

Kỹ Thuật Lập Trình

(Ngôn Ngữ Lập Trình C)

Bài giảng số 1

Giới thiệu về môn học

GV. Nguyễn Trí Cường

C9-101, email: cuong.nguyentri@hust.edu.vn

Sđt: 0983309963

- Teams, bảo mật
- Giờ: trc 7h: 9:30
- Giữ im lặng
 - Vi phạm: mời ra ngoài + ghi tên trừ 1 điểm/lần
- Ko điểm danh
- 2-1-0-4:
 - 2: lý thuyết + chữa BT.
 - 1: chữa BT + BTVN (có tính điểm)
- Thi: giữa kỳ (có điểm công và trừ) + cuối kỳ (chỉ tính bài thi)
 - Chỉ thi trong nội dung học và BT

Mã máy

- ❑ Máy tính chỉ nhận các tín hiệu điện tử - có, không có - tương ứng với các dòng bits.
- ❑ 1 program ở dạng đó gọi là machine code.
- ❑ Ban đầu chúng ta phải dùng machine code để viết CT:
- ❑ Quá phức tạp, giải quyết các bài toán lớn là không tưởng

```
23fc 0000 0001 0000 0040
0cb9 0000 000a 0000 0040
6e0c
06b9 0000 0001 0000 0040
60e8
```

ASSEMBLY LANGUAGE

- NN Assembly là bước đầu tiên của việc xây dựng cơ chế viết chương trình tiện lợi hơn – thông qua các ký hiệu, từ khóa và cả mã máy.
- Tất nhiên, để chạy được các chương trình này thì phải dịch (assembled) thành machine code.
- Vẫn còn phức tạp, cải thiện không đáng kể

```
movl    #0x1,n
compare:
    cmpl    #0xa,n
    cgt     end_of_loop
    acddl    #0x1,n
    bra     compare
end_of_loop:
```

Ngôn ngữ lập trình cấp cao

- Thay vì dựa trên phần cứng (machine-oriented) cần tìm cơ chế dựa trên vấn đề (problem-oriented) để tạo chương trình.
- Chính vì thế *high(er) level* languages – là các ngôn ngữ lập trình gần với ngôn ngữ con người hơn – dùng các từ khóa tiếng anh – đã được xây dựng như : Algol, Fortran, Pascal, Basic, Ada, C, ...

Chương trình máy tính

Những tính chất cần có với các chương trình phần mềm

- Tính mềm dẻo scalability / Khả năng chỉnh sửa modifiability
- Khả năng tích hợp integrability / Khả năng tái sử dụng reusability
- Tính chuyển đổi, linh hoạt, độc lập phần cứng -portability
- Hiệu năng cao -performance
- Độ tin cậy - reliability
- Dễ xây dựng
- Rõ ràng, dễ hiểu
- Ngắn gọn, xúc tích

Lịch sử ra đời

- 1940s : Machine code
- 1950s Khai thác sức mạnh của MT: Assembler code, Autocodes, first version of Fortran
- 1960s Tăng khả năng tính toán: Cobol, Lisp, Algol 60, Basic, PL/1 --- nhưng vẫn dùng phong cách lập trình cơ bản của assembly language.
- 1970s Bắt đầu cuộc khủng hoảng phần mềm “software crisis”:
 1. Giảm sự phụ thuộc vào máy – Tính chuyển đổi.
 2. Tăng sự đúng đắn của chương trình -Structured Programming, modular programming và information hiding.Ví dụ : Pascal, Algol 68 and C.

Lịch sử ra đời

- ❑ 1980s Giảm sự phức tạp – object orientation, functional programming.
- ❑ 1990s Khai thác phần cứng song song và phân tán (parallel và distributed) làm cho chương trình chạy nhanh hơn, kết quả là hàng loạt ngôn ngữ mở rộng khả năng lập trình parallel
- ❑ 2000s Ngôn ngữ lập trình phục vụ phát triển Internet

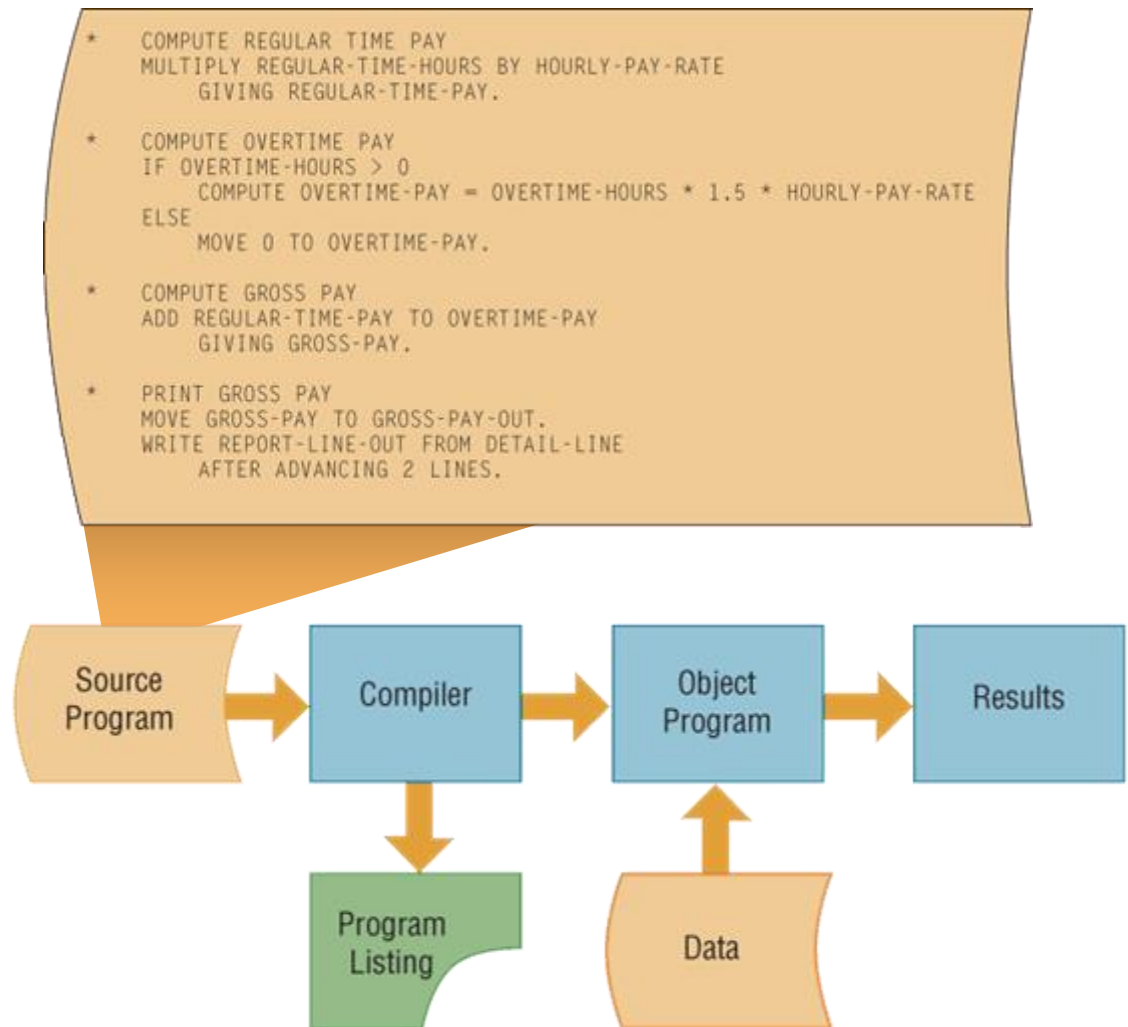
Khủng hoảng phần mềm

- Khái niệm software crisis bao gồm hàng loạt vấn đề nảy sinh trong việc phát triển phần mềm trong những năm 1960s khi muốn xây dựng những hệ thống phần mềm lớn trên cơ sở các kỹ thuật phát triển thời đó.
- Kết quả:
 1. Thời gian và giá thành tăng vọt tới mức không thể chấp nhận nổi.
 2. Năng suất không đáp ứng yêu cầu.
 3. Chất lượng phần mềm bị giảm, thấp.
- Để giải quyết các vấn đề kể trên, chuyên ngành software engineering ra đời.

Chương trình biên dịch

- Biên dịch - Compiler?

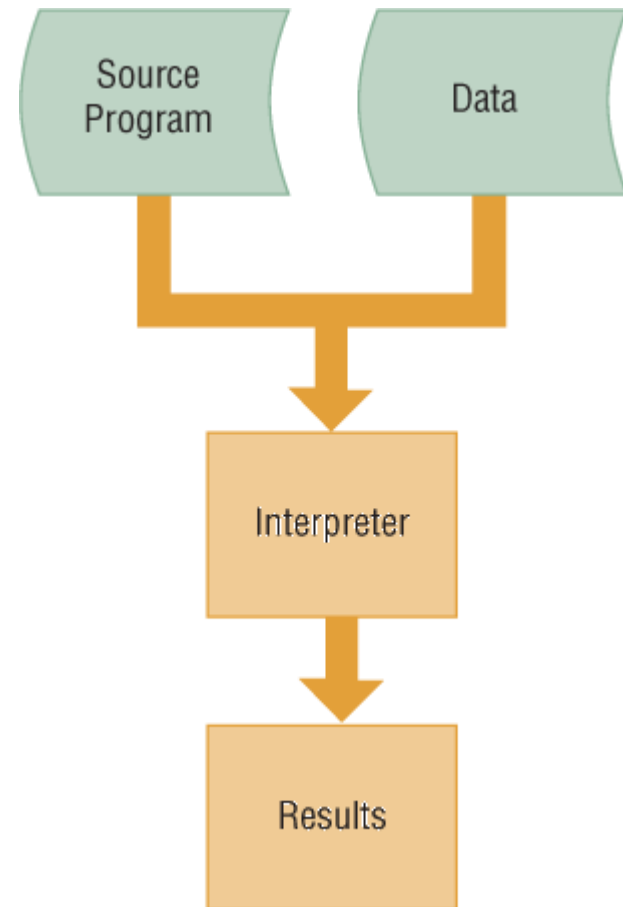
- Là chương trình thực hiện biên dịch toàn bộ chương trình nguồn thành mã máy trước khi thực hiện



Chương trình thông dịch

- Thông dịch - Interpreter?

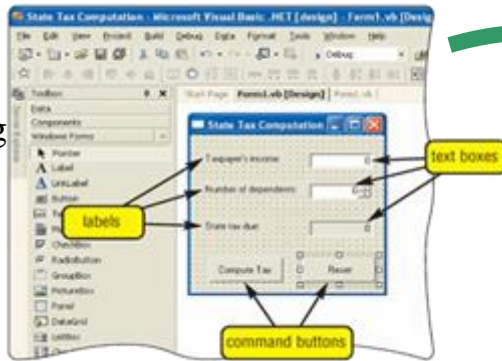
- Là chương trình dịch và thực hiện từng dòng lệnh của chương trình cùng lúc
- Không tạo ra object program



Object-Oriented Programming Languages

- **Visual Studio .NET 2003, 2005?**
 - Bước phát triển của visual programming languages và RAD tools
 - .NET là tập hợp các công nghệ cho phép program chạy trên Internet
 - **Visual Basic .NET 2003-5** dùng để xd các ct hướng đối tượng phức tạp

Step 1. LTV thiết kế giao diện người dùng - user interface.

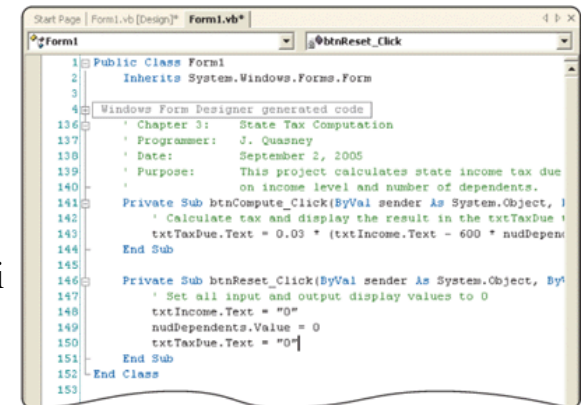


Step 4. LTV kiểm tra application.

Step 2. LTV gán các thuộc tính cho mỗi object trên form.

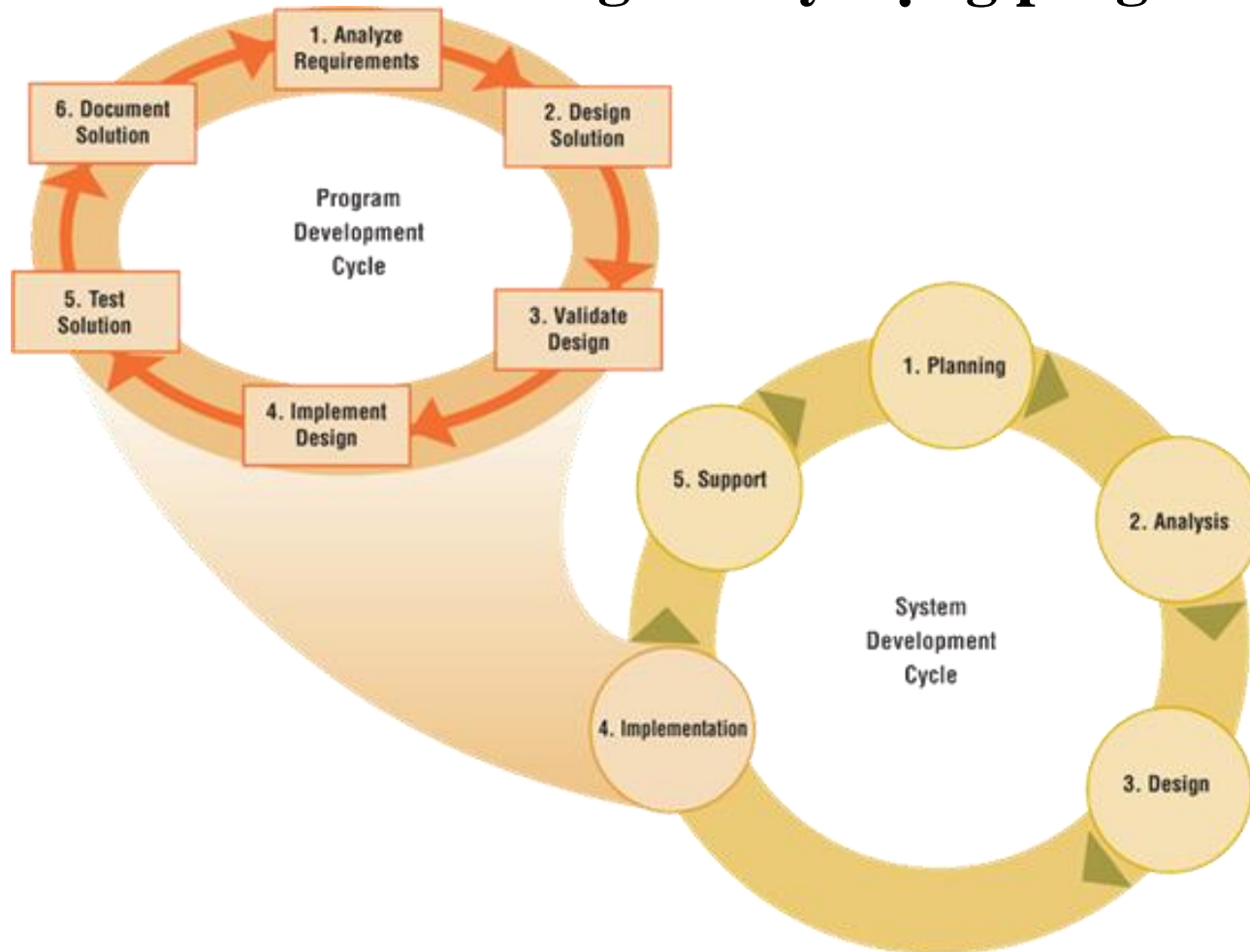


Step 3. LTV viết code để xác định các action cần thực hiện đối với các sự kiện cần thiết.



Chu trình phát triển phần mềm

- Là các bước mà LTV dùng để xây dựng programs



Step 1 — Analyze Requirements

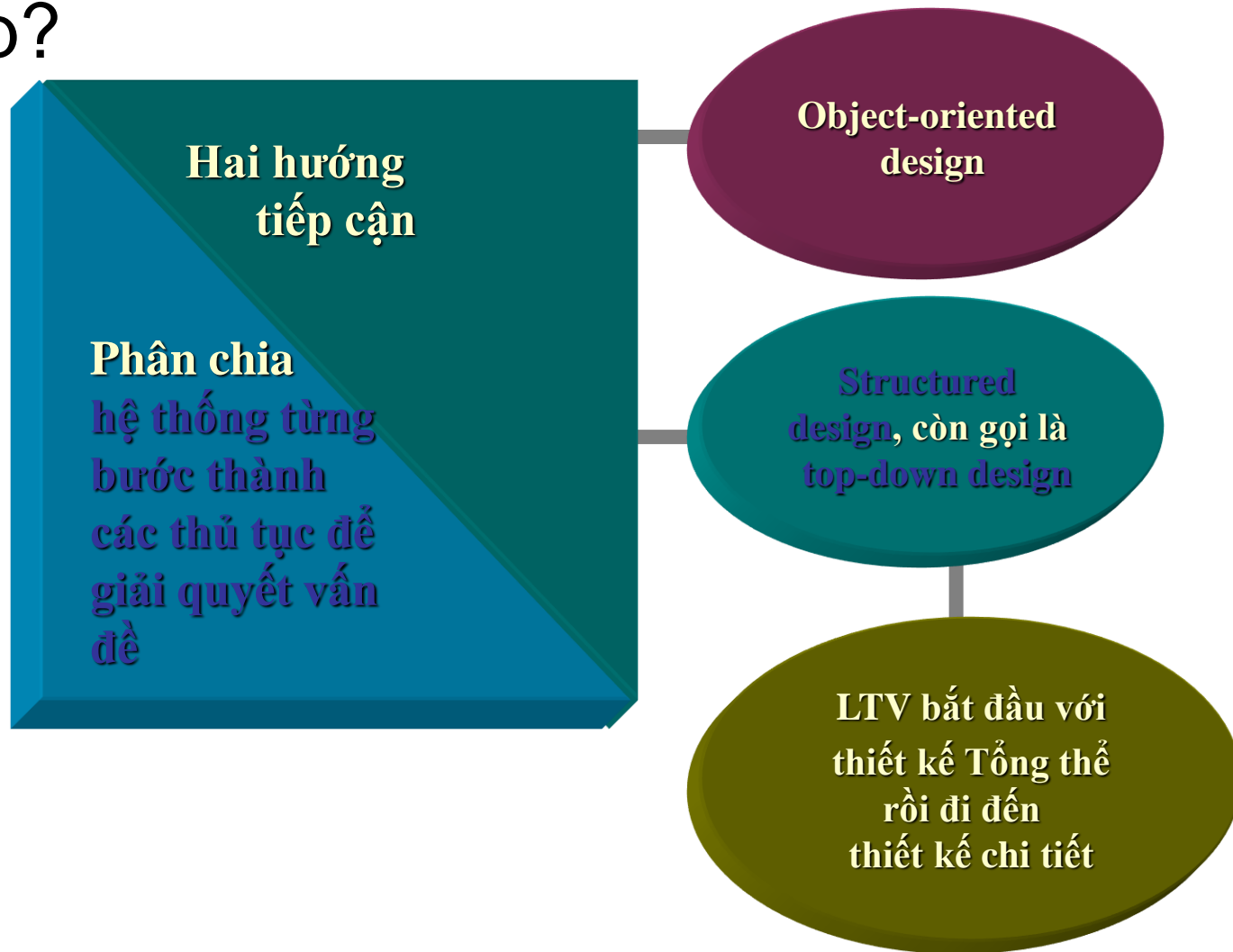
- Các việc cần làm khi phân tích yêu cầu?
 1. Thiết lập các requirements
 2. Gặp các nhà phân tích hệ thống và users
 3. Xác định input, output, processing, và các thành phần dữ liệu
 - IPO chart—Xác định đầu vào, đầu ra và các bước xử lý

IPO CHART

Input	Processing	Output
Regular Time Hours Worked	Read regular time hours worked, overtime hours worked, hourly pay rate.	Gross Pay
Overtime Hours Worked	Calculate regular time pay.	
Hourly Pay Rate	If employee worked overtime, calculate overtime pay.	
	Calculate gross pay. Print gross pay.	

