BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC QUY NHƠN

Biên soạn
TS. Lê Thái Hiệp
ThS. Bùi Văn Vũ

TÀI LIỆU GIẢNG DẠY

TIN HỌC CƠ SỞ

(DÙNG CHO TRÌNH ĐỘ ĐẠI HỌC, CÁC NGÀNH KỸ THUẬT)

Bình Định, 2020

LỜI NÓI ĐẦU

1. Đặt vấn đề

Trong giai đoạn hiện nay cũng như trong tương lai, máy móc và robot sẽ làm thay con người trong rất nhiều công việc. Để máy móc và robot làm theo ý của con người thì phải lập trình cho các thiết bị điều khiển để điều khiển chúng.

Điều đó khẳng định vai trò quan trọng của việc lập trình trong thời đại hiện nay và trong tương lai. Do đó việc học Kỹ thuật lập trình là bắt buộc với người làm trong ngành kỹ thuật. Để tạo tiền đề cho việc học Kỹ thuật lập trình các sinh viên cần học về "Tin học cơ sở".

2. Mục tiêu của học phần

2.1. Mục tiêu đào tạo chung của học phần

- Kiến thức: Giúp sinh viên nắm vững các kiến thức cơ bản về lập trình và các giải thuật cơ bản nhất trong ngành kỹ thuật.

- Kỹ năng: Vận dụng tốt kiến thức để giải quyết các bài toán liên quan và xây dựng các ứng dụng cơ bản trong thực tế.

2.2. Mục tiêu đào tạo cụ thể về kiến thức của học phần

Trang bị các kiến thức thiết yếu để cho sinh viên có thể xây dựng được các chương trình ứng dụng cơ bản trong ngành kỹ thuật.

3. Tóm tắt nội dung học phần

Các kiến thức tổng quan về tin học.

Các kiến thức cơ bản về ngôn ngữ lập trình C / C++: cấu trúc chương trình C/C++; biến, hằng và kiểu dữ liệu; các vấn đề về nhập, xuất; các cấu trúc lệnh điều khiển.

Các thuật toán rất cơ bản trong ngành kỹ thuật.

MỤC LỤC

MỞ ĐẦU

i. Mục đích môn học

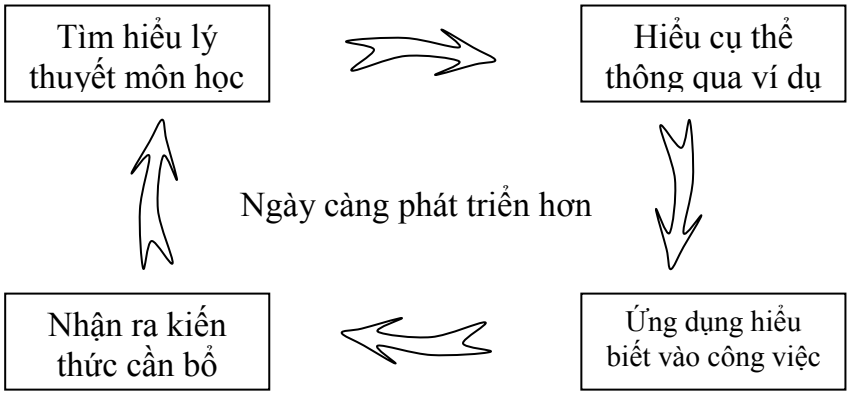
Cung cấp kiến thức cơ bản về ngôn ngữ lập trình C++. Áp dụng các kiến thức đã học vào các ứng dụng cơ bản trong các ngành kỹ thuật.

ii. Yêu cầu của môn học

Để học tốt môn này, ta cần hiểu biết cần thiết về:

+ Toán cao cấp;
+ Các vấn đề về kỹ thuật khác.

iii. Cách thức tiếp thu kiến thức từ môn học



iv. Một số nguyên tắc trong lập trình

Những nguyên tắc này được coi như nền tảng tư tưởng của phương pháp lập trình. Nắm vững các nguyên tắc không chỉ giúp người học có cách tiếp cận ngôn ngữ lập trình nhanh chóng mà con giúp họ cách tư duy trong khi xây dựng các hệ thống ứng dụng. Các nguyên tắc cơ bản được giới thiệu ở đây bao gồm:

- Nguyên tắc tối thiểu.
- Nguyên tắc địa phương.
- Nguyên tắc an toàn.
- Nguyên tắc nhất quán.

iv.1. Nguyên tắc tối thiểu

Hãy bắt đầu từ cốt lõi của vấn đề cần giải quyết với một số lượng tối thiểu các phương tiện là các cấu trúc lệnh, kiểu dữ liệu cùng các phép toán trên nó và thực hiện viết chương trình. Sau khi nắm chắc những công cụ cơ bản mới mở rộng sang hệ thống thư viện tiện ích của ngôn ngữ.

Khi làm quen với một ngôn ngữ lập trình nào đó, không nhất thiết phải lệ thuộc quá nhiều vào hệ thống thư viện hàm của ngôn ngữ, mà điều quan trọng hơn là trước một bài toán cụ thể, chúng ta sử dụng ngôn ngữ để giải quyết nó thế nào, và phương án tốt nhất là lập trình bằng chính hệ thống thư viện hàm của riêng mình. Do vậy, đối với các ngôn ngữ lập trình chẳng hạn như C++, chúng ta chỉ cần nắm vững một số các công cụ tối thiểu như sau:

iv.1.1. Tập các phép toán

Tập các phép toán số học: + (cộng); - (trừ); * (nhân); % (lấy phần dư); / (chia).

