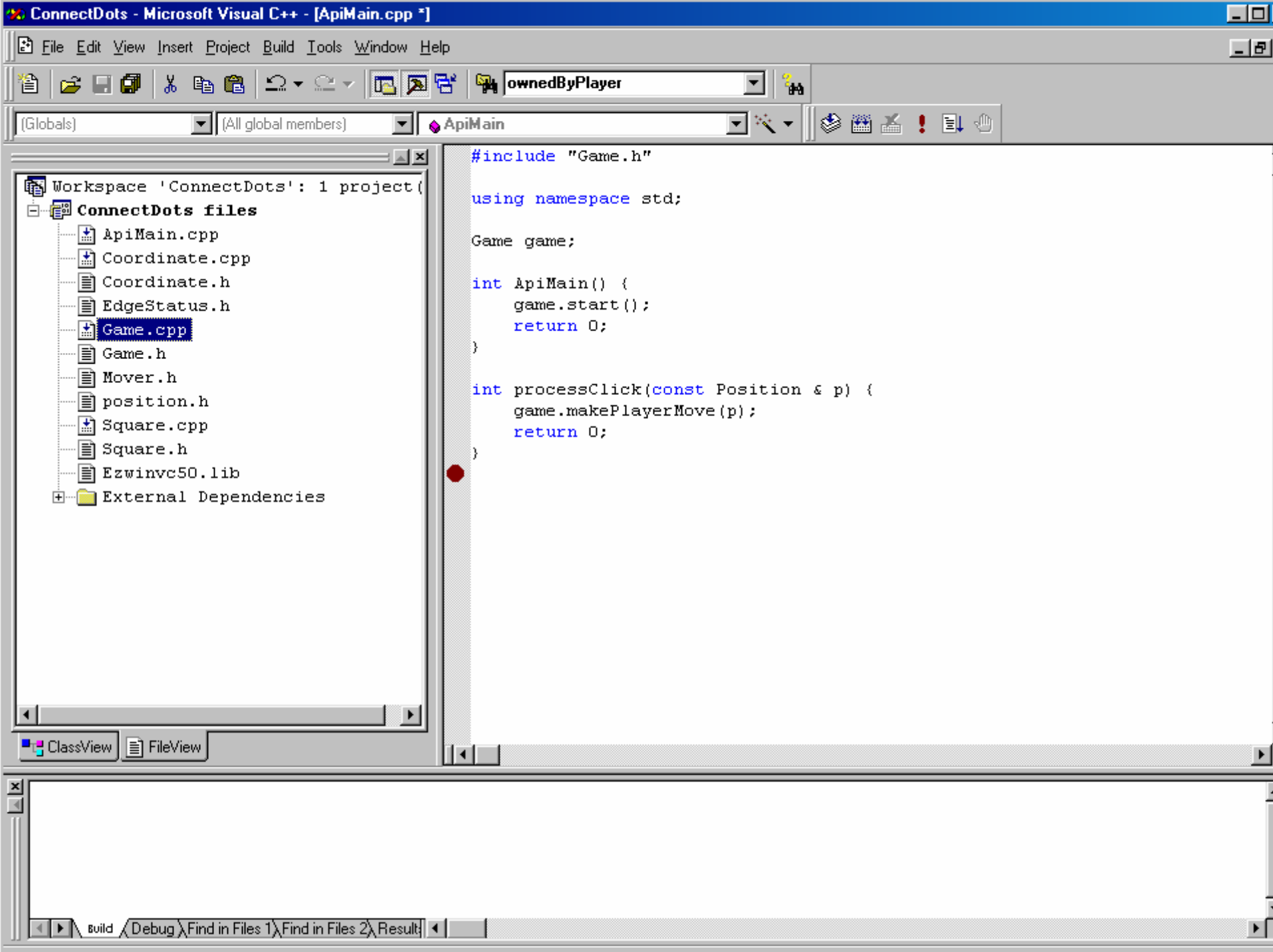
- Lập trình > Mã hóa
- Lập trình ≈ Tư tưởng thiết kế + Mã hóa + Thử nghiệm + Gỡ rối

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)


ConnectDots - Microsoft Visual C++ - [ApiMain.cpp *]
File Edit View Insert Project Build Tools Window Help
ownedByPlayer
(Globals)
(All global members)
ApiMain
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
ClassView
FileView
#include "Game.h"
using namespace std;
Game game;
int ApiMain() {
game.start();
return 0;
}
int processClick(const Position & p) {
game.makePlayerMove(p);
return 0;
}
Build
Debug
Find in Files 1
Find in Files 2
Results

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