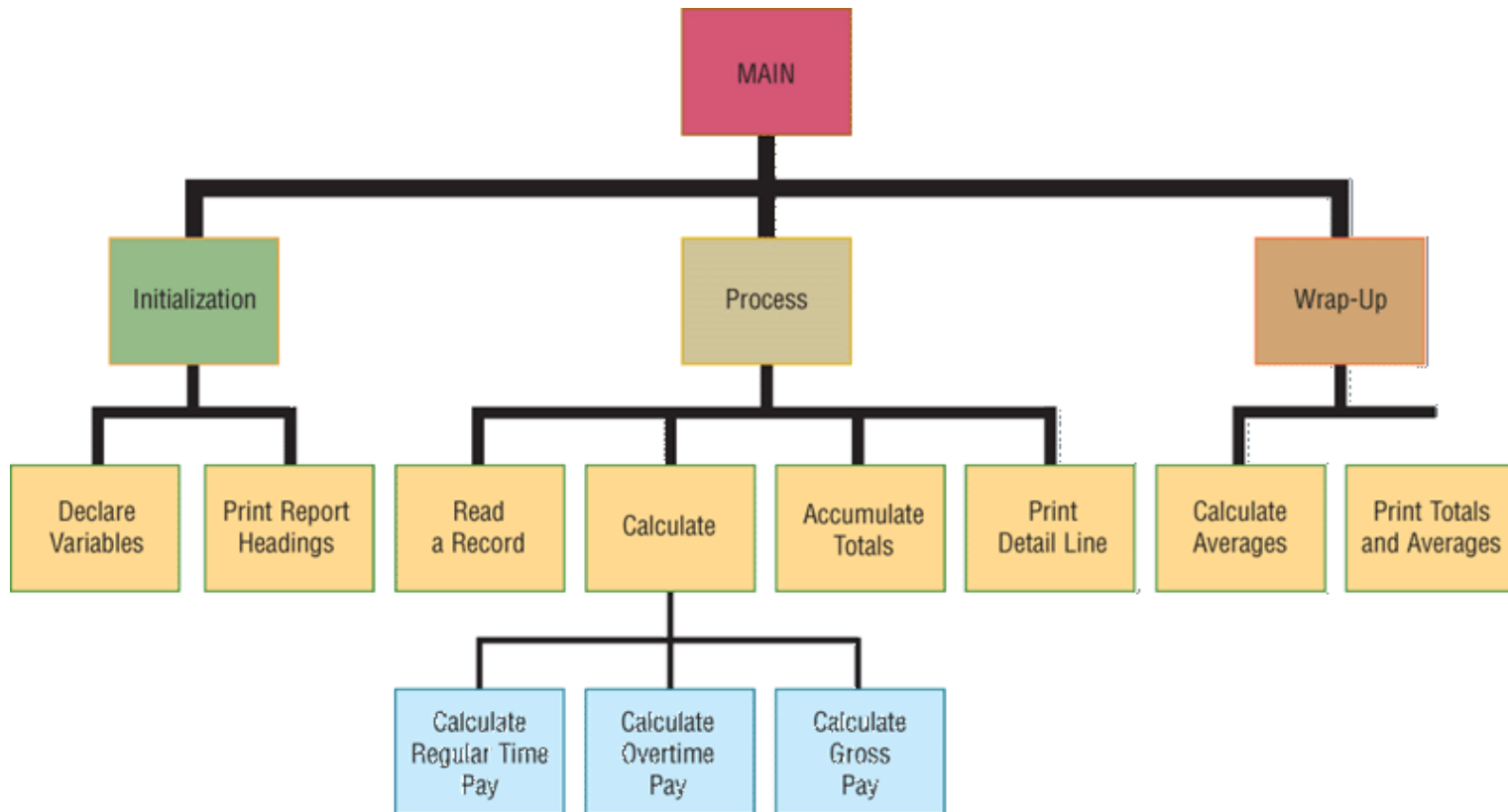
Step 2 — Design Solution

- Những việc cần làm trong bước thiết kế giải pháp?



Step 2 — Design Solution

- Sơ đồ phân cấp chức năng- **hierarchy chart**?
 - Trực quan hóa các modules
 - Sơ đồ cấu trúc

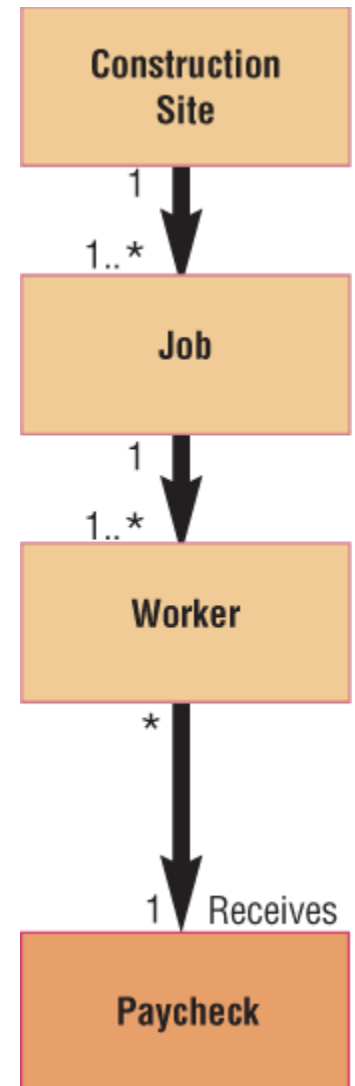


Step 2 — Design Solution

- Object-oriented (OO) design là gì?

- LTV đóng gói dữ liệu và các thủ tục xử lý dữ liệu trong một object

- Các objects được nhóm lại thành các classes
- Biểu đồ lớp thể hiện trực quan các quan hệ phân cấp quan hệ của các classes



Step 3 — Validate Design

- Những điều cần làm trong giai đoạn này?

**Kiểm tra
độ chính xác
của chương trình**

Desk check
**LTV dùng các dữ liệu
thử nghiệm để kiểm tra
chương trình**

Test data
các dữ liệu thử nghiệm
giống như số liệu thực mà
chương trình sẽ thực hiện

**LTV kiểm tra
logic cho tính đúng đắn
và thử tìm các lỗi logic**

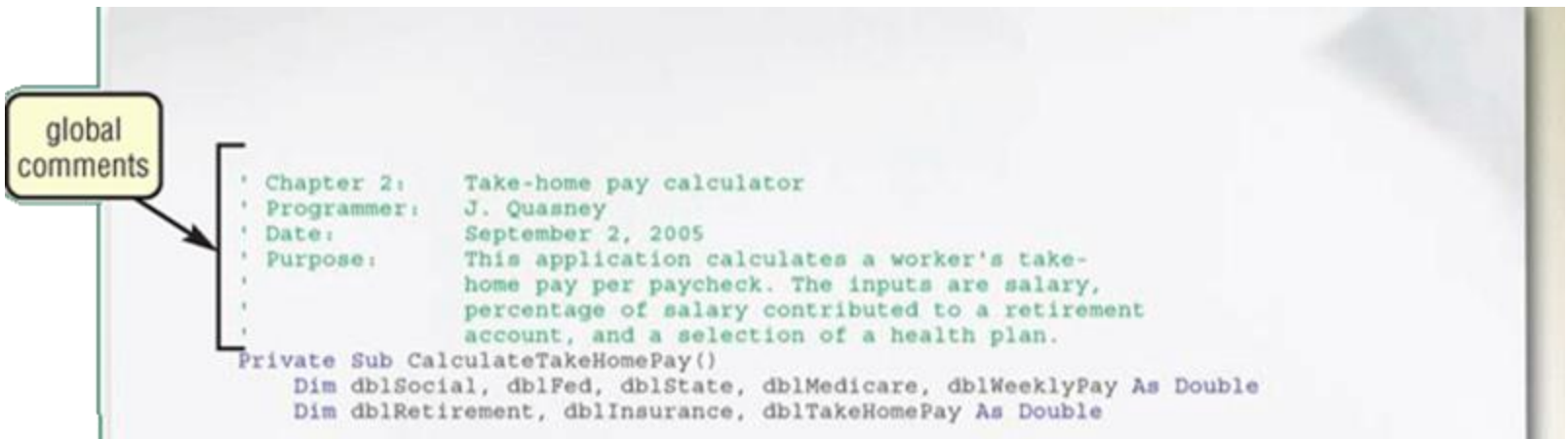
Logic error
các sai sót khi thiết kế
gây ra những kết quả
không chính xác

Structured walkthrough
**LTV mô tả logic
của thuật toán trong khi
programming team duyệt theo
logic chương trình**

Step 4 — Implementation

- **Implementation?**

- **Viết code : dịch từ thiết kế thành program**
 - **Syntax**—Quy tắc xác định cách viết các lệnh
 - **Comments**—program documentation
- **Extreme programming (XP)**—coding và testing ngay sau khi các yêu cầu được xác định



Step 5 — Test Solution

- Những việc cần làm ?

Đảm bảo chương trình chạy
thông và cho kết quả chính xác

Debugging—Tìm và sửa các lỗi
syntax và logic errors

Kiểm tra phiên bản
beta, giao cho Users
dùng thử và thu thập
phản hồi

Step 6 — Document Solution

- Là bước không kém quan trọng
 - 2 hoạt động

Rà soát lại program code—loại bỏ các **dead code**, tức các lệnh mà **chương trình** không bao giờ gọi đến

Rà soát, hoàn thiện documentation

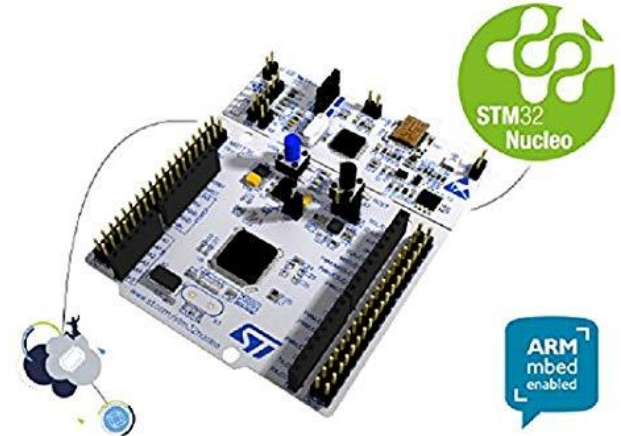
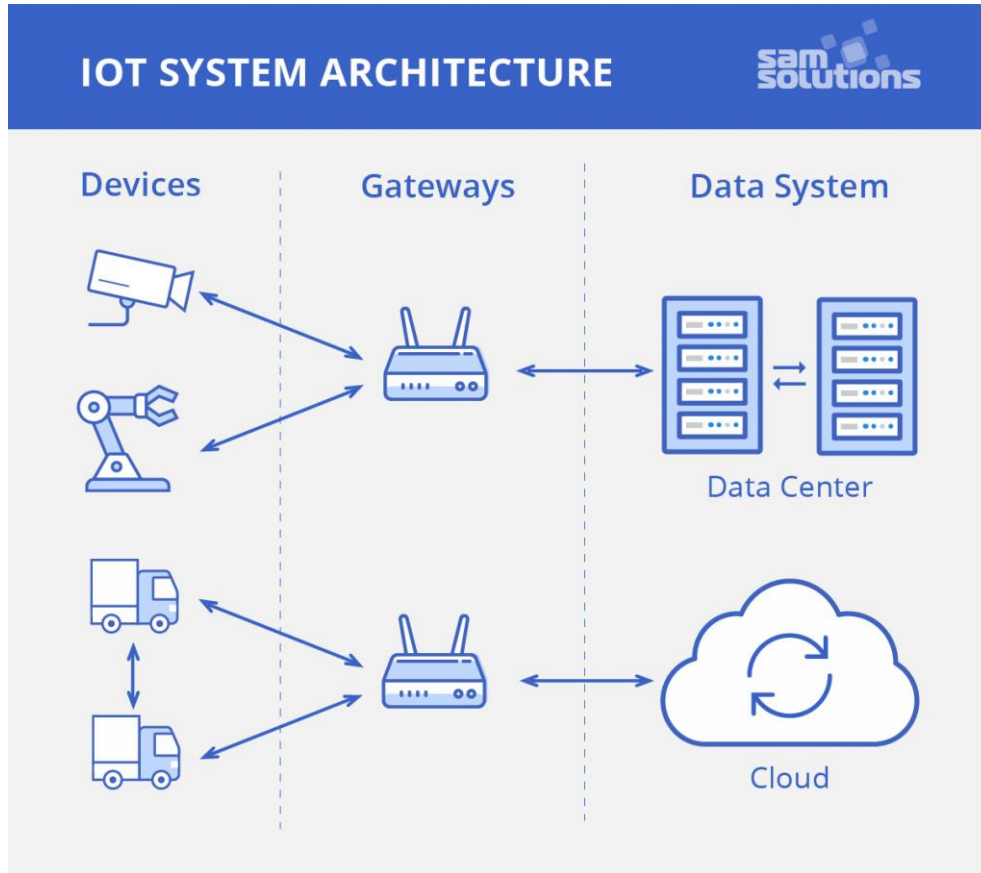
Các mô hình lập trình

- **Programming paradigm**
 - . Là một khuôn mẫu - pattern dùng như Mô hình lập trình máy tính
 - . Là một mô hình cho các lớp có cùng những đặc trưng cơ bản
- **Programming technique**
 - . Liên quan đến các ý tưởng thuật toán để giải quyết một lớp vấn đề tương ứng
 - . Ví dụ: 'Divide and conquer' và 'program development by stepwise refinement'
- **Programming style**
 - . Là cách chúng ta trình bày trong 1 computer program
 - . Phong cách tốt giúp cho chương trình dễ hiểu, dễ đọc, dễ kiểm tra -> dễ bảo trì, cập nhật, gỡ rối, tránh bị lỗi
- **Programming culture**
 - . Tổng hợp các hành vi lập trình, thường liên qua đến các dòng ngôn ngữ lập trình
 - . Là tổng thể của Mô hình chính, phong cách và kỹ thuật lập trình
 - . Là nhân cách trong lập trình cũng như khai thác các chương trình

Môn Học Kỹ Thuật Lập Trình

- Cung cấp *kiến thức cơ bản* trong tư duy *lập trình hệ thống* sử dụng ngôn ngữ lập trình C.
- Cung cấp *kiến thức cơ bản* trong tư duy ngôn ngữ lập trình C++.
- Sinh viên sẽ nắm vững được kiến thức về
 - *lập trình có cấu trúc* với các *kiểu dữ liệu có sẵn*
 - Sử dụng C++ như một ngôn ngữ C mở rộng
 - *Lập trình C++ hướng theo đối tượng (Object oriented design)*
 - *Các cấu trúc dữ liệu điển hình dùng trong hệ thống nhúng (Embedded system)*

Đối tượng lập trình



Tại Sao Học C ?

- C là ngôn ngữ gần với ngôn ngữ máy nhất vì thế được sử dụng tại hầu hết các *hệ thống nhúng*
 - Ngôn ngữ lập trình cấp cao cho hệ vi xử lý: STM32, DsPIC, Atmega,
 - Ngôn ngữ lai: MATLAB, NI CVI, Arduino ?
 - Ngôn ngữ lập trình cho Windows: Visual C++,
- Hệ thống thư viện phần mềm có sẵn trên Internet

<https://github.com/>

www.thecodeproject.com

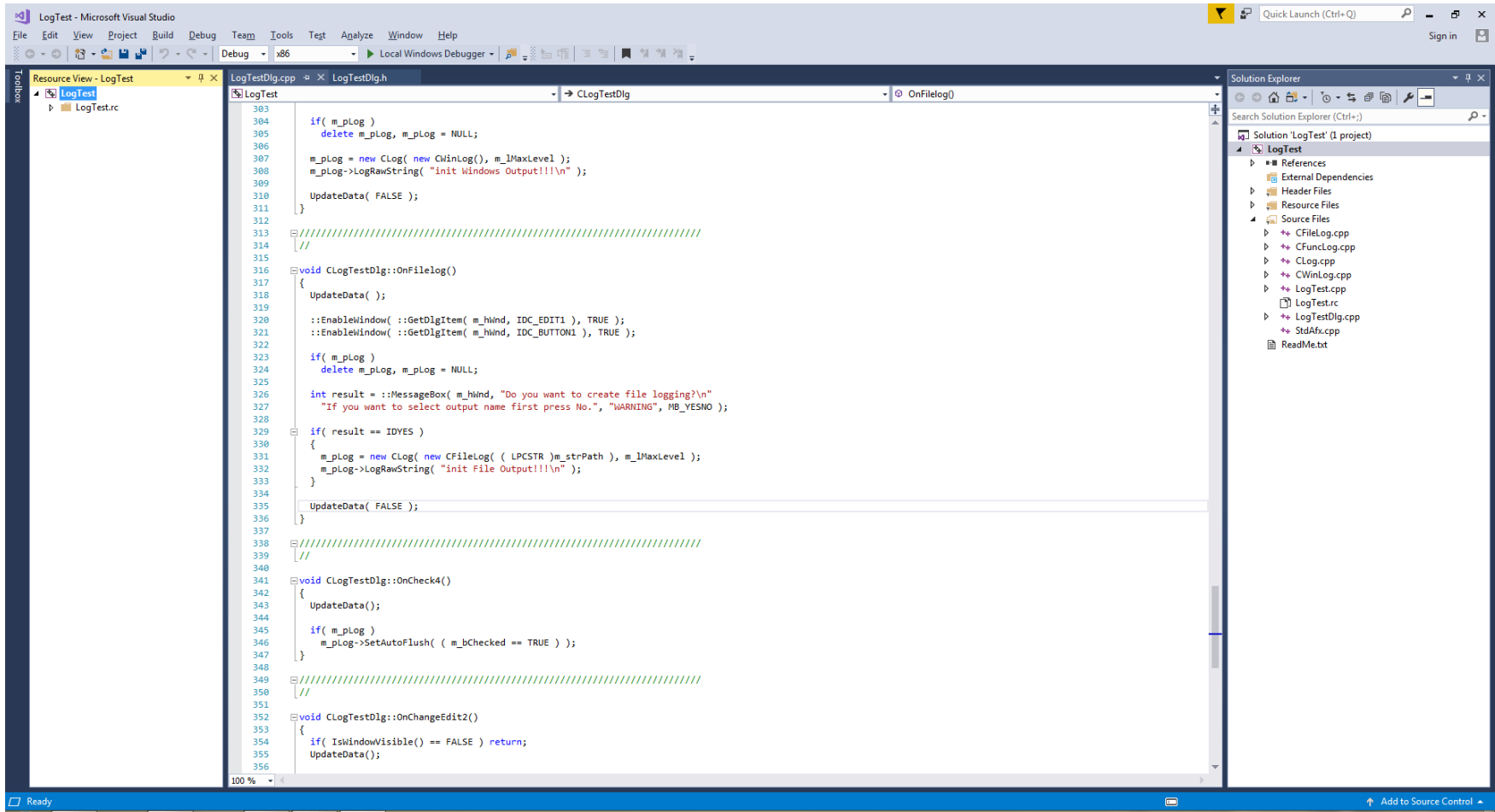
Không Chỉ C

- Môn học đề cập đến những vấn đề cơ bản nhất của một ngôn ngữ lập trình
 - Biến, kiểu dữ liệu, giá trị, biểu thức
 - Thực thi chương trình tuần tự
- Các cấu trúc dữ liệu và giải thuật thông dụng.
- Và một số khái niệm liên quan đến *thiết kế chương trình*
 - Sự trừu tượng hóa thủ tục, hàm, dữ liệu
 - Sự đóng gói, module hóa, tính sử dụng lại