Tập các phép toán số học mở rộng:

++a <=> a = a +1; // tăng giá trị biến nguyên a lên một đơn vị;

--a <=> a = a-1; //giảm giá trị biến nguyên a một đơn vị;

a+= n <=> a = a+n; // tăng giá trị biến nguyên a lên n đơn vị;

a-=n <=> a = a - n; // giảm giá trị biến nguyên a n đơn vị);

a%=n <=> a = a%n; // lấy giá trị biến a modul với n;

a/=n <=> a=a/n;// lấy giá trị biến a chia cho n;

a*=n <=> a = a*n; // lấy giá trị biến a nhân với n;

Tập các phép toán so sánh: >, <, >=, <=, ==, != (lớn hơn, nhỏ hơn, lớn hơn hoặc bằng, nhỏ hơn hoặc bằng, đúng bằng, khác).

Tập các phép toán logic: &&, ||, ! (và, hoặc, phủ định)

Các toán tử thao tác bít (không sử dụng cho float và double)

& : Phép hội các bít.

| : Phép tuyển các bít.

^ : Phép tuyển các bít có loại trừ.

<< : Phép dịch trái (dịch sang trái n bít giá trị 0)

>> : Phép dịch phải (dịch sang phải n bít có giá trị 0)

~ : Phép lấy phần bù.

Toán tử chuyển đổi kiểu: Ta có thể dùng toán tử chuyển đổi kiểu để nhận được kết quả tính toán như mong muốn. Quy tắc chuyển đổi kiểu được thực hiện theo quy tắc: (kiểu) biến.

Thứ tự ưu tiên các phép toán : Khi viết một biểu thức, chúng ta cần lưu ý tới thứ tự ưu tiên tính toán các phép toán.

iv.1.2. Tập các lệnh vào ra cơ bản

Nhập dữ liệu từ bàn phím: *scanf*(“format_string, . . .”, &parameter . . .);

Nhập dữ liệu từ tệp: *fscanf*(file_pointer,”format_string, . . .”, &parameter, . . .);

Nhận một ký tự từ bàn phím: *getch*(); *getchar*();

Nhận một ký tự từ file: *fgetc*(file_pointer, character_name);

Nhập một string từ bàn phím: *gets*(string_name);

Nhận một string từ file text : *fgets*(string_name, number_character, file_pointer);

Xuất dữ liệu ra màn hình: *printf*(“format_string . . .”, parameter . . .);

Xuất dữ liệu ra file : *fprintf*(file_pointer, “format_string . . .”, parameter. . .);

Xuất một ký tự ra màn hình: *putch*(character_name);

Xuất một ký tự ra file: *fputc*(file_pointer, character_name);

Xuất một string ra màn hình: *puts*(const_string_name);

Xuất một string ra file: *fputs*(file_pointer, const_string_name);

iv.1.3. Thao tác trên các kiểu dữ liệu có cấu trúc

Tập thao tác trên string:

char **strchr*(const char *s, int c) : tìm ký tự c đầu tiên xuất hiện trong xâu s;

char **stpcpy*(char *dest, const char *src) : copy xâu scr vào dest;

int *strcmp*(const char *s1, const char *s2) : so sánh hai xâu s1 và s2 theo thứ tự từ điển, nếu s1 < s2 thì hàm trả lại giá trị nhỏ hơn 0. Nếu s1>s2 hàm trả lại giá trị dương. Nếu s1==s2 hàm trả lại giá trị 0.

char **strcat*(char *dest, const char *src) : thêm xâu scr vào sau xâu dest.

char **strlwr*(char *s) : chuyển xâu s từ ký tự in hoa thành ký tự in thường.

char **strupr*(char *s): chuyển xâu s từ ký tự thường hoa thành ký tự in hoa.

char **strrev*(char *s): đảo ngược xâu s.

char **strstr*(const char *s1, const char *s2): tìm vị trí đầu tiên của xâu s2 trong xâu s1.

int *strlen*(char *s): cho độ dài của xâu ký tự s.

Tập thao tác trên con trỏ:

Thao tác lấy địa chỉ của biến: & parameter_name;

Thao tác lấy nội dung biến (biến có kiểu cơ bản): *pointer_name;

Thao tác trỏ tới phần tử tiếp theo: ++pointer_name;

Thao tác trỏ tới phần tử thứ n kể từ vị trí hiện tại: pointer_name = pointer_name +n;

Thao tác trỏ tới phần tử sau con trỏ kể từ vị trí hiện tại: --pointer_name;

Thao tác trỏ tới phần tử sau n phần tử kể từ vị trí hiện tại:

pointer_name = pointer_name - n;

Thao tác cấp phát bộ nhớ cho con trỏ:

void **malloc*(size_t size);

void **calloc*(size_t nitems, size_t size);

Thao tác cấp phát lại bộ nhớ cho con trỏ : void **realloc*(void *block, size_t size);

Thao tác giải phóng bộ nhớ cho con trỏ: void *free*(void *block);

Tập thao tác trên cấu trúc:

Định nghĩa cấu trúc:

```
struct struct_name{
  type_1 parameter_name_1;
  type_2 parameter_name_2;
  . . . . . .
  type_k parameter_name_k;
} struct_parameter_name;
```

Phép truy nhập tới thành phần cấu trúc: struct_parameter_name.parameter_name.

Phép gán hai cấu trúc cùng kiểu: struct_parameter_name1 = struct_parameter_name2;

Phép tham trỏ tới thành phần của con trỏ cấu trúc:

pointer_struct_parameter_name -> struct_parameter_name.

Tập thao tác trên file:

Khai báo con trỏ file: FILE * file_pointer;

Thao tác mở file theo mode: FILE **fopen*(const char *filename,const char *mode);

Thao tác đóng file: int *fclose*(FILE *stream);

Thao tác đọc từng dòng trong file: char **fgets*(char *s, int n, FILE *stream);

iv.2. Nguyên tắc địa phương

Các biến địa phương trong hàm, thủ tục hoặc chu trình cho dù có trùng tên với biến toàn cục thì khi xử lý biến đó trong hàm hoặc thủ tục vẫn không làm thay đổi giá trị của biến toàn cục.

Tên của các biến trong đối số của hàm hoặc thủ tục đều là biến hình thức. Mọi biến hình thức truyền theo trị cho hàm hoặc thủ tục đều là các biến địa phương.