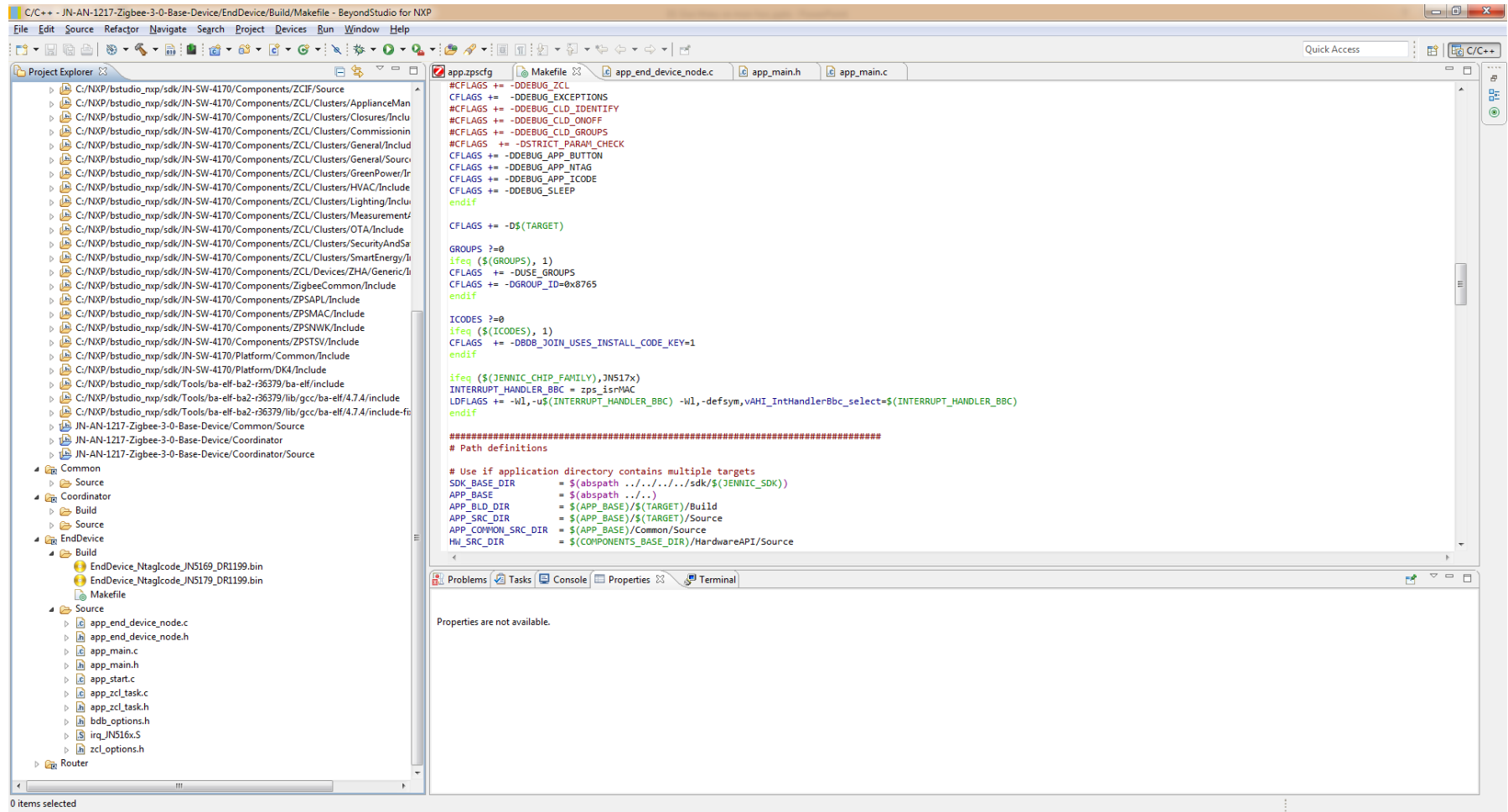
Giải Quyết Vấn Đề / Thiết Kế Chương Trình

- *Xác định* rõ vấn đề.
- *Phân tích* vấn đề.
- Thiết kế *thuật toán* nhằm giải quyết vấn đề.
- *Cài đặt* thuật toán (viết chương trình).
(nhớ ghi chú thích trong chương trình)
- *Chạy thử* và xác nhận tính đúng đắn của chương trình.
(chạy thử - sửa lỗi)
- *Duy trì* và cập nhật chương trình.

Những Gì Đang Chờ Đợi Bạn tương lai



Những Gì Đang Chờ Đợi Bạn tương lai



Các yêu cầu ngắn hạn

- Điểm số: trải đều từ 0 đến 10 với trung bình từ 6 đến 7.
- Một môn học khó với nội dung đề cập đến nhiều vấn đề.
- Bài tập ngắn về nhà hàng tuần, một bài tập dài cho cả kỳ, bài kiểm tra ngắn trên lớp hàng tuần.
- Bài kiểm tra giữa kỳ và cuối kỳ trong 1 tiếng với nội dung
 - Câu hỏi lý thuyết
 - Kiểm tra kết quả chương trình
 - Viết chương trình giải quyết vấn đề thực tế
- Vậy có gì thú vị?

Lời Khuyên Với Sinh Viên

- Học nghiêm túc ngay từ đầu kỳ.
- Có thói quen tìm kiếm sự trợ giúp sớm và thường xuyên đối với những nội dung khó.
- Hoàn thành, chạy thử và gỡ lỗi các bài tập ngắn của từng tuần trước ở nhà vì chúng thường liên quan đến bài kiểm tra ngắn kế tiếp.
- Nộp bài tập dài đúng hạn (bài tập nộp muộn sẽ nhận điểm 0, không nộp bài tập sẽ nhận điểm âm).
- Có thể đặt câu hỏi bất cứ lúc nào.

Tài Liệu Tham Khảo

- *Kernighan B.W., Ritchie D.M., The C Programming Language, 2nd edition, Prentice Hall, 1998*
- *Kernighan B.W., Pike R., The Practice of Programming, Addison Wesley, 1999*
- *Trần Việt Linh, Lê Đăng Hưng, Lê Đức Trung, Nguyễn Thanh Thủy, Nhập Môn Lập Trình Ngôn Ngữ C, NXB Khoa Học & Kỹ Thuật, 2003*
- *Microsoft Developer Network (MSDN)*
- *Internet*

Liên Hệ

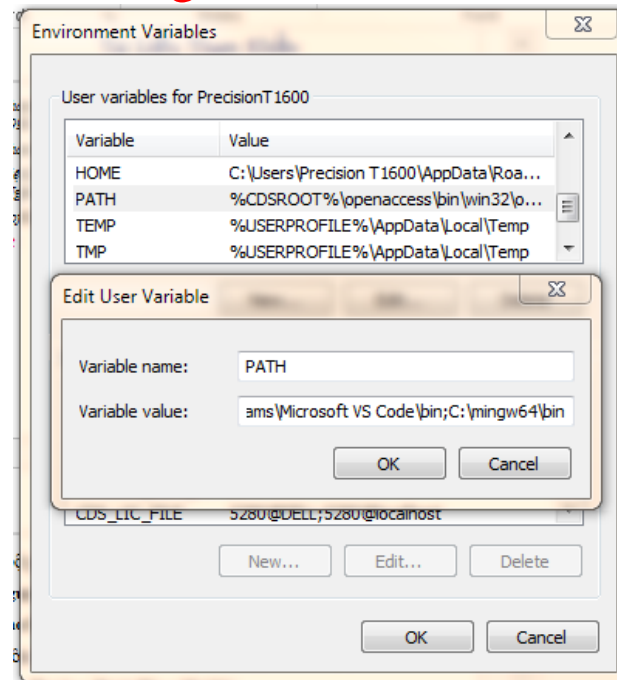
- Cán bộ phụ trách môn học
 - **Nguyễn Hồng Quang** - quang.nguyenhong1@hust.edu.vn
 - **Nguyễn Trí Cường** - cuong.nguyentri@hust.edu.vn
- Bộ môn Tự động hóa CN (C9-104), Viện Điện,
Trường Đại học Bách Khoa Hà Nội.

Hãy Bắt Đầu!

- Cài đặt phần mềm trên nền hệ điều hành Windows

<https://code.visualstudio.com/docs/cpp/config-mingw>

- Cài đặt Visual Studio Code.
- Cài đặt **C++ extension for VSCode**
- Cài đặt **Mingw-w64**, tại thư mục đơn giản như C:\Mingw64
- Cài đặt đường dẫn path cho file **g++.exe**



Các file cấu hình đi kèm

- *c_cpp_properties.json* (compiler path and IntelliSense settings)
- *tasks.json* (build instructions)
- *launch.json* (debugger settings)
(.json JavaScript Object Notation (JSON) format)

Đường dẫn cho trình biên dịch

(compiler path)

Configuration name

A friendly name that identifies a configuration. **Mac**, **Linux**, and **Win32** are special identifiers for configurations that will be auto-selected on those platforms, but the name of the identifier can be anything.

Select a configuration set to edit.

Win32 ▼

Add Configuration

Compiler path

The full path of the compiler being used to build, e.g. `/usr/bin/gcc`, to enable more accurate IntelliSense. Arguments can be added to modify the includes/defines used, e.g. `-nostdinc++`, `-m32`, etc., but paths with spaces must be surrounded by double quotes `"` if arguments are used.

Specify a compiler path or select a detected compiler path from the drop-down list.

C:/mingw64/bin/g++.exe ▼

IntelliSense mode

The IntelliSense mode used by the IntelliSense engine. The `${default}` mode will choose the default for that platform. Windows defaults to `msvc-x64`, Linux defaults to `gcc-x64`, and Mac defaults to `clang-x64`. Select a specific IntelliSense mode to override the `${default}` mode.

gcc-x64 ▼

Tạo tác vụ dịch

(build task)

```
{
  "version": "2.0.0",
  "tasks": [
    {
      "label": "build hello world",
      "type": "shell",
      "command": "g++",
      "args": [
        "-g",
        "-o",
        "hello",
        "hello.cpp"
      ],
      "group": {
        "kind": "build",
        "isDefault": true
      }
    },
    {
      "type": "shell",
      "label": "g++.exe build active file",
      "command": "C:\\mingw64\\bin\\g++.exe",
      "args": [
        "-g",
        "${file}",
        "-o",
        "${fileDirname}\\${fileBasenameNoExtension}.exe"
      ],
    },
  ]
}
```

Cấu hình chế độ debug

(launch.json)

```
{
  // Use IntelliSense to learn about possible attributes.
  // Hover to view descriptions of existing attributes.
  // For more information, visit: https://go.microsoft.com/fwlink/?linkid=830387
  "version": "0.2.0",
  "configurations": [
    {
      "name": "(gdb) Launch",
      "type": "cppdbg",
      "request": "launch",
      "program": "${workspaceFolder}/hello.exe",
      "args": [],
      "stopAtEntry": true,
      "cwd": "${workspaceFolder}",
      "environment": [],
      "externalConsole": true,
      "MIMode": "gdb",
      "miDebuggerPath": "C:\\mingw64\\bin\\gdb.exe",
      "setupCommands": [
        {
          "description": "Enable pretty-printing for gdb",
          "text": "-enable-pretty-printing",
          "ignoreFailures": true
        }
      ]
    }
  ]
}
```

Kỹ Thuật Lập Trình

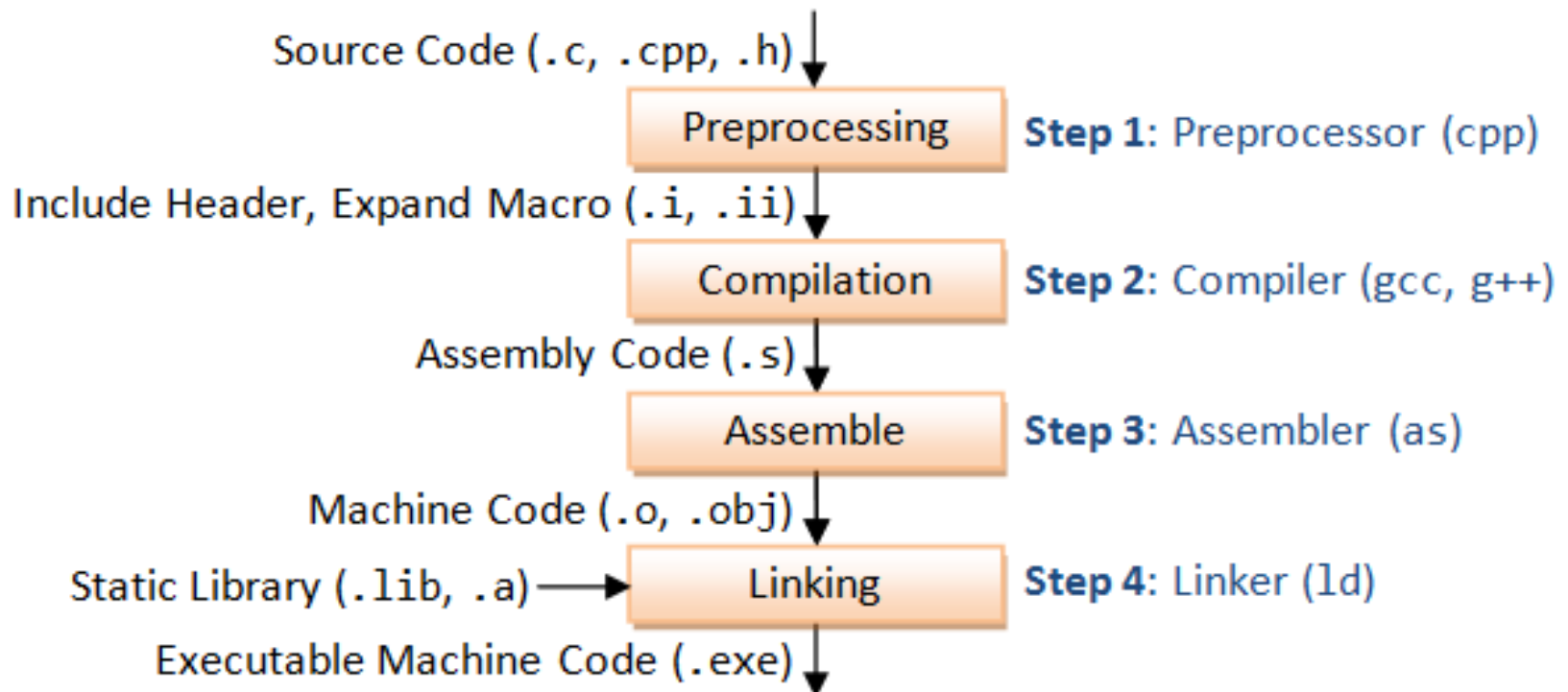
(Ngôn Ngữ Lập Trình C)

Bài giảng số 2

Các khái niệm cơ bản trong C

Nguyễn Trí Cường

Quy trình biên dịch (compiler)




Biến Trong C


```
#include <stdio.h>
int main(void)
{
```

```
int firstOperand;
int secondOperand;
int thirdOperand;
```

```
firstOperand = 1;
secondOperand = 2;
thirdOperand = firstOperand + secondOperand;
printf("%d", thirdOperand);
```

```
return 0;
}
```

 *Thứ bạn cần trong mọi chương trình*

 *Khai báo biến
(xác định ô nhớ cho các biến)*

 *Các chỉ dẫn cho CPU (các câu lệnh)*

Vài Chú Ý

- Mỗi biến trong C được dành riêng một vùng nhớ.
- Tên biến cần được chọn phù hợp nhằm giúp người đọc chương trình hiểu được chức năng của biến trong chương trình.
- Khi tất cả các biến đã có ô nhớ cụ thể, chương trình bắt đầu được thực hiện.
- Tại một thời điểm chỉ có một lệnh được thực hiện, trình tự thực hiện các lệnh phụ thuộc vào vị trí của lệnh trong chương trình.
- Các biến cần được *khởi tạo* trước khi dùng.

Tên Biến

- Trong C, *tên* (từ định danh, identifier) cần tuân theo các nguyên tắc sau
 - Dùng kí tự, số và dấu gạch dưới (_)
 - Không bắt đầu với một số
 - Không trùng với *từ dành riêng* (reserved words, từ khóa)
 - Có phân biệt giữa chữ hoa và chữ thường
 - Có thể có độ dài bất kỳ
- *Việc lựa chọn tên rất quan trọng trong việc đọc hiểu chương trình*
 - Thông thường danh từ hoặc chuỗi danh từ mô tả nội dung biến được chọn cho tên biến

Ví Dụ Tên Biến

Đúng	Không đúng	Đúng nhưng...
<code>rectangleWidth</code>	<code>10TimesLength</code>	<code>a1</code>
<code>rectangle_Width</code>	<code>My Variable</code>	<code>1</code>
<code>rectangle_width</code>	<code>int</code>	<code>0</code>

Khai Báo Biến

- **int months;**

- Các biến kiểu nguyên đại diện cho các số nguyên

- 1, 17, -32, 0

Không phải 1.5, 2.0, 'A'

- **double pi;**

- Các biến dấu phẩy động đại diện cho các số thực

- 3.14, -27.5, 6.02e23, 5.0

Không phải 3

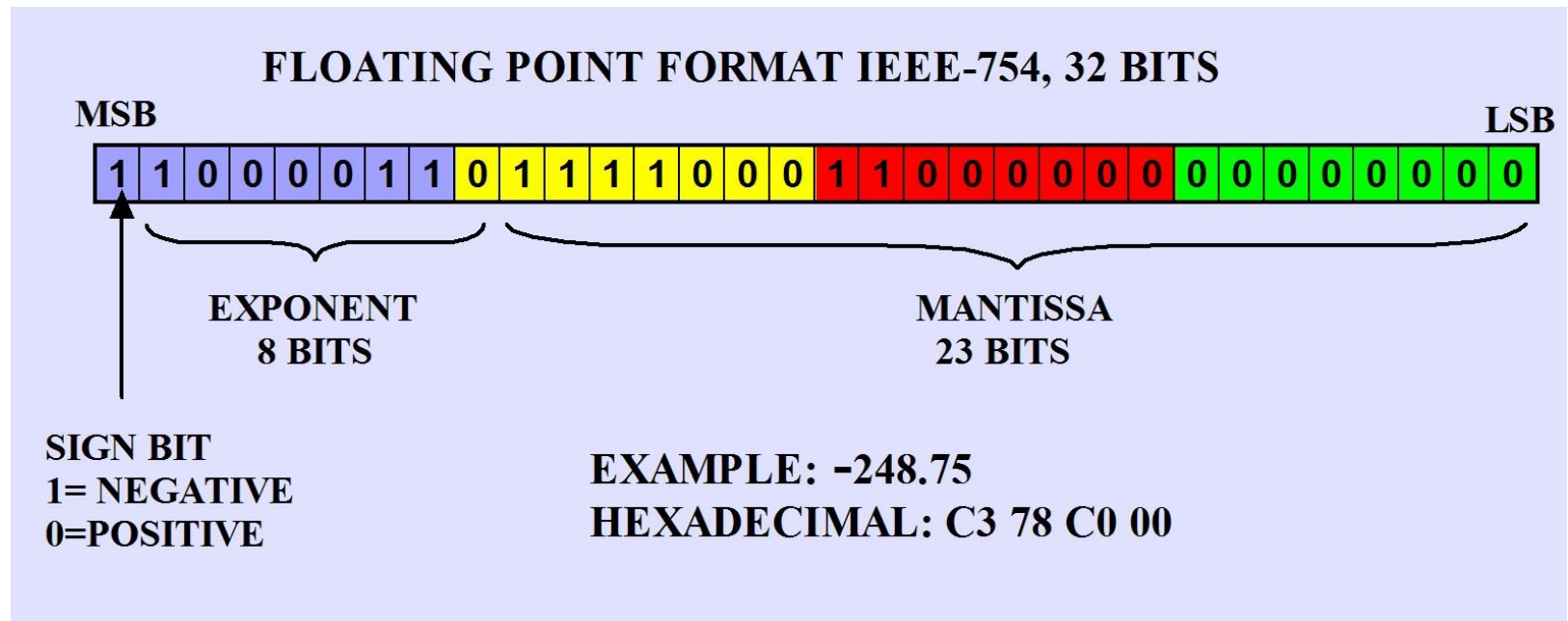
- **char first_initial, marital_status;**

- Các biến kiểu ký tự đại diện cho các ký tự của bàn phím

- 'a', 'b', 'M', '0', '9', '#', ' '

Không phải "Bill"

Ví dụ về kiểu float trong C



BT: Tìm hiểu quy đổi Floating point từ Nhị phân sang Thập phân và ngược lại theo tiêu chuẩn IEEE-754 + viết chương trình

Ví Dụ $F \rightarrow C$ (chương trình 1)

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

```
int main(void)
```

```
{
```

```
    double fahrenheit, celsius;
```

```
    celsius = (fahrenheit - 32.0) * 5.0 / 9.0;
```

```
    return(0) ;
```

```
}
```

Ví Dụ F→C (chương trình 2)

```
#include <stdio.h>

int main(void)
{
    double fahrenheit, celsius;

    printf("Enter a Fahrenheit temperature: ");
    scanf("%lf", &fahrenheit);

    celsius = (fahrenheit - 32.0) * 5.0 / 9.0;

    printf("That equals %f degrees Celsius.",
        celsius);

    return(0);
}
```


Ví Dụ $F \rightarrow C$ (chương trình 3)

```
#include <stdio.h>

int main(void)
{
    double fahrenheit, celsius;

    printf("Enter a Fahrenheit temperature: ");
    scanf("%lf", &fahrenheit);

    celsius = fahrenheit - 32.0;
    celsius = celsius * 5.0 / 9.0;

    printf("That equals %f degrees Celsius.",
        celsius);

    return(0);
}
```

Cần Quan Tâm Đến Biểu Thức?