Các biến khai báo bên trong các chương trình con, hàm hoặc thủ tục đều là biến địa phương.

Khi phải sử dụng biến phụ nên dùng biến địa phương và hạn chế tối đa việc sử dụng biến toàn cục để tránh xảy ra các hiệu ứng phụ.

iv.3. Nguyên tắc nhất quán

Dữ liệu thế nào thì phải thao tác thế ấy. Cần sớm phát hiện những mâu thuẫn giữa cấu trúc dữ liệu và cách thao tác để kịp thời khắc phục.

Như chúng ta đã biết, kiểu là một tên chỉ tập các đối tượng thuộc miền xác định cùng với những thao tác trên nó. Một biến khi định nghĩa bao giờ cũng thuộc một kiểu xác định nào đó hoặc là kiểu cơ bản hoặc kiểu do người dùng định nghĩa. Thao tác với biến phụ thuộc vào những thao tác được phép của kiểu. Hai kiểu khác nhau được phân biệt bởi tên, miền

xác định và các phép toán trên kiểu dữ liệu. Tuy nhiên, trên thực tế có nhiều lỗi nhập nhằng giữa phép toán và cấu trúc dữ liệu.

iv.4. Nguyên tắc an toàn

Lỗi nặng nhất nằm ở mức cao nhất là mức ý đồ thiết kế và ở mức thấp nhất là thủ tục phải chịu tải lớn nhất.

Mọi lỗi, dù là nhỏ nhất cũng phải được phát hiện ở một bước nào đó của chương trình. Quá trình kiểm tra và phát hiện lỗi phải được thực hiện trước khi lỗi đó gây hậu quả lớn.

Các loại lỗi thường xảy ra trong khi viết chương trình có thể được tổng kết lại như sau:

Lỗi được thông báo bởi từ khoá error (lỗi cú pháp): loại lỗi này thường xảy ra trong khi soạn thảo chương trình, chúng ta có thể viết sai các từ khoá ví dụ thay vì viết là int chúng ta soạn thảo sai thành Int (lỗi chữ in thường thành in hoa), hoặc viết sai cú pháp các biểu thức như thiếu các dấu ngoặc đơn, ngoặc kép hoặc dấu chấm phảy khi kết thúc một lệnh, hoặc chưa khai báo nguyên mẫu cho hàm .

Lỗi được thông báo bởi từ khoá Warning (lỗi cảnh báo): lỗi này thường xảy ra khi ta khai báo biến trong chương trình nhưng lại không sử dụng tới chúng, hoặc lỗi trong các biểu thức kiểm tra khi biến được kiểm tra không xác định được giá trị của nó, hoặc lỗi do thứ tự ưu tiên các phép toán trong biểu thức. Hai loại lỗi error và warning được thông báo ngay khi dịch chương trình thành file *.OBJ. Quá trình liên kết (linker) các file *.OBJ để tạo nên file chương trình mã máy *.EXE chỉ được tiếp tục khi chúng ta hiệu đính và khử bỏ mọi lỗi error.

Lỗi xảy ra trong quá trình liên kết: lỗi này thường xuất hiện khi ta sử dụng tới các lời gọi hàm, nhưng những hàm đó mới chỉ tồn tại dưới dạng nguyên mẫu (function prototype) mà chưa được mô tả chi tiết các hàm, hoặc những lời hàm gọi chưa đúng với tên của nó. Lỗi này được khắc phục khi ta bổ sung đoạn chương trình con mô tả chi tiết cho hàm hoặc sửa đổi lại những lời gọi hàm tương ứng.

Ta quan niệm, lỗi cú pháp (error), lỗi cảnh báo (warning) và lỗi liên kết (linker) là lỗi tầm thường vì những lỗi này đã được Compiler của các ngôn ngữ lập trình phát hiện được.

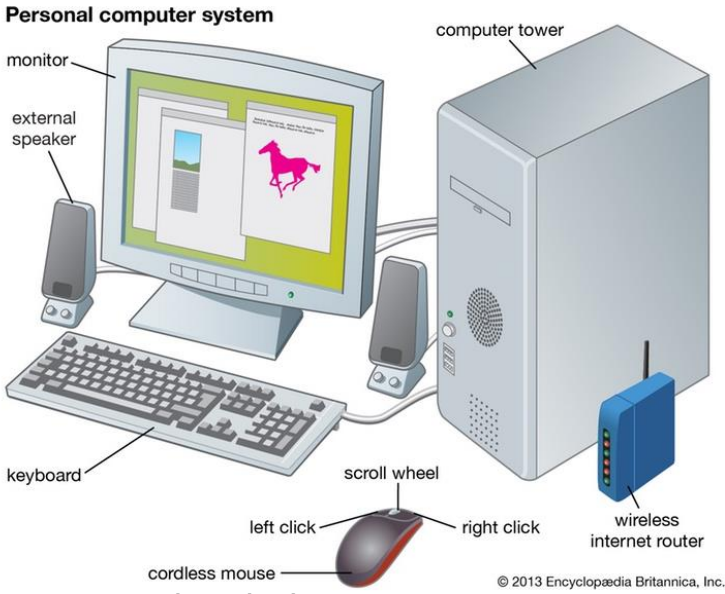
Để khắc phục các lỗi loại này, chúng ta chỉ cần phải đọc và hiểu được những thông báo lỗi. Cũng cần phải lưu ý rằng, do mức độ phức tạp của chương trình dịch nên không phải lỗi nào cũng được chỉ ra một cách tường minh và chính xác hoàn toàn tại nơi xuất hiện lỗi.

Loại lỗi cuối cùng mà các compiler không thể phát hiện nổi đó là *lỗi do chính lập trình viên gây nên trong khi thiết kế chương trình và xử lý dữ liệu*. Những lỗi này không được compiler thông báo mà nó phải trả giá bằng quá trình tự test hoặc chứng minh được tính đúng đắn của chương trình. Lỗi có thể nằm ở chính ý đồ thiết kế, hoặc lỗi do không lường trước được tính chất của mỗi loại thông tin vào.

PHẦN 1. CƠ SỞ LÝ THUYẾT

Chương 1. TỔNG QUAN VỀ TIN HỌC

1.1. Cấu trúc máy tính



Hình 1. 1. Các thiết bị kết nối trong một cụm máy tính cá nhân (PC)

Thiết bị nhập

Thiết bị nhập (Input Devices) là những thiết bị nhập dữ liệu vào máy tính như bàn phím, chuột, webcam, scaner…

Thiết bị xử lý

Thiết bị xử lý (Processing Devies) là thiết bị xử lý dữ liệu, quản lý điều khiển các hoạt động của máy tính thường được gọi là CPU (Central Processing Unit).

Bộ nhớ và thiết bị lưu trữ

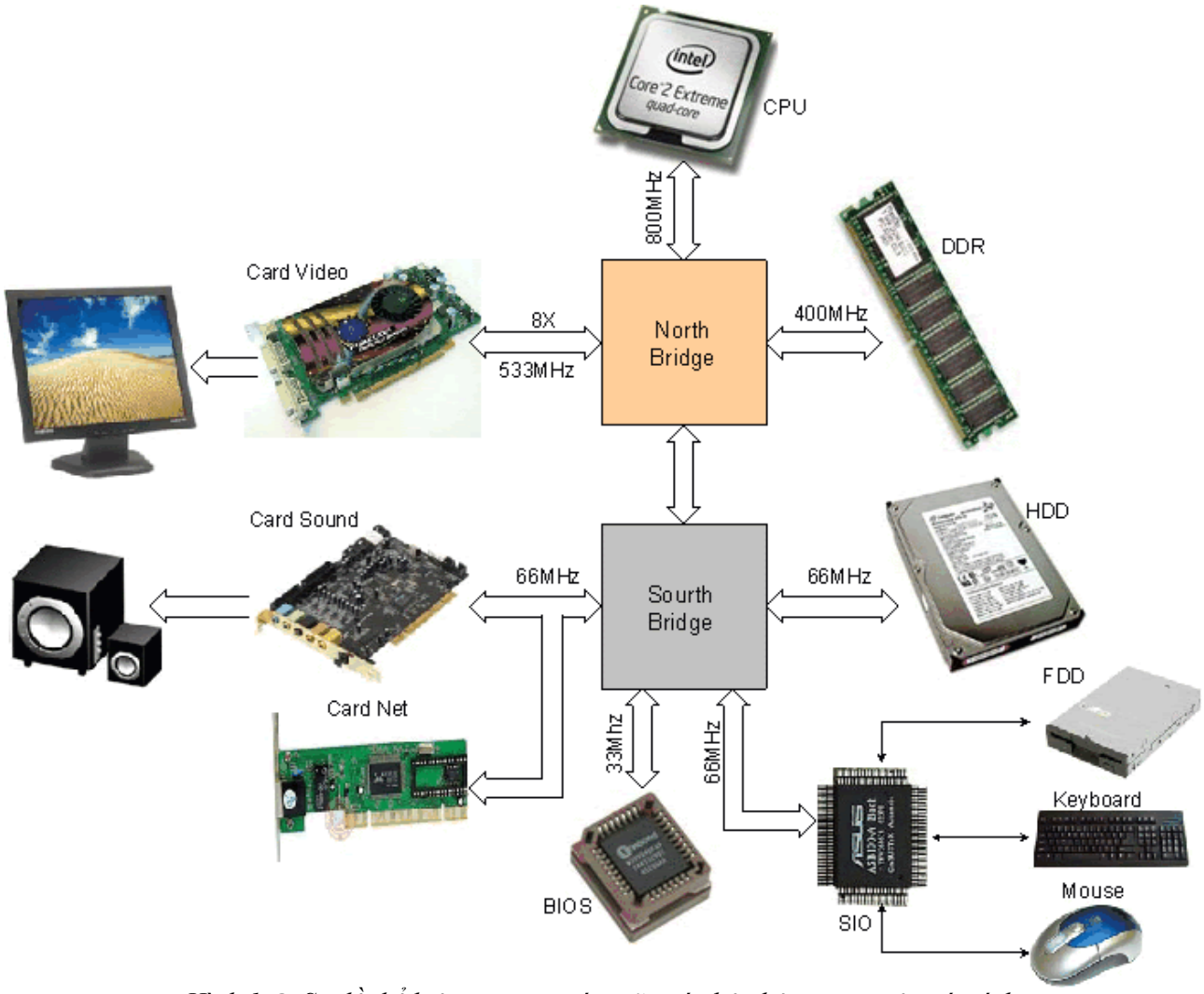
Thiết bị lưu trữ và bộ nhớ (Memory and Storage Devices) là những thiết bị lưu trữ dữ liệu tạm thời hay cố định những thông tin dữ liệu của máy tính bao gồm bộ nhớ trong và bộ nhớ ngoài.

Bộ nhớ trong bao gồm: bộ nhớ cache và bộ nhớ chính (gồm bộ nhớ chỉ đọc ROM, bộ nhớ truy xuất ngẫu nhiên RAM).