- Chúng ta cần các quy tắc chính xác giúp cho việc hiểu ý nghĩa của các biểu thức

$4 - 4 * 4 + 4$ có giá trị bằng bao nhiêu?

- Phép tính số học trong máy tính không phải lúc nào cũng chính xác

$(1.0 / 9.0) * 9.0$ có thể có kết quả
 0.99999998213

- Phép chia hai số có kiểu `int` có thể cho kết quả hoàn toàn khác với những gì ta mong đợi

$2 / 3$ cho kết quả bằng `0` trong C

Tại Sao Cần Dùng **int**

- Đôi khi chỉ số có kiểu **int** là phù hợp
 - “ô số 15” thay vì “ô số 14.9999999”
- Số có kiểu **double** đôi khi không chính xác trong việc biểu diễn số nguyên
 - Trong toán học: $3 * 15 * (1/3) = 15$
 - Nhưng trong máy tính: $3.0 * 15.0 * (1.0/3.0)$ có thể bằng **14.9999999997**
- Và...
 - Các phép tính với số **double** thường chậm hơn với số **int**
 - Số có kiểu **double** cần nhiều bộ nhớ hơn số có kiểu **int**

Ví Dụ Cụ Thể

- Công thức toán học

$$\frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

- Biểu thức trong C

`(- b + sqrt (b*b - 4.0*a*c)) / (2.0*a)`

Biểu Thức Chứa Nhiều Kiểu

- Giá trị của **2 * 3.14** là bao nhiêu?
- Trình biên dịch sẽ *tự động* (implicitly) chuyển kiểu **int** sang kiểu **double** mỗi khi gặp biểu thức dạng này

int + double

double + double

2 * 3 * 3.14

(2*3) * 3.14

6 * 3.14

6.0 * 3.14

18.84

- Bạn nên đặc biệt tránh sử dụng các biểu thức chứa nhiều kiểu dữ liệu
 - Nên dùng **2.0 / 3.0 * 3.14**

Chuyển Kiểu Trong Phép Gán

```
int total, count;
```

```
double avg;
```

```
total = 97;
```

```
count = 10;
```

```
avg = total / count;    /*avg is 9.0*/
```

```
total = avg;            /*BAD*/
```

chuyển kiểu tự động
trong phép gán



Phép Toán Chuyển Kiểu

- Bạn có thể sử dụng *phép toán chuyển kiểu* (casting) trong chương trình để chuyển kiểu cho một giá trị
 - Chuyển giá trị của một biểu thức sang một kiểu khác
- Dạng lệnh: **(type) expression**
- Ví dụ
(double) myage
(int) (balance + deposit)
- Chú ý: Phép toán chuyển kiểu không thay đổi cách thức tính toán biểu thức.

Sử Dụng Phép Toán Chuyển Kiểu

```
int total, count ;
```

```
double avg;
```

```
total = 97 ;
```

```
count = 10 ;
```

```
/*avg is 9.0*/
```

```
avg = total / count;
```

```
/*avg is 9.7*/
```

```
avg = (double) total / (double) count;
```

```
/*avg is 9.0*/
```

```
avg = (double) (total / count);
```

chuyển kiểu tự động
trong phép gán



phép toán
chuyển kiểu



Kiểu Dữ Liệu Trong C

- Tất cả các biến, giá trị, biểu thức đều có kiểu dữ liệu.
- C quan tâm đến kiểu dữ liệu của từng đại lượng.
- Từ bây giờ: *luôn nhớ kiểu dữ liệu của tất cả các đại lượng trong chương trình của bạn.*

Bài Học Cơ Bản

- Chương trình cần được viết sáng của nhất có thể.
- Chương trình cần đơn giản, những biểu thức phức tạp cần được cài đặt bằng những câu lệnh đơn giản.
- Sử dụng dấu ngoặc đơn để chỉ rõ thứ tự ưu tiên của các toán tử trong những trường hợp có thể gây hiểu nhầm.
- Quá trình chuyển đổi kiểu cần được thể hiện rõ ràng bởi người lập trình nhằm tránh việc tự động chuyển kiểu trong các biểu thức hay các lệnh gán.
- Cần chú ý đến kiểu dữ liệu trong việc viết các biểu thức.

Hiển Thị Dữ Liệu Xuất/Nhập

- Các hàm **printf** và **scanf** là các hàm cung cấp các chức năng xuất, nhập cơ bản

printf("control string", list of expressions);

scanf("control string", list of &variables);

- *Control string*: chuỗi xác định *định dạng* (format) của chuỗi xuất/nhập.
- *Expressions*: chứa dữ liệu cần đưa ra.
- *Variables*: các biến lưu trữ dữ liệu đầu vào.
- **&**: ký tự bắt buộc.

Ví Dụ Định Dạng Đầu Ra

%10.2f 1 2 3 . 5 5 **double**

%10.4f 1 2 3 . 5 5 0 0

%.2f 1 2 3 . 5 5

%10d 4 7 5 **int**

%-10d 4 7 5

%10c a **char**

scanf()

```
scanf("control string", &input list);
```

```
int numPushups;
```

```
printf("Hello. Do how many pushups? ");
```

```
scanf("%d", &numPushups);
```

```
printf("Do %d pushups.\n", numPushups);
```

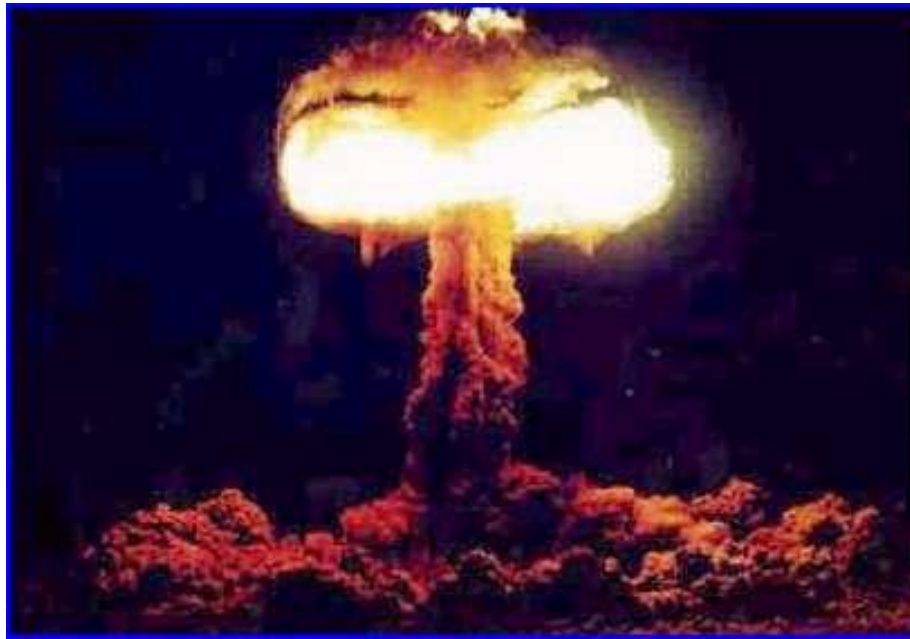
output:

Hello. Do how many pushups? 5
Do 5 pushups.

- *Các biến nằm trong danh sách đầu vào bắt buộc phải có ký tự **&** ở phía trước.*

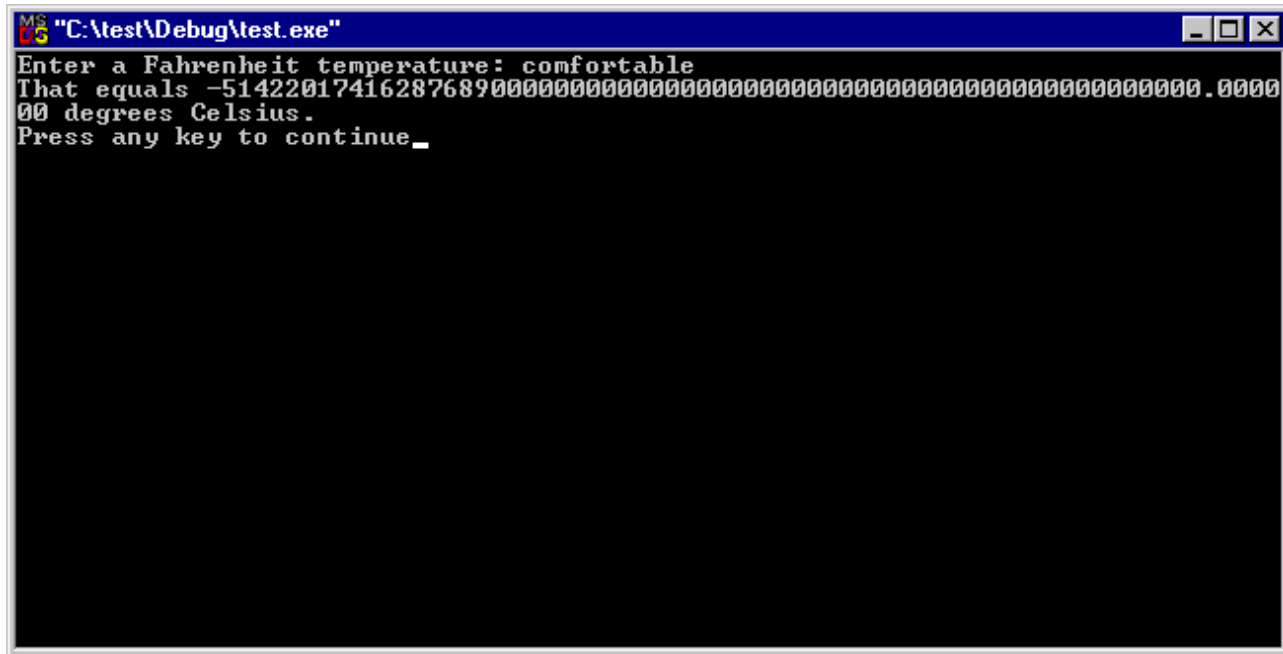
Nếu Bạn Quên &

- Chương trình của bạn vẫn có thể dịch thành công nhưng...



Lỗi Thường Gặp Với Xuất/Nhập

- Giả sử chương trình của bạn có câu lệnh sau
`scanf("%lf", &fahrenheit);`
- Nhưng tại con trỏ lệnh bạn gõ chuỗi “comfortable” như là đầu vào
 - Dữ liệu không được đọc và do vậy **fahrenheit** không được khởi tạo
 - Chương trình có thể gặp nhiều lỗi tiếp theo vì vấn đề này
- *Đây là lỗi của người lập trình, không phải của người dùng.*



Bạn Có Thể Làm Gì?

- Bản thân lệnh **scanf()** trả về số biến đầu vào đã được đọc thành công

```
int scanfCount;
```

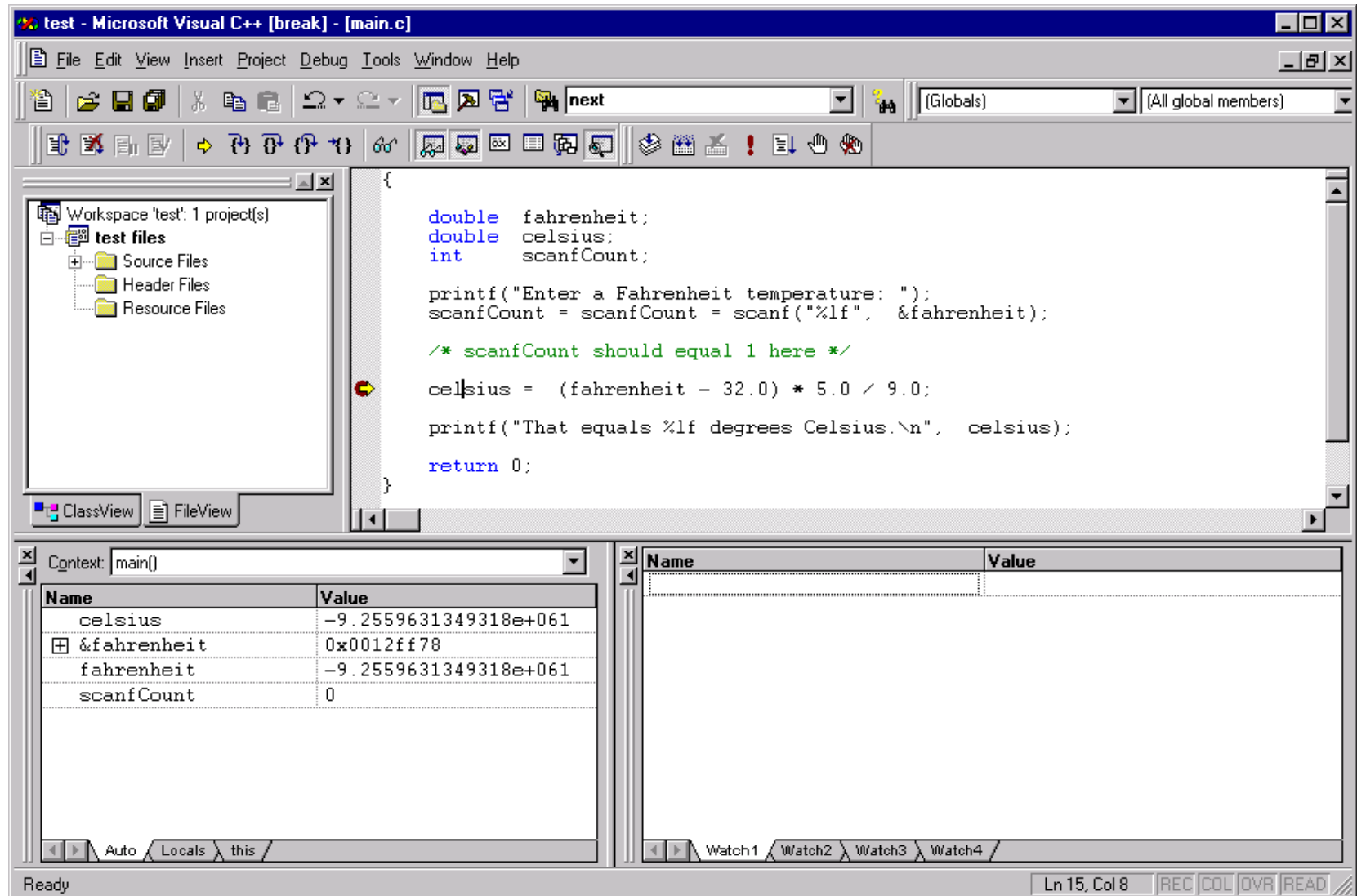
```
scanfCount = scanf("%d", &studentID) ;
```

```
/* if scanfCount is not equal to 1 at this point,
```

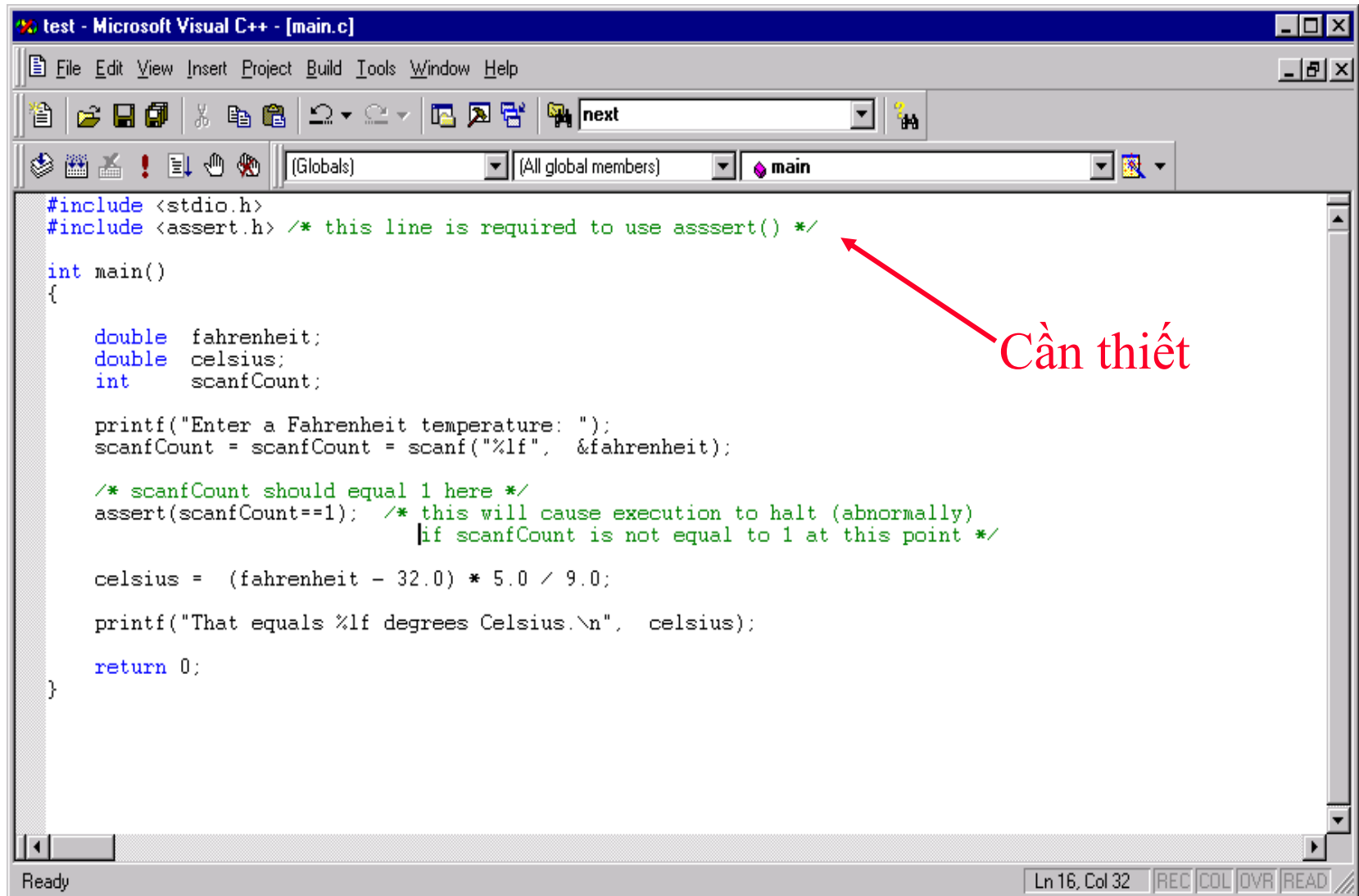
```
the user has made some kind of mistake.
```

```
Handle it. */
```

Chương Trình Dừng Tại Breakpoint



Dùng Câu Lệnh `assert()`



The screenshot shows the Microsoft Visual C++ IDE with a file named `main.c`. The code is as follows:

```
#include <stdio.h>
#include <assert.h> /* this line is required to use assert() */

int main()
{
    double fahrenheit;
    double celsius;
    int scanfCount;

    printf("Enter a Fahrenheit temperature: ");
    scanfCount = scanf("%lf", &fahrenheit);

    /* scanfCount should equal 1 here */
    assert(scanfCount==1); /* this will cause execution to halt (abnormally)
                           |if scanfCount is not equal to 1 at this point */

    celsius = (fahrenheit - 32.0) * 5.0 / 9.0;
    printf("That equals %lf degrees Celsius.\n", celsius);
    return 0;
}
```

A red arrow points from the text "Cần thiết" (Necessary) to the `#include <assert.h>` line, indicating its importance.