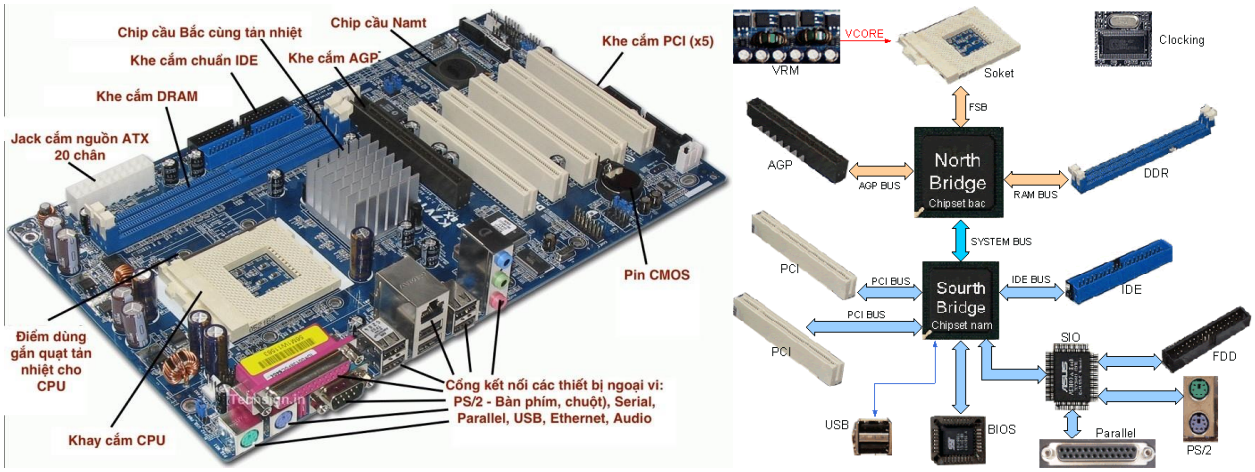
Bộ nhớ ngoài bao gồm ổ cứng (HDD - Hard Disk Drive, SDD - Solid State Drive), đĩa mềm (FDD - Floppy Disk Drive), đĩa CD, DVD, ổ USB, thẻ nhớ và các thiết bị lưu trữ khác. Hiện nay FDD, CD, DVD không còn được sử dụng phổ biến, mà chỉ còn sử dụng trong vài trường hợp đặc biệt. Thay vào đó việc lưu trữ trên internet (dữ liệu được lưu trữ ở các server) ngày càng phổ biến.

Thiết bị xuất

Thiết bị xuất (Output Devices) là những thiết bị hiển thị và xuất dữ liệu từ máy như màn hình, máy in, loa, máy chiếu (projector)…



Hình 1. 2. Sơ đồ thể hiện sự tương tác giữa các bộ phận trong một máy tính.



Hình 1. 3. Manboard và sơ đồ khe cắm trên mainboard.

Bo mạch chủ (Mainboard)

Bo mạch chủ là bảng mạch chính và lớn nhất trong cấu trúc máy tính, nó đóng vai trò là trung gian giao tiếp giữa các thiết bị với nhau. Việc kết nối và điều khiển thông thường là do các chip cầu Bắc và cầu Nam, chúng là trung tâm điều phối các hoạt động của máy vi tính.

Soket (đế cắm CPU): Có nhiều loại đế cắm cho CPU tuỳ theo chủng loại Mainboard:

SOCKET	SỐ LƯỢNG PIN	NGÀY PHÁT HÀNH	TƯƠNG THÍCH VỚI CPU
Socket 0	168	1989	✧ 486 DX
LGA775(Socket T)	775	Tháng 8/2004	✧ Pentium 4 (LGA775) ✧ Pentium 4 Extreme Edition (LGA775) ✧ Pentium DPentium Extreme EditionCeleron D (LGA 775) ✧ Core 2 Duo ✧ Core 2 QuadCore 2 ✧ ExtremePentium Dual Core ✧ Pentium E6000 series
LGA1155(Socket H2)	1.155	Tháng 1/2011	✧ Core i3 2000 series ✧ Core i3 3000 series ✧ Core i5 2000 series ✧ Core i5 3000 series ✧ Core i7 2000 series ✧ Core i7 3000 series ✧ Pentium G600 series ✧ Pentium G800 series ✧ Pentium G2000 series ✧ Celeron G400 series ✧ Celeron G500 series
LGA1156(Socket H1)	1.156	Tháng 9/2009	✧ Core i3 500 series ✧ Core i5 600 series ✧ Core i5 700 series ✧ Core i7 800 series ✧ Pentium G6900 series ✧ Celeron G1101
LGA1366(Socket B)	1.366	Tháng 9/2009	✧ Core i7 900 series ✧ Celeron P1053
LGA2011(Socket R)	2.011	Tháng 11/2011	✧ Core i7 3800 series ✧ Core i7 3900 series

- *North Bridge (Chipset bắc):* Chipset bắc có nhiệm vụ điều khiển các thành phần có tốc độ cao như CPU, RAM và Card Video. Chipset điều khiển về tốc độ BUS và điều khiển chuyển mạch dữ liệu, đảm bảo cho dữ liệu qua lại giữa các thành phần được thông suốt và liên tục, khai thác hết được tốc độ của CPU và bộ nhớ RAM
- *Sourth Bridge (Chipset nam):* Chức năng của chipset nam tương tụ như chipset bắc, nhưng chipset nam điều khiển các thành phần có tốc độ chậm như: Card Sound, Card Net, ổ cứng, ổ CD ROM, các cổng USB, IC SIO và BIOS v v...

- *ROM BIOS (Read Olly Memory - Basic In Out System):* ROM là IC nhớ chỉ đọc, BIOS là chương trình nạp trong ROM do nhà sản xuất Mainboard nạp vào, chương trình BIOS có các chức năng chính sau đây:

+ Khởi động máy tính, duy trì sự hoạt động của CPU
+ Kiểm tra lỗi của bộ nhớ RAM và Card Video
+ Quản lý trình điều khiển cho chipset bắc, chipset nam, IC-SIO và card video onboard
+ Cung cấp bản cài đặt CMOS SETUP mặc định để máy có thể hoạt động ta chưa thiết lập CMOS

- *Mạch tạo xung Clock:* Mạch tạo xung Clock có vai trò quan trọng trên Main, chúng tạo xung nhịp cung cấp cho các thành phần trên Main hoạt động. Đồng thời đồng bộ sự hoạt động của toàn hệ thống máy tính. Nếu mạch Clock bị hỏng thì các thành phần trên Main không thể hoạt động được, mạch Clocking hoạt động đầu tiên sau khi Main có nguồn chính cung cấp.

VRM (Vol Regu Module) Modul ổn áp: Đây là mạch điều khiển nguồn VCORE cấp cho CPU, mạch có nhiệm vụ biến đổi điện áp 12V/2A thành điện áp khoảng 1,5V và cho dòng lên tới 10A để cấp cho CPU, mạch bao gồm các linh kiện như đèn Mosfet, IC dao động, các mạch lọc L,C

Ngoài ra còn một số khối hỗ trợ khác: IC điều khiển các cổng vào ra dữ liệu, modul ổn áp…

1.2. Các hệ điều hành

1.2.1. Khái niệm hệ điều hành

Hệ điều hành là tập những chương trình tổ chức thành hệ thống với nhiệm vụ đảm bảo sự tương tác giữa người dùng với máy tính, cung cấp những phương tiện và dịch vụ để điều phối việc thực hiện những chương trình, quản lý chặt chẽ những tài nguyên của máy, tổ chức khai thác chúng một cách thuận tiện và tối ưu. Hệ điều hành đóng vai trò là cầu nối giữa người sử dụng hay là chương trình ứng dụng với phần cứng máy tính.

Hệ điều hành sẽ quản lý tất cả phần mềm và phần cứng trên thiết bị. Có một số chương trình khác nhau sẽ chạy cùng một lúc trong hầu hết thời gian và tất cả chúng đều cần phải truy cập vào bộ xử lý trung tâm (CPU), bộ nhớ tạm (RAM) và ổ cứng của máy tính. Hệ điều hành sẽ phối hợp tất cả điều này lại để đảm bảo cách chương trình sẽ nhận được những gì chúng cần để có thể khởi chạy. Tóm lại, các công việc chính của một hệ điều hành bao gồm:

- Điều khiển và quản lý trực tiếp các phần thuộc về phần cứng như bo mạch chủ, card đồ họa cũng như card âm thanh...
- Thực hiện một số thao tác cơ bản trong máy tính như các thao tác đọc, ghi tập tin, quản lý hệ thống tập tin (file system) và quản lý các kho dữ liệu.
- Cung ứng một hệ thống giao diện sơ khai cho các ứng dụng, thường là thông qua một hệ thống thư viện các hàm chuẩn để điều hành các phần cứng mà từ đó các ứng dụng có thể được gọi tới.
- Cung ứng một hệ thống lệnh cơ bản để điều hành máy tính. Các lệnh này gọi là lệnh hệ thống (system command).

- Ngoài ra, trong một vài trường hợp, hệ điều hành cũng cung cấp các dịch vụ cơ bản cho các phần mềm ứng dụng thông thường như chương trình duyệt web, chương trình soạn thảo văn bản…

1.2.2. Một số hệ điều hành máy tính thông dụng

- Hệ điều hành windows

Hệ điều hành windows được ra đời vào những năm 1985 bởi Microsoft và được sử dụng phổ biến trên máy vi tính với hơn 90% người dùng. Chính vì nó quá thông dụng nên phần mềm ứng dụng đã được phát triển để chạy trên windows cũng nhiều hơn bất kỳ hệ điều hành nào khác. Một số đặc trưng chung của hệ điều hành windows là :

- Là hệ điều hành đa nhiệm, một người dùng.
- Do Bill Gates phát triển, nhà sáng lập của Microsoft. Tuy nhiên hiện nay Bill Gates đã từ bỏ windows. Có nghĩa là hệ điều hành windows 10 không phải do Bill Gate phát triển.
- Có khả năng làm việc ở trong môi trường mạng…

Ưu điểm hệ điều hành windows

- Do là hệ điều hành thông dụng nên hầu như những nhà phát triển phần mềm đều đã phát triển phần mềm và ứng dụng lên hệ điều hành này tạo ra một sự đa dạng về ứng dụng và phần mềm
- Sử dụng đơn giản với chuột và bàn phím. Không khó như trên hệ điều hành Mac OS
- Dễ dàng cài đặt từ cài đặt của hệ điều hành tới cài đặt phần mềm và ứng dụng

Nhược điểm hệ điều hành windows

Tốn chi phí mua bản quyền của hệ điều hành hằng năm với giá khá cao.

Các phần mềm và ứng dụng ở trên hệ điều hành windows thường là phải trả phí với giá cũng khá là đắt.

Không chuyên dụng để làm việc như là hệ điều hành Mac OS

- Hệ điều hành Mac OS

Mac OS được thiết kế bởi nhà sản xuất Apple dành riêng cho máy tính Mac Do. Vì chỉ được thiết kế để chạy với máy tính của Apple nên không được phổ biến như hệ điều hành windows. Vì vậy có rất ít chương trình, ứng dụng và phần mềm được viết để dành riêng cho nó.

Tuy nhiên, Mac OS được coi như là một trong những hệ điều hành sáng tạo của Apple. Nhiều người dùng đánh giá cho rằng nó là một hệ điều hành mạng, dễ sử dụng và đặc biệt là đối với thiết kế đồ họa chuyên nghiệp, làm chế bản trên máy tính, lập trình viên cũng như các fan cứng của Apple.

- Hệ điều hành Linux

Ngoài sự độc bá của hệ điều hành Windows và Mac OS ở trên thế giới, thì chúng ta vẫn còn có một hệ điều hành đang âm thầm phát triển với các đặc trưng riêng biệt. Đó chính là hệ điều hành Linix được ra đời năm 1992. Ngày nay thì Linux được phân chia thành nhiều nhánh như là Ubuntu, Linux Mint hay là Fedora… Dù ít được sử dụng nhưng mà Linux vẫn là một trong những hệ điều hành phổ biến nhất ở hiện nay.