Ready Ln 16, Col 32 REC COL OVR READ

Tóm Lược Về Xuất/Nhập

printf(“control string”, output list);

- **control string** : kiểu dữ liệu và định dạng mong muốn
- **output list** : các biểu thức, giá trị cần xuất ra

scanf(“control string”, &input list);

- **control string**: các biến, giá trị cần đọc vào
- **input list**: các kiểu dữ liệu và định dạng mong muốn
- Có thể dùng để khởi tạo các biến
- Chú ý: không dùng **&** với **printf()**, dùng **&** với **scanf()**
- Với cả hai câu lệnh trên: *điều khiển giữ chỗ trong chuỗi định dạng cần phù hợp với các biểu thức xuất/nhập về số lượng, thứ tự và kiểu dữ liệu.*

Ý Niệm Về Hàm

- Xác định một “*vấn đề nhỏ*” cần được giải quyết trong chương trình của bạn.
- Viết một đoạn mã giải quyết vấn đề.
- Đặt tên cho đoạn mã.
- Mỗi khi bạn cần giải quyết “*vấn đề nhỏ*” đó, sử dụng tên đã đặt cho đoạn mã để nói rằng “*chạy đến đoạn mã để giải quyết vấn đề này và không quay lại cho đến khi vấn đề được giải quyết*”.

Các Hàm Viết Sẵn

- Các hàm viết sẵn thông thường được đóng gói trong “*thư viện*”.
- Mọi trình biên dịch C chuẩn đều có một tập hợp các thư viện chuẩn.
- Nhớ lại **#include <stdio.h>**?
 - Thông báo cho trình biên dịch biết bạn dự định dùng các hàm trong “*thư viện xuất/nhập chuẩn*”
 - **printf()** và **scanf()** thuộc thư viện xuất/nhập chuẩn
 - Và còn rất nhiều hàm liên quan đến xuất/nhập khác nữa
- Có rất nhiều hàm hữu dụng khác trong nhiều thư viện khác nhau.

void

- Từ khóa void có hai chức năng khác nhau trong định nghĩa hàm này:

Cho biết hàm sẽ không
trả về giá trị

```
/* write separator line on output*/
```

```
void PrintBannerLines (void)
```

```
{
```

```
    printf("*****");
```

```
    printf("*****\n");
```

```
}
```

Cho biết hàm sẽ
không có tham số

Kiểu Và Giá Trị Hàm

- Một hàm có thể trả về một giá trị.
- Giống như tất cả các giá trị trong C, giá trị trả về của hàm có kiểu. Hàm được gọi là có kiểu của biến trả về.

```
/* return a "random" number */  
double GenRandom (void)  
{  
    double result;  
    result = ...  
    return result;  
}
```

Kiểu hàm (kiểu của giá trị trả về). Ta nói **GenRandom()** là hàm có kiểu **double** hoặc **GenRandom()** trả về một giá trị **double**

Biến cục bộ, tồn tại chỉ khi hàm đang được thực hiện

Câu lệnh trả về

Giá trị trả về

Các Biến Cục Bộ

- Một hàm có thể định nghĩa các *biến cục bộ* (local variable) dùng riêng.
- Các biến cục bộ này *chỉ* có nghĩa trong phạm vi hàm
 - Các biến cục bộ được cấp phát bộ nhớ khi hàm được gọi
 - Được giải phóng khỏi bộ nhớ khi thoát khỏi hàm
- *Các tham số cũng là các biến cục bộ.*

```
/* Yield area of circle with radius r */  
double circle_area (double r) {  
    double x, area1;  
    x = r * r ;  
    area1 = 3.14 * x ;  
    return( area1 );  
}
```

Tham số

Các biến cục bộ

Thứ Tự Các Hàm Trong .c File

- Tên hàm cũng phải tuân theo nguyên tắc: cần phải được khai báo *trước* khi dùng.

```
#include <stdio.h>

void    fun2 (void) { ... }

void    fun1 (void) { ...; fun2(); ... }

int     main (void) { ...; fun1(); ... return 0; }
```

- Hàm **fun1** gọi hàm **fun2** do vậy hàm **fun2** cần phải được khai báo trước hàm **fun1**, etc.

Ví dụ về lỗi biến cục bộ

- Hàm có thể trả về con trỏ.

```
int* func_returns_pointer(void) ;
```

- Tuy nhiên sẽ là lỗi nếu trả về con trỏ của biến cục bộ.

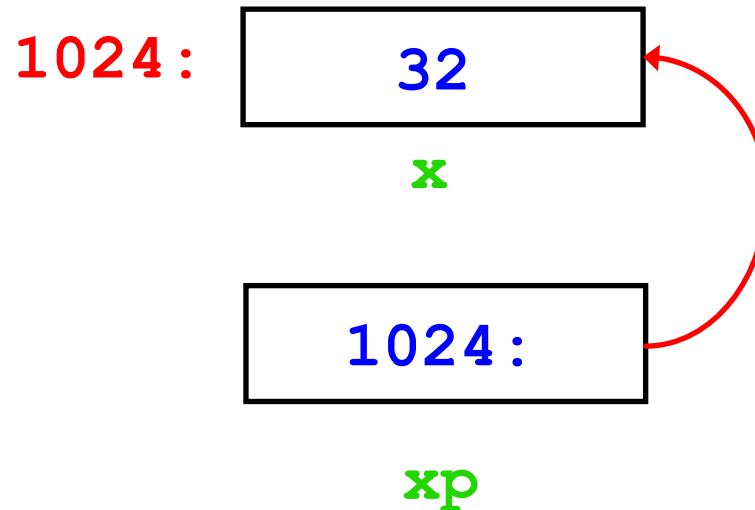
```
int* misguided(void)
{
    int array[10], i;
    for (i = 0; i < 10; ++i)
        array[i] = i;
    return array;
}
```

Cảnh Báo

- C cho phép bạn định nghĩa biến không nằm trong bất cứ hàm nào, còn gọi là *biến toàn cục*.
- Chú ý: ký hiệu **#define** không tạo ra biến.
- Biến toàn cục có thể được dùng thay cho việc truyền tham số cho hàm nhưng đây là một cách tồi, bạn nên có thói quen sử dụng biến cục bộ.

Kiểu Dữ Liệu Mới: Con Trỏ

- Con trỏ chứa tham chiếu đến một biến khác – có nghĩa là con trỏ có giá trị là địa chỉ của một biến khác.



- xp** là con trỏ tới một biến kiểu **int**.

Khai Báo Và Sử Dụng Con Trỏ

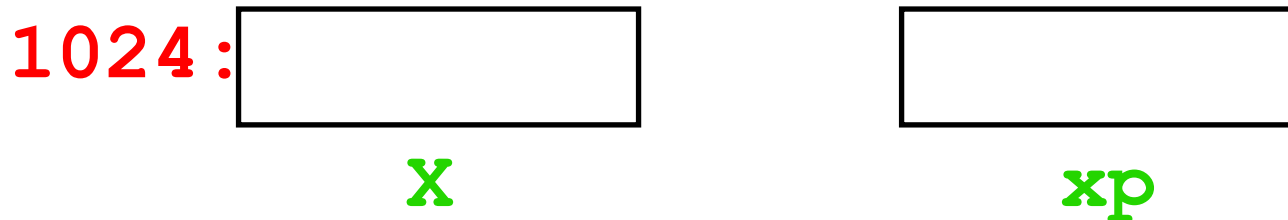
```
int x;           /* declares an int variable */
```

```
int *xp;         /* declares a pointer to int */
```

Giả sử **xp** lưu trữ địa chỉ của **x**, khi đó:

```
*xp = 0;         /* Assign integer 0 to x */
```

```
*xp = *xp + 1;   /* Add 1 to x */
```



Giải Pháp Dùng Con Trỏ Với `move_one`

```
void move_one (int *x_ptr, int *y_ptr) {  
    *x_ptr = *x_ptr - 1;  
    *y_ptr = *y_ptr + 1;  
}
```

```
int main (void) {  
    int a, b;  
    a = 4;  
    b = 7;  
    move_one (&a , &b) ;  
    printf ("%d %d", a, b) ;  
    return (0) ;  
}
```

Địa Chỉ Và Con Trỏ

- Ba kiểu mới dữ liệu mới
 - **int *** “con trỏ tới kiểu **int**”
 - **double *** “con trỏ tới kiểu **double**”
 - **char *** “con trỏ tới kiểu **char**”
- Hai toán tử (một ngôi) mới
 - **&**: toán tử lấy địa chỉ của biến
 - Có thể được dùng với mọi loại biến (hoặc tham biến)
 - *****: toán tử lấy giá trị lưu trong ô nhớ con trỏ trỏ tới
 - Chỉ được sử dụng bởi con trỏ

Quay Lại Hàm **scanf**

```
int x,y,z;
```

```
printf("%d %d %d", x, y, x+y);
```

```
scanf("%d %d %d", x, y, x+y);
```

NO!

```
scanf("%d %d", &x, &y);
```

YES! ?

Tại Sao Cần Dùng Con Trỏ?

- Các hàm cần thay đổi tham số thực trong lời gọi hàm
 - Ví dụ như hàm **move_one**
- Có thể trả về nhiều giá trị
 - Ví dụ như hàm **scanf**
- Trong lập trình cấp cao, con trỏ được dùng để tạo ra các cấu trúc dữ liệu động.

Sắp Xếp Hai Số Nguyên

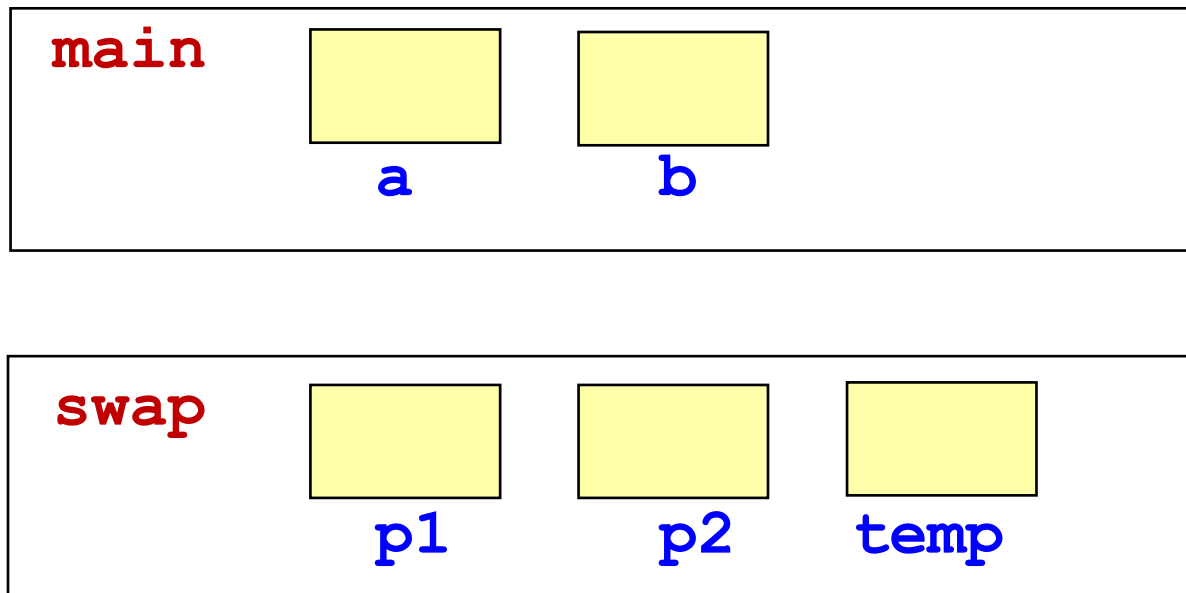
```
/* read in and sort 2 integers */
int c1, c2, temp;
printf("Enter 2 integers: ");
scanf("%d%d", &c1, &c2);
/* the 2 values may be in either order */
if (c2 < c1) {    /* swap if out of order */
    temp = c1;
    c1    = c2;
    c2    = temp;
}
/* at this point c1 <= c2 (guaranteed) */
```

Hàm **swap** Dùng Con Trỏ

```
void swap(int *p1, int *p2) {  
    int temp;  
    temp = *p1;  
    *p1  = *p2;  
    *p2  = temp;  
}
```

```
int a, b ;  
a = 4; b = 7;  
...  
swap(&a, &b) ;
```

Hàm **swap** Dùng Con Trỏ



Ngôn Ngữ C Định Kiểu Mạnh

```
int i;   int * ip;
double x; double * xp;
...
x = i;           /* no problem */
i = x;           /* not recommended */

ip = 30;         /* No way */
ip = i;          /* Nope */
ip = &i;         /* just fine */
ip = &x;         /* forget it! */
xp = ip;         /* bad */
&i = ip;         /* meaningless */
```

Mảng

- Định nghĩa: mảng là một tập hợp có thứ tự các giá trị có cùng kiểu dữ liệu được đặt tên.
- Ví dụ: điểm của 7 sinh viên
 - Đặt tên tập hợp: **grade**
 - Đánh số các phần tử: **0** đến **6**

double grade[7];	0	3.0
	1	3.8
		1.7
	.	2.0
	.	2.5
		2.1
	6	3.2

Biểu thức trong C:

grade[0] = 3.0;

grade[6] = 3.2;

2.0*grade[3] = 4.0;

...

Thuật Ngữ Mảng

```
type name[size];
```

Khai báo mảng

Kích thước mảng phải là một số nguyên

```
double grade[7];
```

- **grade** là *mảng số thực* với **7** thành phần (kích thước = 7)
- **grade[0], grade[1], ... , grade[6]** là các *thành phần* (element) của mảng **grade**. Từng thành phần đều có kiểu **double**.
- **0, 1, ... , 6** là các *chỉ số* của mảng
- Giá trị lớn nhất và nhỏ nhất của chỉ số là *ranh giới* (bound) của mảng (ở đây là 6 và 0)

Quy Tắc Với Chỉ Số Mảng

- Quy tắc: *Chỉ số mảng phải có giá trị là một số nguyên trong khoảng từ 0 đến $n-1$, trong đó n là số thành phần của mảng (không có ngoại lệ).*

- Ví dụ

grade[i+3+k] /* OK as long as $0 \leq i+3+k \leq 6$ */

- Chỉ số có thể đơn giản chỉ là một số nguyên

grade[0]

- Hoặc có thể là một biểu thức phức tạp

grade[(int) (3.1 * fabs(sin (2.0*PI*sqrt(29.067))))]

Kiểm Tra Ranh Giới Mảng

```
#define CLASS_SIZE 7
```

```
double grade[CLASS_SIZE];
```

```
int index;
```

```
index = 9;
```

```
...
```

```
grade[index] = 3.5; /* Out of range?? */
```

```
if (0<=index && index<CLASS_SIZE) {
```

```
    grade[index] = 3.5;
```

```
} else {
```

```
    printf("Index %d out of range.\n", index);
```

```
}
```

Ví Dụ Sử Dụng Thành Phần Mảng

```
double grade[7];   int i=3;   /*declarations*/

printf("Last two are %f, %f", grade[5],
grade[6]);

grade[5] = 0.0;

grade[i] = 2.0 * grade[i+1];

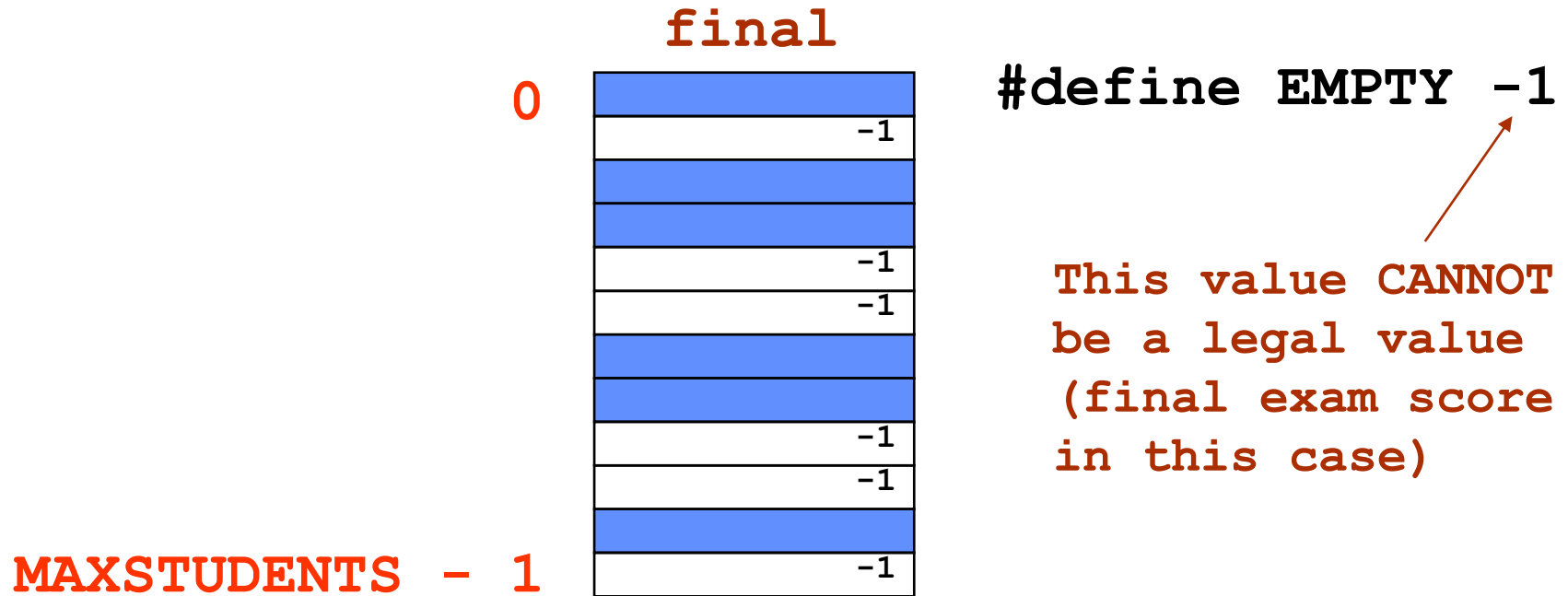
scanf("%lf", &grade[0]);

swap(&grade[i], &grade[i+1]);
```

Có Thể & Không Thể

- Bạn **không thể**
 - Dùng toán tử **=** để gán giá trị của một mảng này cho một mảng khác
 - Dùng toán tử **==** để so sánh trực tiếp hai mảng
 - Dùng câu lệnh **scanf** và **printf** trực tiếp với cả một mảng
- Nhưng bạn **có thể**
 - Thực hiện các thao tác trên với các thành phần mảng
 - Hoặc viết các hàm thực hiện các thao tác trên với mảng

Dùng Giá Trị Đặc Biệt



!!!!!!

```
for (student=0; student < MAXSTUDENTS; student++) {  
    if (final[student] != EMPTY) {  
        ...  
    }  
}
```

Dịch Chuyển Thành Phần Mảng

```
/* Shift x[0], x[1], ..., x[n-1] one  
   position upwards to make space for a new  
   element at x[0]. Insert the value new at  
   x[0]. Update the value of n. */
```

```
for (k = n; k >= 1; k = k - 1)
```

```
    x[k] = x[k-1];
```

```
x[0] = new;
```

```
n = n + 1;
```



Khởi Tạo Mảng

```
int w[4] = {1, 2, 30, -4};
```

```
/* w has size 4, all 4 are initialized */
```

```
char vowels[6] = {'a', 'e', 'i', 'o', 'u'};
```

```
/* vowels has size 6, only 5 have initializers */
```

```
/* vowels[5] is uninitialized */
```

- Bạn không thể dùng cách trên trong câu lệnh gán

```
w = {1, 2, 30, -4}; /* SYNTAX ERROR */
```

Khi Kích Thước Mảng Không Xác Định

```
double x[] = {1.0, 3.0, -15.0, 7.0, 9.0};  
/* x has size 5, all 5 are initialized */
```

- Tuy nhiên

```
double x[];      /* ILLEGAL */
```


Cả Mảng Làm Tham Số

```
#define ARRAY_SIZE 200

double average (int a[ARRAY_SIZE]) {
    int i, total = 0;
    for (i = 0; i < ARRAY_SIZE; i = i + 1)
        total = total + a[i];
    return ((double) total / ARRAY_SIZE);
}

int x[ARRAY_SIZE];

...

x_avg = average(x);
```

Mảng Làm Tham Số Trả Về

```
/* Sets vsum to sum of vectors a and b. */  
void VectorSum(int a[3], int b[3], int vsum[3]) {  
    int i;  
    for (i = 0; i < 3; i = i + 1)  
        vsum[i] = a[i] + b[i];  
}
```

Không có toán
tử * và &

```
int main(void) {  
    int x[3] = {1,2,3}, y[3] = {4,5,6}, z[3];  
    VectorSum(x,y,z);  
    printf("%d %d %d", z[0], z[1], z[2]);  
    reutrn 0;  
}
```

Mảng 2 Chiều

- Một tập hợp có trật tự các giá có cùng kiểu.
 - **Đặt tên** tập hợp, **đánh số** các thành phần trong tập hợp
- Ví dụ: điểm số của 7 sinh viên trong 4 bài kiểm tra

score	hw	0	1	2	3
student	0	22	15	25	25
student	1	12	12	25	20
student	2	5	17	25	24
student	3	15	19	25	13
student	4	2	0	25	25
student	5	25	22	24	21
student	6	8	4	25	12

Biểu thức trong C:

score[0][0] = 22;

score[6][3] = 12;

2***score**[3][0] = 30;

Khai Báo Mảng 2 Chiều

```
#define MAX_STUDENTS 80
```

```
#define MAX_HWS 6
```

```
...
```

```
int score[MAX_STUDENTS][MAX_HWS];
```

```
int nstudents, nhws, i, j;
```

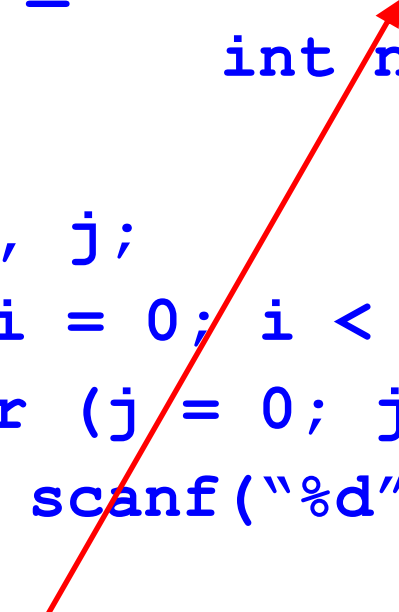
Nhập Mảng 2 Chiều

```
scanf("%d %d", &nstudents, &nhws);  
  
if (nstudents <= MAX_STUDENTS &&  
    nhws <= MAX_HWS) {  
    for (i = 0; i < nstudents; i = i + 1)  
        for (j = 0; j < nhws; j = j + 1)  
            scanf("%d", &score[i][j]);  
}
```

- Một phần mảng không được dùng đến.

Mảng 2 Chiều Làm Tham Số Hình Thức

```
void read_2D (int a[MAX_STUDENTS][MAX_HWS],
              int nstudents, int nhws)
{
    int i, j;
    for (i = 0; i < nstudents; i = i + 1)
        for (j = 0; j < nhws; j = j + 1)
            scanf("%d", &a[i][j]);
}
```



`int a[][MAX_HWS]`

`int (*a)[MAX_HWS]` : a pointer to an array of 13 integers

`int *a[MAX_HWS]` : an array of 13 pointers to integers

Mảng 2 Chiều Làm Tham Số Thực

```
int main(void)
{
    int score[MAX_STUDENTS][MAX_HWS];
    int nstudents, nhws;

    scanf("%d %d", &nstudents, &nhws);
    if (nstudents <= MAX_STUDENTS &&
        nhws <= MAX_HWS)
        read_2D (score, nstudents, nhws);
    ...
}
```



no &

Tóm Tắt Về Mảng

- Thành phần mảng
 - Giống như một biến có cùng kiểu với mảng, có thể được dùng làm tham số xuất/nhập
- Toàn bộ mảng
 - Không được truyền tham số cho mảng bằng cách copy
 - Tham số hình thức: *không có toán tử **
type array_name [SIZE]
type array_name []
 - Tham số thực: **array_name** , *không có toán tử [] và &*

Ký Tự Và Chuỗi Ký Tự

- Hằng ký tự: sử dụng `' '`

`'a', 'A', '0', '\n', ' ', 'i', 'l', '\0'`

- Hằng chuỗi ký tự: sử dụng `" "`

`"Bill"`

`"Mary had a little %c%c%c%c. \n"`

the null character



- Biến ký tự

```
char va = 'l', vb = 'a', vc = 'm', vd = 'b';
```

```
printf("Mary had a little  
      %c%c%c%c.\n", va, vb, vc, vd);
```

Chuỗi Ký Tự

- Chuỗi ký tự: mảng các ký tự

```
char pet[5] = { 'l', 'a', 'm', 'b', '\0' };  
printf("Mary had a little %s.\n", pet);
```

- Định nghĩa chính xác hơn: mảng các ký tự với ký tự kết thúc là ký tự null

pet:

'l'	'a'	'm'	'b'	'\0'
-----	-----	-----	-----	------

pet[0] **pet[4]**

- Chuỗi ký tự không phải là một kiểu dữ liệu trong C.
- Lập trình viên cần phải chắc chắn ký tự kết thúc là ký tự null **'\0'**.

Khởi Tạo Chuỗi

```
char pet[5] = { 'l', 'a', 'm', 'b', '\0' };
```

```
char pet[5];
```

```
pet[0] = 'l'; pet[1] = 'a'; pet[2] = 'm';
```

```
pet[3] = 'b'; pet[4] = '\0';
```

```
char pet[5] = "lamb";
```

```
char pet[ ] = "lamb";
```

tương
đương

- Chú ý không được dùng

```
char pet[5];
```

```
pet = "lamb"; /* No array assignment in C */
```

- Không được dùng câu lệnh gán để khởi tạo mảng trong ngôn ngữ C.

Có Thể Và Không Thể

- Bạn không thể
 - Dùng toán tử **=** để gán giá trị của một chuỗi ký tự này cho một chuỗi ký tự khác (hãy dùng hàm **strcpy** trong thư viện)
 - Dùng toán tử **==** để so sánh trực tiếp hai chuỗi ký tự (hãy dùng hàm **strcmp** trong thư viện)
 - Định nghĩa một hàm có kiểu trả về là chuỗi ký tự
- Bạn có thể
 - Trực tiếp xuất/nhập chuỗi ký tự sử dụng hàm **printf** và **scanf** (sử dụng điều khiển giữ chỗ là **%s**)

Tự Gán Chuỗi Ký Tự

```
char str1[10], str2[ ] = "Saturday";
int i;
/* can't do: str1 = str2; */
/* can do: */
i = 0;
while (str2[i] != '\0') {
    str1[i] = str2[i];
    i = i + 1;
}
str1[i] = '\0';
```

Sử Dụng Hàm **strcpy**

```
/* strcpy is defined in string.h:
   copy source string into dest, stop with \0 */
void strcpy(char dest[ ], char source[ ])
{
    int i = 0;
    while (source[i] != '\0') {
        dest[i] = source[i];
        i = i + 1;
    }
    dest[i] = '\0' ;
}
```

Nguy Hiêm Tiềm Ân

```
#include <string.h>

...

char medium[ ] = "Four score and seven";
char big[1000];
char small[5];
strcpy(big, medium);
strcpy(big, "Bob");
strcpy(small, big);
strcpy(small, medium);
/* looks like trouble... */
```

Kết Quả Sử Dụng Hàm **strcpy**

medium: Four score and seven\0

big: Four score and seven\0?????...

big: Bob\0 score and seven\0?????...

small: Bob\0?

small: Four score and seven\0

Tính Độ Dài Chuỗi Ký Tự: **strlen**

```
/* return the length of string s, i.e.,  
   number of characters before terminating '\0',  
   or equivalently, index of first '\0'.  
*/  
int strlen(char s[ ])  
{  
    int n = 0;  
    while (s[n] != '\0')  
        n = n + 1 ;  
    return (n) ;  
}
```

Ví Dụ Về Độ Dài Chuỗi Ký Tự

```
#include <string.h> /* defn of strlen, strcpy */
...
char pet[ ] = "lamb";
int len1, len2, len3, len4, len5;

len1 = strlen(pet);
len2 = strlen("wolf");
len3 = strlen("");
len4 = strlen("Help\n");
strcpy(pet, "cat");
len5 = strlen(pet);
```

	0	1	2	3	4	5	6
l a m b \0							
w o l f \0							
\0							
H e l p \n \0							
c a t \0 \0							

Nối Chuỗi Ký Tự

```
#include <string.h>
```

```
...
```

```
char str1[ ] = "lamb", str2[ ] = "chop";
```

```
char str3[11];
```

```
strcpy(str3, str1);
```

```
strcat(str3, str2);
```

```
/* strcat(s1, s2)
```

```
    make a copy of s2 at the end of s1. */
```

Kết Quả Sử Dụng Hàm **strcat**

str1

l a m b \0

str2

c h o p \0

str3

? ? ? ? ? ? ? ? ? ?

str3

l a m b \0 ? ? ? ? ? ?

str3

l a m b c h o p \0 ? ?

So Sánh Chuỗi Ký Tự

- Chuỗi **str_1** được coi là nhỏ hơn chuỗi **str_2** nếu
 - **j** là vị trí đầu tiên hai chuỗi khác nhau
 - Và **str_1[j] < str_2[j]**

“**lamb**” is less than “**wolf**” **j = 0, ‘l’ < ‘w’**

“**lamb**” is less than “**lamp**” **j = 3, ‘b’ < ‘p’**

“**lamb**” is less than “**lambch**” **j = 4, ‘\0’ < ‘c’**

Lỗi So Sánh Chuỗi Ký Tự

`str1 = str2;` **Syntax "error"**

`if (str1 == str2) ...` **No syntax error (but
almost surely a logic error)**

`if (str1 < str2) ...` **Likewise**

Hàm So Sánh Chuỗi Ký Tự

```
/* function strcmp in <string.h> */  
int strcmp(char str_1[ ], char str_2[ ]);
```

- Giá trị nguyên trả về có giá trị
 - Âm nếu **str_1** nhỏ hơn **str_2**
 - Bằng không nếu **str_1** bằng **str_2**
 - Dương nếu **str_1** lớn hơn **str_2**
- Lỗi thường gặp

```
if (!strcmp(str1, str2)) ...
```

means "if they ARE equal"

Xuất/Nhập Chuỗi Ký Tự

- **scanf** với điều khiển giữ chỗ “%s”
 - Bỏ qua ký tự trắng ở đầu
 - Chèn ký tự null ‘\0’ vào vị trí của ký tự trắng kế tiếp
 - Nguy hiểm tiềm ẩn: không kiểm tra độ dài chuỗi ký tự

char in_string[10];

scanStatus = scanf(“%s”, in_string);

- **printf** với điều khiển giữ chỗ “%s”



không sử
dụng &

Tự Nhập Cả Dòng Ký Tự

```
char line[LENGTH + 1];
int i, scanStatus;

/* read input characters into line until end of
   input line reached or all available space in
   line used */
i = 0;
scanStatus = scanf("%c", &line[i]);
while (1 == scanStatus && i < LENGTH &&
       line[i-1] != '\n') {
    i++;
    scanStatus = scanf("%c", &line[i]);
}
line[i] = '\0'; /* is this a bug? */
```

Mảng Chuỗi Ký Tự

```
char month[12][10] = {  
    "January",  
    "February",  
    ...  
    "September", /* longest month: 9 letters */  
    ...  
    "December" };  
...  
printf("%s is hot\n", month[7]); /* August */
```

Ví Dụ Nhập/Xuất Chuỗi Ký Tự

```
char name[NUM_NAMES][MAX_NAME + 1];
```

```
int age[NUM_NAMES], i;
```

```
for (i = 0; i < NUM_NAMES; i = i + 1)
```

```
{
```

```
    scanf("%s %d", name[i], &age[i]);
```

không sử
dụng &

```
    printf("%s %d \n", name[i], age[i]);
```

```
}
```

Rất Nhiều Hàm Trong **<string.h>**

strcat, strncat

nối

strcmp, strncmp

so sánh

strtod, strtol, strtoul

chuyển

- Các hàm hững dụng liên quan trong **<ctype.h>**
 - Hoạt động với từng ký tự đơn lẻ
 - Chuyển dạng ký tự, kiểm tra loại ký tự...

Sử Dụng Thư Viện Các Hàm

- Bạn nên sử dụng các hàm xử lý chuỗi ký tự có sẵn trong thư viện **string.h**
- Sử dụng các hàm có sẵn trong thư viện là một đặc trưng của lập trình với ngôn ngữ C
 - Các thư viện chuẩn ANSI C như **stdio.h**, **string.h**, **ctype.h**
 - Còn rất nhiều thư viện mở khác
 - Bạn thậm chí có thể tự tạo thư viện cho riêng mình
- Bạn khó có thể trở thành một lập trình viên tài ba nếu không có khả năng học nhanh cách sử dụng các thư viện mới

Ôn Lại: Cấu Trúc Dữ Liệu

- Hàm được dùng để tổ chức chương trình.
- Các cấu trúc dữ liệu được sử dụng để tổ chức dữ liệu, đặc biệt khi
 - Lượng dữ liệu lớn
 - Lượng dữ liệu thay đổi
 - Các tập dữ liệu có liên hệ với nhau
- Mảng có thể được dùng với trường hợp thứ nhất và thứ hai, không thể dùng trong trường hợp thứ ba
 - Dữ liệu mô tả một chiếc xe tăng: kiểu, màu sắc, vị trí
 - Thông tin về một sinh viên: tên, điểm, mã số sinh viên,...

Định Nghĩa **struct**

```
#define MAX_NAME 40
typedef struct{
/* typedefs go at the top of the program */
    char    name [MAX_NAME + 1];
    int     id;
    int     hw, exams;
    double  grade;
} student_record;
```

- Định nghĩa một kiểu dữ liệu mới **student_record**, không phải là khai báo hay tạo ra một biến mới, không cấp phát bộ nhớ.

Thuật Ngữ

- Một cấu trúc (*struct*) đôi khi được gọi là một bản ghi (*record*).
- Các thành phần (*component*) của cấu trúc còn được gọi là các trường (*field*) hay các thành viên (*member*) của cấu trúc.
- Cấu trúc là cơ sở của lớp (*class*) trong C++ và Java.

Các Kiểu Dữ Liệu Tự Định Nghĩa

- Cung cấp một tập hợp có giới hạn các kiểu dữ liệu có sẵn: **int**, **char**, **double** (và các biến thể).
- Bạn có thể bổ xung thêm thông qua việc dùng con trỏ và mảng.
- Tuy nhiên các đối tượng trong thế giới thực và trong các chương trình máy tính thường phức tạp và không thể được biểu diễn bằng các kiểu dữ liệu trên.
- Với **struct**, bạn có thể tự định nghĩa thêm các kiểu dữ liệu khi cần.

Khai Báo Biến Kiểu **struct**

```
/* typedef students_record goes at top of
program */

...

int    i1;                /* int decls. */
int    count = 0;         /* nothing new */
char   c1[5];             /* array decls. */

student_record s1;
student_record harvey;

/* student_record is a type; s1 and harvey are
variables. */
```

Có Thể Và Không Thể

- Bạn có thể
 - Dùng toán tử **=** để gán giá trị của hai biến có cùng kiểu cấu trúc
 - Định nghĩa một hàm có kiểu trả về là một cấu trúc
- Bạn không thể
 - Dùng toán tử **==** để so sánh trực tiếp hai biến có cùng kiểu cấu trúc (tuy nhiên bạn có thể so sánh các trường)
 - Trực tiếp xuất/nhập chuỗi ký tự sử dụng hàm **printf** và **scanf** (tuy nhiên có thể xuất/nhập từng trường)

Khởi Tạo Biến Kiểu **struct**

```
/*typedef structs go at top*/
```

```
int  i1;                /* int decls. */
int  count = 0;         /* nothing new */
char c1[5];             /* array decls. */
char pet[5] = "lamb";   /* string initializer */

student_record harvey = {"Harvey S.",
                          9501234, 87, 74, 3.1};
```

So Sánh struct

```
if (pt1 == pt2) { ... } /* Doesn't work */
```

```
int points_equal(point pt1, point pt2) {  
    return (pt1.x == pt2.x && pt1.y == pt2.y);  
}
```

```
if (points_equal(pt1, pt2)) { ... } /* OK */
```

Xuất/Nhập struct

```
void print_point(point p) {  
    printf("(%f,%f)", p.x, p.y);  
}  
void scan_point(point *ptptr) {  
    point temp;  
    scanf("%lf %lf", &temp.x, &temp.y);  
    *ptptr = temp;  
}
```

```
point a;  
scan_point(&a);  
print_point(a);
```

ptptr:

temp:

x:

y:

a:

x:

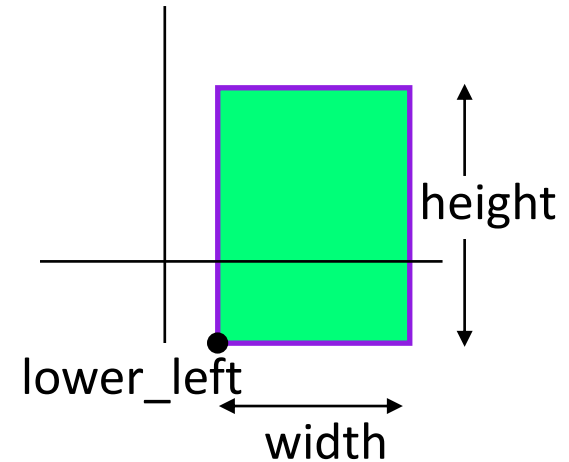
y:

struct Lồng Nhau

```
typedef struct {  
    double x, y;  
} point;
```

```
typedef struct {  
    double width, height;  
} dimension;
```

```
typedef struct {  
    dimension size;  
    point lower_left;  
    int line_color, fill_color;  
} rectangle;
```



Cấu Trúc Và Toán Học Con Trỏ

- Chương trình dịch sẽ tự động điền đầy lỗ trống, ví dụ cấu trúc sau 5-bytes, nhưng thực tế có thể là 6 or 8-bytes

```
struct Stype {  
    char c;  
    int i;  
};
```

- Toán tử **sizeof** luôn trả về kích thước đúng

Cấu Trúc Và Mảng

- Một cấu trúc thể hiện một bản ghi đơn nhất, các chương trình ứng dụng chạy trên máy tính làm việc với tập hợp các bản ghi.
- Ví dụ: bản ghi sinh viên, bản ghi nhân viên, bản ghi khách hàng...
- Trong mỗi trường hợp, sẽ có nhiều biến có cùng kiểu cấu trúc và do vậy sẽ rất tự nhiên nếu bạn dùng mảng cấu trúc để lưu giữ dữ liệu.

Mảng Cấu Trúc

- Mỗi khai báo phía dưới khai báo một mảng với các thành phần mảng có kiểu cấu trúc

```
point corner_points[10];
```

```
time meeting_times[MAX_MEETINGS];
```

```
student_record tdh_34[MAX_STUDENTS];
```

- Sử dụng mảng cấu trúc là một mở rộng tự nhiên các nguyên tắc đã được học.