Ưu điểm :

Nền tảng mã nguồn mở và miễn phí, không cần phải mất tiền mua mà vẫn sử dụng đầy đủ những tính năng ở trên hệ điều hành Linux

Hầu như các phần mềm độc hại không thể nào hoạt động trên hệ điều hành này

Cho phép người dùng có thể tự do chỉnh sửa, phù hợp cho những bạn thích tìm tòi khám phá cái mới

Hoạt động mượt mà ở trên máy tính có cấu hình yếu

Hệ điều hành này được những lập trình viên ưa chuộng

Nhược điểm :

Số lượng ứng dụng hỗ trợ còn rất hạn chế

Nhiều nhà sản xuất không được phát triển driver hỗ trợ nền tảng hệ điều hành Linux

Khó làm quen

- Hệ điều hành Ubuntu

Được phát triển và bảo trợ bởi công ty Canonical. Mục tiêu của hệ điều hành Ubuntu này là hướng đến các dòng máy tính xách tay, máy tính để bàn và máy chủ (server). Đối tượng sử dụng của hệ điều hành này là người dùng phổ thông và cả những chuyên gia. Điều đã biệt nhất mà bất kỳ ai cũng phải thích thú đó là hệ điều hành Ubuntu luôn luôn miễn phí.

Ưu điểm : Hoàn toàn không có virus và miễn phí.

Nhược điểm :

Phần cứng ít được hỗ trợ hơn những hệ điều hành khác.

Chính sách hỗ trợ khách hàng thiếu sự nhất quán và tốn kém.

Thao tác sử dụng cần sự thành thạo.

Android

Hệ điều hành Android được đánh giá là hệ điều hành dành cho di động phổ biến nhất hiện nay, chiếm tới hơn 50% lượng sử dụng trên toàn thế giới. Android là một hệ điều hành có

mã nguồn mở, được xây dựng dựa trên nền tảng Linux. Ban đầu, Android được xây dựng và phát triển bởi Tổng công ty Android có sự hỗ trợ từ Google. Vào năm 2005, Google đã mua lại phần mềm này và chính thức cho ra mắt hệ điều hành Android vào năm 2007.

Ưu điểm:

Thân thiện và dễ sử dụng.

Có mã nguồn mở, khả năng tuỳ biến cao.

Có mặt trên đa dạng các dòng sản phẩm của rất nhiều hãng công nghệ.

Kho ứng dụng Google Play Store vô cùng đồ sộ.

Có khả năng đa nhiệm.

Có thể chạy trên nhiều thiết bị khác nhau như mobile, watch, tivi, car hay camera…

Nhược điểm:

Dễ nhiễm phần mềm độc hại và virus.

Kho ứng dụng lớn dẫn đến khó kiểm soát chất lượng.

Cập nhật không tự động với tất cả thiết bị, có những thiết bị không thể nâng cấp lên cao hơn nên buộc phải thay thiết bị mới.

iOS

Apple đã xây dựng một hệ điều hành không chỉ tối ưu phần cứng mà còn mang tính bảo mật rất cao. Đó chính là hệ điều hành iOS. iOS là hệ điều hành dành cho điện thoại di động, máy tính bảng, iPod được phát triển độc quyền của hãng Apple. Hiện tại, hệ điều hành iOS chiếm khoảng hơn 20% thị trường hệ điều hành điện thoại di động trên thị trường và tới hơn 40% thị trường hệ điều hành máy tính bảng. Kho ứng dụng của iOS cũng rất đồ sộ, phục vụ tối đa nhu cầu làm việc và giải trí của người sử dụng.

Ưu điểm:

Nền tảng ổn định.

Ứng dụng trên iOS hoạt động rất mượt mà.

Độ tin cậy và tính bảo mật cao.

Ứng dụng phong phú, chất lượng, cập nhật nhanh.

Nhược điểm:

Trao đổi dữ liệu cần phải qua iTunes nên dễ gây bất tiện, tốn thời gian.

Khả năng tuỳ chỉnh còn hạn chế.

iOS chỉ hoạt động trên các thiết bị công nghệ độc quyền của Apple.

Còn một số hệ điều hành khác như Windows Phone,…

1.3. Hệ đếm nhị phân

Hệ nhị phân (hay hệ đếm cơ số hai hoặc mã nhị phân) là một hệ đếm dùng hai ký tự để biểu đạt một giá trị số, bằng tổng số các lũy thừa của 2. Hai ký tự đó thường là 0 và 1; chúng thường được dùng để biểu đạt hai giá trị hiệu điện thế tương ứng (hiệu điện thế cao là 1 và hiệu điện áp thấp là 0). Do có ưu điểm tính toán đơn giản, dễ dàng thực hiện về mặt vật lý, chẳng hạn như trên các mạch điện tử, hệ nhị phân trở thành một phần kiến tạo căn bản trong các máy tính đương thời.

Ví dụ:

$1_2 = 1 \times 2^0 = 1 \times 1 = 1_{10}$

$10_2 = (1 \times 2^1) + (0 \times 2^0) = 2 + 0 = 2_{10}$

$101_2 = (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0) = 4 + 0 + 1 = 5_{10}$

Phép cộng:

0 + 0 = 0

0 + 1 = 1

1 + 0 = 1

1 + 1 = 0 (nhớ 1 lên hàng thứ 2)

Phép trừ:

0 − 0 = 0

0 − 1 = 1 (mượn 1 ở bit tiếp theo)

1 − 0 = 1

1 − 1 = 0

Phép nhân:

Phép tính nhân trong hệ nhị phân cũng tương tự như phương pháp làm trong hệ thập phân.

Phép chia:

Chia nhị phân cũng tương tự như phép chia trong hệ thập phân.