Ôn Lại: Biến Kiểu Cấu Trúc Làm Đối Số

- Biến kiểu cấu trúc được truyền theo giá trị
 - Tất cả các thành phần của biến được sao chép từ đối số để khởi tạo tham số

```
point midpoint (point a; point b) {...}  
  
int main (void) {  
    point p1, p2, m; /* declare 3 points */  
    ...  
    m = midpoint(p1, p2);  
}
```

Truyền Mảng Cấu Trúc

- Một mảng cấu trúc trước hết là một mảng.
- Khi mảng được dùng làm đối số trong lời gọi hàm, nó được truyền theo tham chiếu
 - Tham số thực chất là một bí danh khác của đối số

```
int avg (student_rec class_db[MAX_N]) {...}

int main (void) {
    student_rec ktlt_k50[MAX_N];
    int average;
    ....
    average = avg(ktlt_k50); /*by reference*/
}
```

Sắp Xếp Mảng Cấu Trúc

David 920915 2.9	Kathryn 901028 4.0	Sarah 900317 3.9	Phil 920914 2.8	Casey 910607 3.6
------------------------	--------------------------	------------------------	-----------------------	------------------------

Phil 920914 2.8	David 920915 2.9	Casey 910607 3.6	Sarah 900317 3.9	Kathryn 901028 4.0
-----------------------	------------------------	------------------------	------------------------	--------------------------

```
typedef struct {  
    char    name[MAX_NAME + 1];  
    int     id;  
    double  score;  
} StudentRecord;
```

Ôn Lại: Sắp Xếp Lựa Chọn

```
int min_loc (int a[ ], int k, int n) {  
    int j, pos; pos = k;  
    for (j = k + 1; j < n; j = j + 1)  
        if (a[j] < a[pos])  
            pos = j;  
    return pos;  
}
```

```
void swap (int *x, int *y);
```

```
void sel_sort (int a[ ], int n) {  
    int k, m;  
    for (k = 0; k < n - 1; k = k + 1) {  
        m = min_loc(a, k, n);  
        swap(&a[k], &a[m]);  
    }  
}
```

Sắp Xếp Mảng Cấu Trúc

- Trước tiên cần xác định trường cần sắp xếp
 - Có thể sử dụng điểm số
- Thay đổi kiểu mảng sang **StudentRecord**
- Viết lại đoạn mã so sánh trong hàm **min_loc**
- Viết hàm **swap** cho **StudentRecord**

Sắp Xếp Mảng Cấu Trúc

```
int min_loc (StudentRecord a[ ], int k, int n) {  
    int j, pos; pos = k;  
    for (j = k + 1; j < n; j = j + 1)  
        if (a[j].score < a[pos].score)  
            pos = j;  
    return pos;  
}
```

```
void swap (StudentRecord *x, StudentRecord *y);
```

```
void sel_sort (StudentRecord a[ ], int n) {  
    int k, m;  
    for (k = 0; k < n - 1; k = k + 1) {  
        m = min_loc(a, k, n);  
        swap(&a[k], &a[m]);  
    }  
}
```


Sắp Xếp Theo Thứ Tự A-B-C

David 920915 2.9	Kathryn 901028 4.0	Sarah 900317 3.9	Phil 920914 2.8	Casey 910607 3.6
------------------------	--------------------------	------------------------	-----------------------	------------------------

Phil 920914 2.8	David 920915 2.9	Casey 910607 3.6	Sarah 900317 3.9	Kathryn 901028 4.0
-----------------------	------------------------	------------------------	------------------------	--------------------------

```
typedef struct {  
    char    name[MAX_NAME + 1];  
    int     id;  
    double  score;  
} StudentRecord;
```

- Cần viết một hàm để so sánh hai chuỗi ký tự.

Ôn Lại: So Sánh Chuỗi Ký Tự

- “Alice” nhỏ hơn “Bob”
- “Dave” nhỏ hơn “David”
- “Rob” nhỏ hơn “Robert”

```
#include <string.h>
```

```
int strcmp(char str1[ ], char str2[ ] );
```

- Giá trị trả về
 - Là số âm nếu **str1** nhỏ hơn **str2**
 - Bằng không nếu **str1** bằng **str2**
 - Là số dương nếu **str1** lớn hơn **str2**

Sắp Xếp Theo Thứ Tự A-B-C

```
int min_loc (StudentRecord a[ ], int k, int n) {  
    int j, pos; pos = k;  
    for (j = k + 1; j < n; j = j + 1)  
        if (0 > strcmp(a[j].name, a[pos].name))  
            pos = j;  
    return pos;  
}
```

```
void swap (StudentRecord *x, StudentRecord *y);
```

```
void sel_sort (StudentRecord a[ ], int n) {  
    int k, m;  
    for (k = 0; k < n - 1; k = k + 1) {  
        m = min_loc(a, k, n);  
        swap(&a[k], &a[m]);  
    }  
}
```

Một Chút Về Cấu Trúc Dữ Liệu

- Nếu bạn muốn lưu trữ thông tin về một bài hát trong máy tính
 - Những thông tin gì cần được lưu trữ?
 - Chúng được tổ chức như thế nào?
 - Cách thực hiện trong C?
- Còn nếu
 - Bạn muốn thông tin của một đĩa CD
 - Hoặc thông tin của một tập hợp các đĩa CD

Sử Dụng Sắp Xếp Chèn

```
/* sort student records a[0..size-1] in */  
/* ascending order by score */  
void sort (student_record a[ ], int size)  
{  
    int j;  
    for (j = 1; j < size; j = j + 1)  
        insert(a, j);  
}
```

Sử Dụng Sắp Xếp Chèn

```
/* given that a[0..j-1] is sorted, move a[j]
to the correct location so that that a[0..j]
is sorted by score */
void insert (student_record a[ ], int j) {
    int i;
    student_record temp;
    temp = a[j];
    for (i = j; i > 0 &&
        a[i-1].score > temp.score; i = i-1) {
        a[i] = a[i-1];
    }
    a[i] = temp;
}
```

Sử Dụng Sắp Xếp Chèn

```
/* given that a[0..j-1] is sorted, move a[j] to  
the correct location so that that a[0..j] is  
sorted by score */
```

```
void insert (student_record a[ ], int j) {  
    int i;  
    student_record temp;  
    temp = a[j];  
    for (i = j; i > 0 &&  
        strcmp(a[i-1].name, temp.name) > 0;  
        i = i-1) {  
        a[i] = a[i-1];  
    }  
    a[i] = temp;  
}
```

Kỹ Thuật Lập Trình

(Ngôn Ngữ Lập Trình C)

Bộ tiền xử lý

Bộ Tiền Xử Lý

- Các chỉ thị tiền xử lý xuất hiện trước khi chương trình được biên dịch.
- Các thao tác có thể thực hiện
 - Gộp thêm các tệp khác vào tệp đang biên dịch
 - Định nghĩa các hằng, macro
 - Biên dịch có điều kiện mã chương trình
 - Thực hiện có điều kiện các chỉ thị tiền dịch
- Các chỉ thị tiền dịch đều bắt đầu bằng ký tự #
 - Chỉ có các ký tự dấu cách được phép đứng trước các chỉ thị tiền xử lý trong dòng đó

Bộ Tiền Xử Lý

- Các chỉ thị tiền xử lý chỉ có ý nghĩa trong phạm vi file mà nó được định nghĩa.
- Chỉ thị tiền xử lý không phải là cú pháp cơ bản của C.
- Tuy nhiên việc sử dụng chúng có thể làm thay đổi cấu trúc của chương trình.

Chỉ Thị **#include**

- Chỉ thị tiền dịch **#include** thường được sử dụng với các file tiêu đề

#include <standard.h>

#include “myheader.h”

- Chỉ thị này thực hiện việc sao chép file vào vị trí của chỉ thị.

Định Nghĩa Các Hằng

- Sử dụng từ khóa **#define** để định nghĩa các hằng.
- Thường được dùng để loại bỏ “magic numbers \ hard code” trong mã nguồn.
- Dạng thường gặp

#define name text

- Trước khi chương trình được biên dịch, tất cả **name** sẽ được thay tự động bằng **text**

Định Nghĩa Các Hằng

```
#define BUFFERSIZE 256
```

```
#define MIN_VALUE -32
```

```
#define PI 3.14159
```

- Chú ý
 - Không có dấu `=` và dấu `;`
 - Tên được định nghĩa bởi **#define** có thể được loại bỏ bằng sử dụng **#undef** (có thể được định nghĩa lại sau đó)
- Trong thực tế, các biểu tượng hằng thường dùng chữ IN HOA để phân biệt nó với tên biến và tên hàm.

#define, const Và enum

- Có thể thay thế #define

```
#define ARRAYSIZE      10  
  
const int ArraySize = 10;  
  
double array[ARRAYSIZE]; /* Valid. */
```

```
#define PI              3.14159  
  
#define ARRAYSIZE      10  
  
const double Pi = 3.14159; /* Preferred */  
  
enum {ARRAYSIZE = 10};      /* Preferred */
```

Enum example

```
// example program to demonstrate working of enum in C
```

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

```
enum week{Mon, Tue, Wed, Thur, Fri, Sat,  
Sun};
```

```
int main()  
{  
    enum week day;  
    day = Wed;  
    printf("%d",day) ;  
    return 0;  
}
```

Enum example

```
// Another example program to demonstrate working
#include<stdio.h>

enum year{Jan, Feb, Mar, Apr, May, Jun, Jul,
          Aug, Sep, Oct, Nov, Dec};

int main()
{
    int i;
    for (i=Jan; i<=Dec; i++)
        printf("%d ", i);

    return 0;
}
```


Định Nghĩa Các Macro

- Lệnh **#define** thường được sử dụng trong việc tạo ra các macro

#define MAX(x,y) ((x)>(y) ? (x) : (y))

- Macro giống như hàm, nhưng thực chất không phải là hàm. Thực tế tên macro (MAX) sẽ được thay thế bằng dòng lệnh tương ứng với các đối số trước khi chương trình được biên dịch

int a = 4, b = -7, c;

c = MAX(a,b);

thay bằng:

c = ((a)>(b) ? (a) : (b));

Tại Sao Dùng Macro

- Tốc độ
 - Thực hiện như hàm nhưng sử dụng lời gọi hàm, đoạn mã được chèn vào trong chương trình trước khi biên dịch
 - Vấn đề này ít có ý nghĩa với chương trình viết trên PC
- Mã chung
 - Macro cho phép làm việc với mọi loại kiểu (**int**, **double**...)

```
int max(int x, int y) {  
    return x > y ? x : y;  
}
```

Ví Dụ Về Macro

```
#define SQR(x)          ((x) * (x))

#define SGN(x)          (((x)<0) ? -1 : 1)

#define ABS(x)          (((x)<0) ? -(x) : (x))

#define ISDIGIT(x)      ((x) >= '0' && (x) <= '9')

#define NELEMS(array)   sizeof(array)/sizeof(array[0])

#define CLAMP(val,low,high) \
((val)<(low) ? (low) : (val) > (high) ? (high) : (val))

#define ROUND(val) \
((val)>0 ? (int)((val)+0.5) : -(int)(0.5-(val)))
```

Một Vài Nhược Điểm

- Dễ dàng mắc lỗi với macro đơn giản.
- Ba nhược điểm chính
 - Sử dụng dấu ngoặc không chính xác
 - Dùng các toán tử `++`, `--`
 - Không kiểm tra kiểu

Cạm Bẫy Macro

```
#define SQR(x)  x * x
```

- Ví dụ

```
int a = 7;
```

```
b = SQR(a+1);
```

dẫn tới

```
b = a+1 * a+1;
```

- Giải pháp: dùng dấu ngoặc khi có thể

Cạm Bẫy Macro

- Ví dụ

b = ABS(a++);

trở thành

b = (((a++)<0) ? -(a++) : (a++));

- Giải pháp: không sử dụng các toán tử này trong macro

Cạm Bẫy Macro

- Không kiểm tra kiểu là con dao hai lưỡi, có thể dẫn tới các lỗi tính toán

```
int a = 7, b;
```

```
double c = 5.3, d;
```

```
d = SQR(a);
```

```
b = SQR(c);
```

Macro Chiếm Nhiều Dòng

- Bạn có thể viết các macro nhiều dòng với cuối dòng kết thúc bằng \

```
#define ERROR(condition, message) \  
    if (condition) printf(message)
```


Ví Dụ

```
#define TIMELOOP(CODE) { \  
    t0 = clock(); \  
    for (i = 0; i < n; ++i) { CODE; } \  
    printf("%7d ", clock() - t0); \  
}
```

Sử dụng như sau

```
TIMELOOP(y = sin(x));
```

Hằng Xâu Ký Tự Trong Macro

- Nếu đối số của macro có ký tự **#** đứng trước, thì đối số đấy sẽ được chuyển thành xâu ký tự hằng.

```
#define PRINT_DEBUG(expr) \  
    printf(#expr " = %g\n", expr)
```

Ví dụ:

```
PRINT_DEBUG(x/y) ;  
printf("x/y" " = %g\n", x/y) ;
```

Macro Có Sẵn

- Vài macro có sẵn trong chương trình dịch

__LINE__

__FILE__

__DATE__

__TIME__

__STDC__

Ví Dụ

```
#define PRINT_DEBUG(expr, type) \  
    printf(__FILE__ "[%d] (" #expr "): \  
           %" type##Conv "\n", __LINE__, (expr))
```

- Đoạn chương trình sau

```
#define intConv          "d"  
  
#define doubleConv      "f"  
  
PRINT_DEBUG(x/y, int);
```

Sẽ in ra tên file, thứ tự dòng và kết quả

Dịch Có Điều Kiện

- Các chỉ thị dịch có điều kiện trong C

`#if, #elif, #else, #endif`

`#ifdef, #ifndef`

- Mục đích
 - Thêm vào các đoạn mã gỡ lỗi chương trình
 - Thêm vào các đoạn mã không phải mã chuẩn
 - Tránh việc chèn các file header nhiều lần

Gỡ Lỗi

- Khi dịch chương trình trong chế độ debug

```
//#define DEBUG
```

- Bạn có thể chèn vào các đoạn mã gỡ lỗi

```
#ifdef DEBUG
```

```
    printf("Pointer %#x points  
           to value %f", pd, *pd);
```

```
#endif
```

Đoạn Mã Không Chuẩn

- Sử dụng trong trường hợp đoạn mã chỉ dùng cho các vi xử lý khác nhau

```
#ifdef __WIN32__
    return WaitForSingleObject(Handle, 0) ==
                                   WAIT_OBJECT_0;
#elif defined(__QNX__) || defined(__linux__)
    if (flock(fd, LOCK_EX | LOCK_NB) == -1)
        return 0;
    else
        return 1;
#endif
```

Chống Chèn File Nhiều Lần

- File header chỉ nên được chèn vào đúng một lần trong chương trình (mặc dù được nhiều file sử dụng).
- Chèn nhiều lần sẽ làm cho các biến, các hàm, các nhãn được định nghĩa lại và chương trình sẽ không thể dịch thành công
- Phương pháp chống: sử dụng “*header guards*”

```
#ifndef A_HEADER_H_
```

```
#define A_HEADER_H_
```

```
/* Contents of header file is here. */
```

```
#endif
```


Kỹ Thuật Lập Trình

(Ngôn Ngữ Lập Trình C)

Bài giảng số 3

Xử lý Date/Time trong ngôn ngữ C

Tại sao xử lý time/date

- Ví dụ về dữ liệu GPS

- \$GPGGA,123519,4807.038,N,01131.000,E,1,08,0.9,545.4,M,46.9,M,,*47
 - 123519 Fix taken at 12:35:19 UTC

- Log file data

C:\Program Files (x86)\EaseUS\Todo Backup\Agent.exe

2017-07-10 17:35:16 [M:00,T/P:1940/6300] Init Log

2017-07-10 17:35:16 [M:29,T/P:1940/6300] Ldq : Agent start install!

2017-07-10 17:35:16 [M:29,T/P:1940/6300] Ldq : Agent call CreateService!

2017-07-10 17:35:16 [M:29,T/P:1940/6300] Ldq : Agent call CreateService is success!

Các hàm cơ bản trong C với Time

```
// variables to store date and time components
int hours, minutes, seconds, day, month, year;

// time_t is arithmetic time type
time_t now;

// Obtain current time
// time() returns the current time of the system as a time_t value
time(&now);

// Convert to local time format and print to stdout
printf("Today is : %s", ctime(&now));

// localtime converts a time_t value to calendar time and
// returns a pointer to a tm structure with its members
// filled with the corresponding values
struct tm *local = localtime(&now);

hours = local->tm_hour;           // get hours since midnight (0-23)
minutes = local->tm_min;          // get minutes passed after the hour (0-59)
seconds = local->tm_sec;          // get seconds passed after minute (0-59)

day = local->tm_mday;             // get day of month (1 to 31)
month = local->tm_mon + 1;        // get month of year (0 to 11)
year = local->tm_year + 1900;     // get year since 1900

// print local time
if (hours < 12) // before midday
    printf("Time is : %02d:%02d:%02d am\n", hours, minutes, seconds);
else // after midday
    printf("Time is : %02d:%02d:%02d pm\n", hours - 12, minutes, seconds);

// print current date
printf("Date is : %02d/%02d/%d\n", day, month, year);
```

Sự dụng hàm make time

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

int main(void)
{
    struct tm str_time;
    time_t time_of_day;

    str_time.tm_year = 2012-1900;
    str_time.tm_mon = 6;
    str_time.tm_mday = 5;
    str_time.tm_hour = 10;
    str_time.tm_min = 3;
    str_time.tm_sec = 5;
    str_time.tm_isdst = 0;

    time_of_day = mktime(&str_time);
    printf(ctime(&time_of_day));

    return 0;
}
```

Time Zone

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

#define PST (-8)
#define CET (1)

int main ()
{
    time_t raw_time;
    struct tm *ptr_ts;

    time ( &raw_time );
    ptr_ts = gmtime ( &raw_time );

    printf ("Time Los Angeles: %2d:%02d\n",
            ptr_ts->tm_hour+PST, ptr_ts->tm_min);
    printf ("Time Amsterdam: %2d:%02d\n",
            ptr_ts->tm_hour+CET, ptr_ts->tm_min);

    printf ("Time Hanoi: %2d:%02d\n",
            ptr_ts->tm_hour+ 7, ptr_ts->tm_min);
    return 0;
}
```

Measure time taken in C?

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

// A function that terminates when enter key is pressed
void fun()
{
    printf("fun() starts \n");
    printf("Press enter to stop fun \n");
    while(1)
    {
        if (getchar())
            break;
    }
    printf("fun() ends \n");
}

// The main program calls fun() and measures time taken by fun()
int main()
{
    // Calculate the time taken by fun()
    clock_t t;
    t = clock();
    fun();
    t = clock() - t;
    double time_taken = ((double)t)/CLOCKS_PER_SEC; // in seconds

    printf("fun() took %f seconds to execute \n", time_taken);
    return 0;
}
```

Phương pháp số 2

```
#include <stdio.h>
#include <time.h>      // for time()
#include <unistd.h>     // for sleep()

// main function to find the execution time of a C program
int main()
{
    time_t begin = time(NULL);

    // do some stuff here
    sleep(3);

    time_t end = time(NULL);

    // calculate elapsed time by finding difference (end - begin)
    printf("Time elapsed is %d seconds", (end - begin));

    return 0;
}
```

Phương pháp số 3

```
#include <stdio.h>
#include <sys/time.h> // for gettimeofday()
#include <unistd.h>    // for sleep()

// main function to find the execution time of a C program
int main()
{
    struct timeval start, end;

    gettimeofday(&start, NULL);

    // do some stuff here
    sleep(5);

    gettimeofday(&end, NULL);

    long seconds = (end.tv_sec - start.tv_sec);
    long micros = ((seconds * 1000000) + end.tv_usec) - (start.tv_usec);

    printf("Time elapsed is %d seconds and %d micros\n", seconds, micros);

    return 0;
}
```

```
struct timeval {
    long tv_sec; /* seconds */
    long tv_usec; /* microseconds */
};
```


Kỹ Thuật Lập Trình

(Ngôn Ngữ Lập Trình C)

Làm việc với bit

Các Thao Tác Chính

- Cung cấp các toán tử
 - Cho phép thay đổi trực tiếp từng bit riêng lẻ
 - Cho phép thực hiện các phép toán mà thường chỉ có trong ngôn ngữ assembler
- Chương trình C làm việc với bit có thể chạy được trên các hệ điều hành khác nhau, tuy nhiên phần lớn các chương trình khi đã làm việc với bit đều liên quan đến các phần cứng riêng biệt.

Số Âm

- Bit MSB (most significant bit) có giá trị bằng 1 thì số đó gọi là số âm.
- Phương pháp bù 2 để tính số âm, ví dụ **-22425**
1010 1000 0110 0111
- Các bước tính bù 2
 - Lấy số ban đầu trừ đi 1: **22425** -> **22424**
 - Chuyển sang dạng nhị phân
0101 0111 1001 1000
 - Sau đó lấy bù 1
1010 1000 0110 0111

Toán Tử Làm Việc Với Bit

- C cung cấp 6 toán tử bit:

& | ^ ~ << >>

- Các toán tử này chỉ làm việc với các kiểu dữ liệu **char**, **short**, **int**, **long**.

- Không dùng cho dấu phẩy động

- Và 5 phép gán bit như sau

&= |= ^= <<= >>=

- Phép gán này tương tự với phép gán số học

z &= x | y;

z = z & (x | y);

Toán Tử Làm Việc Với Bit

- Lưu ý: không nên nhầm lẫn các toán tử bit với các toán tử logic

& | ~ << >>

&& || ! < >

AND &

- Thực hiện việc AND hai số nguyên theo từng bit.
- Ví dụ **b1, b2, b3** là các số **unsigned char**

b3 = b1 & b2;

b1 00011001 25

b2 01001101 & 77

b3 00001001 9

- Thường được sử dụng để
 - Reset bit
 - Chọn bit để kiểm tra

OR |

- Thực hiện việc OR hai số nguyên theo từng bit.
- Ví dụ **b1**, **b2**, **b3** là các số **unsigned char**

b3 = b1 | b2;

b1 00011001 25

b2 01101010 | 106

b3 01111011 123

- Thường được sử dụng để
 - Set một bit nào đó

XOR ^

- Thực hiện việc XOR (hoặc có loại trừ) hai số nguyên theo từng bit.
- Ví dụ **b1**, **b2**, **b3** là các số **unsigned char**

b3 = b1 ^ b2;

b1 00011001 25

b2 01001101 ^ 77

b3 01010100 84

- Thường được sử dụng để
 - Đảo trạng thái các bit được lựa chọn

NOT ~

- Thực hiện việc NOT (bù 1) một số nguyên theo từng bit.
- Ví dụ **b1**, **b2** là các số **unsigned char**

b2 = ~b1;

b1 00011001 25

b2 11100110 230

- Thường được sử dụng để
 - Lật trạng thái một nhóm bit

Dịch Trái <<

- Thực hiện việc dịch các bit của một số nguyên sang phía trái.
- Ví dụ **b1**, **b2** là các số **unsigned char**

b2 = b1 << 2;

b1 00011010 26

b2 01101000 104

- Lưu ý
 - Bít MSB mất, bit chèn vào LSB luôn có giá trị là 0
 - **b2 = b1*4**
 - Dịch trái thực hiện việc nhân **2ⁿ**.

Dịch Phải >>

- Hơi phức tạp hơn một chút: dịch các bit của một số nguyên sang phía phải.
 - Bit LSB luôn mất, bit chèn vào MSB có giá trị
 - Bằng 0 nếu thao tác trên số không dấu (**unsigned**)
 - Bằng 1 (dịch phải số học) hoặc 0 (dịch phải logic)
- ```
signed char x = -75; /* 1011 0101 */
signed char y = x>>2; /* 0010 1101 (logical) */
 /* 1110 1101 (arithmetic) */
```
- Kết quả phép dịch này tùy thuộc vào từng máy tính và từng hệ điều hành. Ví dụ ở trên là 45 đối với dịch logic và -19 với dịch số học. Thực tế luôn luôn sử dụng số không dấu cho dịch phải (tương đương với chia cho **2<sup>n</sup>**).

# Lũy Thừa 2

---

- Phép dịch bit thường được dùng thay cho phép nhân.
- Phép dịch bit có tốc độ thực hiện nhanh hơn phép nhân.

**x \* 2**

**x << 1**

**x / 16**

**x >> 4**

**x % 8**

**x & 7**

- Tuy nhiên việc này sẽ làm cho chương trình trở nên khó đọc hơn.

# Cảnh Báo

---

- Nếu bạn dịch bit với số lần dịch lớn hơn kích cỡ (**sizeof**) của toán tử thì kết quả nhận được thường không xác định.

# Các Toán Tử Điều Khiển

---

- Thứ tự ưu tiên của các toán tử điều khiển
  - NOT             $\sim$
  - AND            $\&$
  - XOR            $\wedge$
  - OR             $|$
- Tuy nhiên nên sử dụng dấu  $()$  trong mọi trường hợp.

# Ví dụ về checksum 8bit

---

```
#include <reg51.h>
void main(void)
{
 unsigned char
 mydata[]={0x25,0x62,0x3F,0x52};
 unsigned char sum=0, x
 unsigned char chksumbyte;
 for (x=0;x<4;x++)
 {
 P2=mydata[x];
 sum=sum+mydata[x];
 P1=sum;
 }
 chksumbyte=~sum+1;
 P1=chksumbyte;
}
```

```
#include <reg51.h>
void main(void)
{
 unsigned char mydata[]
 ={0x25,0x62,0x3F,0x52,0xE8};
 unsigned char shksum=0;
 unsigned char x;
 for (x=0;x<5;x++)
 chksum=chksum+mydata[x];
 if (chksum==0)
 P0='Good';
 else
 P0='Bad';
}
```

# Mặt Nạ Bit

---

- Các toán tử bit thường dùng vào 2 mục đích chính
  - Để tiết kiệm bộ nhớ bằng cách lưu các trạng thái cờ trung gian vào một byte
  - Để giao tiếp với thanh ghi phần cứng
- Yêu cầu trong cả hai trường hợp là có thể sửa đổi từng bit và kiểm tra trạng thái từng bit.
- C cho phép tạo ra các macro, dùng bật (set), tắt (reset) bit hoặc đảo trạng thái của bit đó, thường được gọi chung là mặt nạ (masking).



# Mặt Nạ Bit

---

- Bước 1: Tạo ra số nguyên để đại diện cho từng trạng thái của bit (hoặc nhóm bit).
- Ví dụ

```
enum {
 FIRST = 0x01, /* 0001 binary */
 SECND = 0x02, /* 0010 binary */
 THIRD = 0x04, /* 0100 binary */
 FORTH = 0x08, /* 1000 binary */
 ALL = 0x0f /* 1111 binary */
};
```

# Mặt Nạ Bit

---

- Một cách khác

```
enum {
 FIRST = 1 << 0,
 SECND = 1 << 1,
 THIRD = 1 << 2,
 FORTH = 1 << 3,
 ALL = ~(~0 << 4)
};
```

- Dòng cuối cùng thường dùng để bật tắt một nhóm bit

```
1111 1111 /* ~0 */
1111 0000 /* ~0 << 4 */
0000 1111 /* ~(~0 << 4) */
```

# Thao Tác Với Mặt Nạ Bit

---

```
unsigned flags = 0;
```

```
flags |= SECND | THIRD | FORTH; /* (1110) . */
```

```
flags &= ~(FIRST | THIRD); /* (1010) . */
```

```
flags ^= (THIRD | FORTH); /* (1100) . */
```

```
if ((flags & (FIRST | FORTH)) == 0)
```

```
 flags &= ~ALL; /* (0000) . */
```

- Toán tử `|` dùng để tổ hợp các mặt nạ; toán tử `~` dùng để đảo dấu tất cả các bit (mọi bit là 1 trừ những bit được che mặt nạ); `|=` dùng để set bits; `&=` dùng để reset bits; `^=` dùng để đảo dấu bits; `&` dùng để chọn bits (cho việc kiểm tra trạng thái).

# Macro Cho Từng Bit

---

```
#define BitSet(arg,posn) ((arg) | (1L << (posn)))
#define BitClr(arg,posn) ((arg) & ~(1L << (posn)))
#define BitFlp(arg,posn) ((arg) ^ (1L << (posn)))
#define BitTst(arg,posn) ((arg) & (1L << (posn)))
```

```
enum {FIRST, SECND, THIRD};
unsigned flags = 0;
```

```
flags = BitSet(flags, FIRST); /* Set first bit. */
flags = BitFlp(flags, THIRD); /* Toggle third bit. */
if (BitTst(flags, SECND) == 0) /* Test second bit. */
 flags = 0;
```

# Ví Dụ

---

- Thực hiện thuật toán hoán đổi giá trị hai biến sử dụng phép toán XOR

```
#define SWAP(a,b) {a^=b; b^=a; a^=b;}
```

# Union

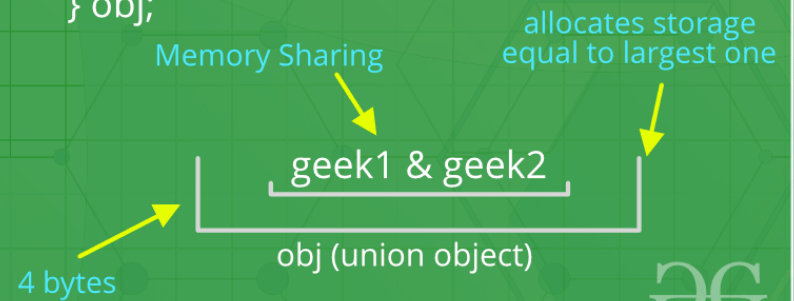
## Structure

```
struct Geeksforgeeks
{
 char X; //size 1 byte
 float Y; //size 4 byte
} obj;
```



## Unions

```
union Geeksforgeeks
{
 char X;
 float Y;
} obj;
```



# Ví dụ về Union

```
#include <stdio.h>

// Declaration of union is same as structures
union test {
 int x, y;
};

int main()
{
 // A union variable t
 union test t;

 t.x = 2; // t.y also gets value 2
 printf("After making x = 2:\n x = %d, y = %d\n\n",
 t.x, t.y);

 t.y = 10; // t.x is also updated to 10
 printf("After making y = 10:\n x = %d, y = %d\n\n",
 t.x, t.y);
 return 0;
}
```

## Output:

After making x = 2:

x = 2, y = 2

After making y = 10:

x = 10, y = 10

```
#include <stdio.h>

union test1 {
 int x;
 int y;
} Test1;

union test2 {
 int x;
 char y;
} Test2;

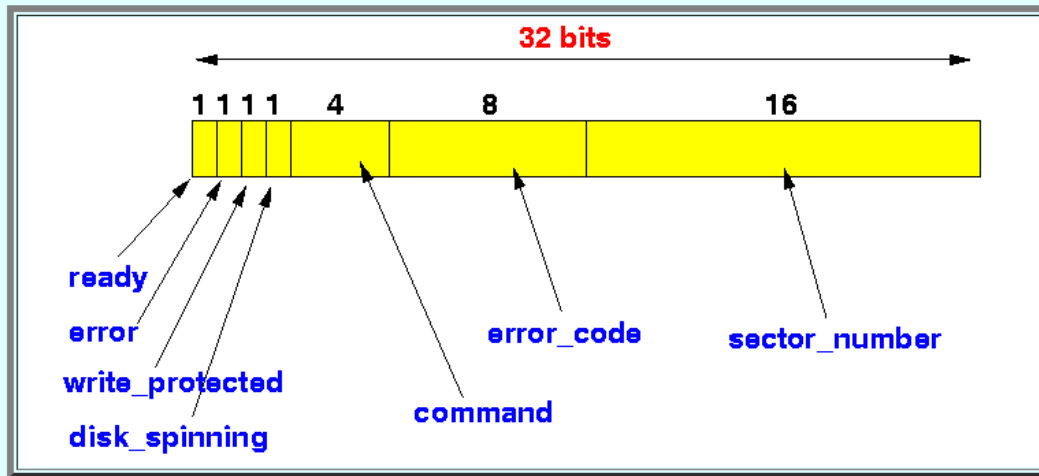
union test3 {
 int arr[10];
 char y;
} Test3;

int main()
{
 printf("sizeof(test1) = %lu, sizeof(test2) = %lu, "
 "sizeof(test3) = %lu",
 sizeof(Test1),
 sizeof(Test2), sizeof(Test3));
 return 0;
}
```

## Output:

sizeof(test1) = 4, sizeof(test2) = 4, sizeof(test3) = 40

# Bit Field



```
struct Disk_Register
{
 unsigned int ready:1 ; // 1 bit field named "ready"
 unsigned int error:1 ; // 1 bit field named "error"
 unsigned int wr_prot:1 ;
 unsigned int dsk_spinning:1 ;
 unsigned int command:4 ; // 4 bits field named "command"
 unsigned int error_code:8 ;
 unsigned int sector_no:16 ;
};
```

Giả sử ta có thanh ghi 32bit có những thông số như trên



# Ví dụ phần mềm

```
struct Disk_Register
{
 unsigned int ready:1 ; // 1 bit field named "ready"
 unsigned int error:1 ; // 1 bit field named "error"
 unsigned int wr_prot:1 ;
 unsigned int dsk_spinning:1 ;
 unsigned int command:4 ; // 4 bits field named "command"
 unsigned int error_code:8 ;
 unsigned int sector_no:16 ;
};

int main(int argc, char* argv[])
{
 struct Disk_Register r;

 printf("sizeof(r) = %d\n", sizeof(r)); // 4 bytes (32 bits)

 int* p = (int *) &r; // Access r as in int through pointer p

 *p = 0; // Clear all 32 bits in r !

 r.error = 1; // Set the error bit (bit #30)
 printBits(*p); // Call the printBits() function
 putchar('\n');

 r.dsk_spinning = 1; // Set the dsk_spinning bit (bit #28)
 printBits(*p); // Call the printBits() function
 putchar('\n');
}
```

# Làm việc với UNION

```
int* p = (int *) &r; // We need to CAST the
 // "address of struct DiskRegister"
 // to an "address of int (int *)"

*p = 0; // Clear all 32 bits in r !
```

```
/* -----
 Define the mapping of the 32 bits in the Disk Register
 ----- */
struct Disk_Register
{
 unsigned int ready:1 ; // 1 bit field named "ready"
 unsigned int error:1 ; // 1 bit field named "error"
 unsigned int wr_prot:1 ;
 unsigned int dsk_spinning:1 ;
 unsigned int command:4 ; // 4 bits field named "command"
 unsigned int error_code:8 ;
 unsigned int sector_no:16 ;
};

/* =====
 Re-map the 32 bits Disk Register AND a integer together
 ===== */
union U_Disk_Register
{
 struct Disk_Register Reg; // (1) 32 bits mapped as struct Disk_Register
 int Whole_Reg; // (2) 32 bits as one int
};
```

# Examples with Union

```
struct Disk_Register
{
 unsigned int ready:1 ; // 1 bit field named "ready"
 unsigned int error:1 ; // 1 bit field named "error"
 unsigned int wr_prot:1 ;
 unsigned int dsk_spinning:1 ;
 unsigned int command:4 ; // 4 bits field named "command"
 unsigned int error_code:8 ;
 unsigned int sector_no:16 ;
};

/* =====
Re-map the 32 bits Disk Register AND a integer together
===== */
union U_Disk_Register
{
 struct Disk_Register Reg; // (1) 32 bits mapped as struct Disk_Register
 int Whole_Reg; // (2) 32 bits as one int
};

int main(int argc, char* argv[])
{
 union U_Disk_Register r;

 printf("sizeof(r) = %d\n", sizeof(r)); // Still 4 bytes !!!

 r.Whole_Reg = 0; // Clear all 32 bits

 r.Reg.error = 1; // Set the error bit (bit #30)
 printBits(r.Whole_Reg); // Call the printBits() function
 putchar('\n');

 r.Reg.dsk_spinning = 1; // Set the dsk_spinning bit (bit #28)
 printBits(r.Whole_Reg); // Call the printBits() function
 putchar('\n');
}
```

# Tính khả chuyển (portability)

---

- Các chương trình dịch trên các Vi điều khiển có thể coi int là số 16bit, hoặc 32bit, tùy theo từng loại
- Các chương dịch C có thể có thứ tự
  - Từ trái sang phải
  - Từ phải sang trái

# Truyền dữ liệu

## Servomotor Command Summary

Command: X, seg#, data <CR> . Distance to be travelled. Data provided in encoder counts. (0 <= seg# 0 <= 23, -32768 <= data <= 32767)  
Command: A, seg#, data <CR> . Acceleration. Data provided in encoder counts/Tservo^2/65536 . (0 <= seg# 0 <= 23, -32768 <= data <= 32767)  
Command: V, seg#, data <CR> . Velocity Limit. Data provided in encoder counts/Tservo/256 . (0 <= seg# 0 <= 23, 1 <= data <= 32767)  
Command: T, seg#, data <CR> . Wait Time. Data in Tservo multiples . (0 <= seg# 0 <= 23, 0 <= data <= 32767)  
Command: G, startseg, stopseg <CR> . Execute a range of motion profiles segments. (0 <= startseg, stopseg 0 <= 23)  
Command: S <CR> . Stops execution of a motion profile  
Command: P, data <CR> . Change Proportional gain for PID algorithm. (-32768 <= data <= 32767)  
Command: D, data <CR> . Change Differential gain for PID algorithm. (-32768 <= data <= 32767)  
Command: W <CR> . Enable or disable the PWM driver stage

es. Type on the Virtual Terminal the following commands:

```
X,0,32000 <CR> V,0,100 <CR> A,0,200 <CR>
P, 3 <CR> D,-10 <CR>
G,0,0 <CR>
W <CR>
```

Temperature: 25.28 Humidity: 87 Power 111.4W

# Sscanf/Sprintf

---

```
// Example program to demonstrate sprintf()
#include<stdio.h>
int main()
{
 char buffer[50];
 int a = 10, b = 20, c;
 c = a + b;
 sprintf(buffer, "Sum of %d and %d is %d", a, b, c);

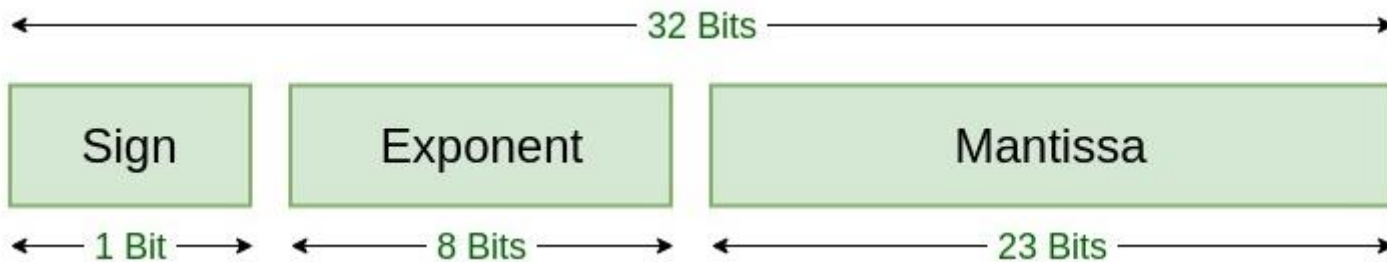
 // The string "sum of 10 and 20 is 30" is stored
 // into buffer instead of printing on stdout
 printf("%s", buffer);

 return 0;
}
```

```
#include<stdio.h>
int main()
{
 const char *str = "12345";
 int x;
 sscanf(str, "%d", &x);
 printf("\nThe value of x : %d", x);
 return 0;
}
```

# Ví dụ thực tế

---



Single Precision  
IEEE 754 Floating-Point Standard

# Chương trình Float to IEEE 32bit

```
#include <stdio.h>

void printBinary(int n, int i)
{
 // Prints the binary representation
 // of a number n up to i-bits.
 int k;
 for (k = i - 1; k >= 0; k--) {
 if ((n >> k) & 1)
 printf("1");
 else
 printf("0");
 }
}

typedef union {
 float f;
 struct
 {
 // Order is important.
 // Here the members of the union data structure
 // use the same memory (32 bits).
 // The ordering is taken
 // from the LSB to the MSB.
 unsigned int mantissa : 23;
 unsigned int exponent : 8;
 unsigned int sign : 1;
 } raw;
} myfloat;
```

```
// Function to convert real value
// to IEEE floating point representation
void printIEEE(myfloat var)
{
 // Prints the IEEE 754 representation
 // of a float value (32 bits)

 printf("%d | ", var.raw.sign);
 printBinary(var.raw.exponent, 8);
 printf(" | ");
 printBinary(var.raw.mantissa, 23);
 printf("\n");
}

// Driver Code
int main()
{
 // Instantiate the union
 myfloat var;
 // Get the real value
 var.f = -2.25;

 // Get the IEEE floating point representation
 printf("IEEE 754 representation of %f is : \n",
 var.f);
 printIEEE(var);

 return 0;
}
```



# Convert float to hex and hex to float

```
#include <stdio.h>

int main(void)
{
 union
 {
 float f;
 unsigned char b[sizeof(float)];
 } v ;

 v.f = 3.1415926535897932384626433832795F ;

 size_t i;
 printf("%.20f is stored as ", v.f);
 for (i = 0; i < sizeof(v.b); ++i)
 {
 printf("%02X%c", v.b[i], i < sizeof(v.b) - 1 ? '-' : '\n');
 }

 v.f = 0.0;

 v.b[0] = 0xdb;
 v.b[1] = 0x0f;
 v.b[2] = 0x49;
 v.b[3] = 0x40;

 printf ("PI value is %.20f \n", v.f);

 int i = 0;
 i = 2082;

 printf("i = %d \n", i);

 return 0;
}
```