Quy đổi hệ số 10 và hệ nhị phân:

118 tương đương:

$59 \times 2 + 0$

$(29 \times 2 + 1) \times 2 + 0$

$((14 \times 2 + 1) \times 2 + 1) \times 2 + 0$

$(((7 \times 2 + 0) \times 2 + 1) \times 2 + 1) \times 2 + 0$

$((((3 \times 2 + 1) \times 2 + 0) \times 2 + 1) \times 2 + 1) \times 2 + 0$

$(((((1 \times 2 + 1) \times 2 + 1) \times 2 + 0) \times 2 + 1) \times 2 + 1) \times 2 + 0$

$1 \times 2^6 + 1 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$

1110110_2

Phép tính	Số dư
118 ÷ 2 = 59	0
59 ÷ 2 = 29	1
29 ÷ 2 = 14	1
14 ÷ 2 = 7	0
7 ÷ 2 = 3	1
3 ÷ 2 = 1	1
1 ÷ 2 = 0	1

1.4. Các ngôn ngữ lập trình

Các ngôn ngữ lập trình phổ biến nhất hiện nay:

1. Java

Java là một trong các ngôn ngữ lập trình phổ biến nhất, là ngôn ngữ được sử dụng bởi hàng trăm triệu lập trình viên và được phát triển trên hàng tỷ thiết bị trên toàn cầu. Java là ngôn ngữ lập trình hướng đối tượng và nó được thiết kế để chạy đa nền tảng, đa hệ điều hành.

Java có thể xây dựng các ứng dụng Desktop, các trò chơi. Thêm nữa, Java còn được sử dụng rộng rãi trong lập trình phía server, thường được sử dụng bởi các doanh nghiệp, xây dựng các hệ thông back-end (tầng truy cập dữ liệu của một phần mềm, hoặc cơ sở hạ tầng vật lý hoặc phần cứng).

2. Javascript

Javascript đang ngày càng phổ biến và có thứ hạng cao trong các ngôn ngữ lập trình phổ biến. Đây là một ngôn ngữ kịch bản mạnh mẽ và linh hoạt. Phần lớn cú pháp giống như ngôn ngữ lập trình C. Hơn nữa, nó có khả năng tương thích trên mọi trình duyệt web và có hơn 90% các website hiện nay đang sử dụng ngôn ngữ kịch bản này. Với sự ra đời của Node.js, nó đang là một công nghệ lập trình phía server và cho phép tương tác thời gian thực.

3. Python

Python đang là một trong các ngôn ngữ lập trình bậc cao phổ biến được sử dụng rộng rãi cho mọi chương trình máy tính. Đây là một ngôn ngữ đơn giản cả về cú pháp lẫn cách sử dụng và là ngôn ngữ dễ tiếp cận nhất cho người mới học lập trình. Python cũng được sử dụng nhiều trong các bài toán về trí tuệ nhân tạo và học máy.

4. C

C được phát triển bởi Dennis Ritchie vào năm 1972 để sử dụng trong hệ điều hành UNIX. Nó là tiền thân của ngôn ngữ C++. Đây là một ngôn ngữ hướng chức năng và thủ tục (hướng đối tượng được bổ sung thêm ở C++). Ban đầu, nó được sử dụng nhiều trong lập trình hệ thống, nhưng do tính hiệu quả và mạnh mẽ nên nó đã được sử dụng trong nhiều ứng dụng khác nữa.

5. C++

Đây là ngôn ngữ kế thừa của ngôn ngữ lập trình C. C++ có thêm lập trình hướng đối tượng, bổ sung thêm các thư viện, hàm và một số tính năng mới chưa có ở C.

C++ là một ngôn ngữ có hiệu năng cao nên được sử dụng xây dựng các ứng dụng desktop, các phần mềm hệ thống và các chương trình game.

6. C# (C Sharp)

C# là một ngôn ngữ đơn giản, hiện đại, hướng đối tượng được phát triển bởi Microsoft nằm trong bộ framework .NET. Nó được thiết kế là một ngôn ngữ nền tảng chung, bao gồm chương trình thực thi và môi trường thực thi cho phép sử dụng các ngôn ngữ lập trình bậc cao khác để lập trình cho các nền tảng và kiến trúc khác nhau.

7. PHP

Đây là một ngôn ngữ lập kịch bản dùng ở phía server (back-end) và sử dụng chủ yêu trong lập trình web. Hiện nay, hơn 80% website được xây dựng bằng PHP bao gồm Wikipedia, WordPress, Facebook, Tumblr,… Bên cạnh tính phổ biến, nó cũng dễ để sử dụng và cung cấp nhiều tính năng nâng cao cho các lập trình viên có kinh nghiệm.

8. Ruby

Ruby là một ngôn ngữ linh động, hướng đối tượng và là một ngôn ngữ kịch bản đa chức năng. Được sử dụng phổ biến bởi nó là một web framework – Rais.

9. Swift

Swift được kế thừa từ Objective-C. Nó khá dễ hiểu, nhanh và có thể giảm độ dài của dòng lệnh, tiết kiệm thời gian và năng lượng. Swift được phát triển bởi Apple.

Swift dùng để xây dựng các ứng dụng cho hệ điều hành Mac, IOS, tvOS và watchOS. Thêm nữa, sau khi trở thành mã nguồn mở, nó cũng có thể sử dụng trên hệ điều hành Linux hoặc windows để chạy các ứng dụng tương thích với các thiết bị của Apple.

10. R

Là một ngôn ngữ lập trình mã nguồn mở. R được sử dụng chủ yếu trong môi trường phần mềm bao gồm cả tính toán thống kê và đồ họa. Bạn có thể dùng R để tính toán số học qua sử dụng các gói (package) bổ sung.

R được sử dụng trong khai phá dữ liệu và khoa học thống kê, xây dựng các phần mềm thống kê cho phân tích dữ liệu.

11. SQL (Structured Query Language)

SQL hầu như được hiểu bởi các quản trị viên cơ sở dữ liệu. Trong một thời gian, SQL dường như đã mất đi sự liên quan với sự phát triển của các dịch vụ NoQuery, như MongoDB và Redis và các nền tảng điện toán Big Data không sử dụng SQL, ví dụ như Hadoop, Spark và Cassandra.

Với sự gia tăng của big data và các khó khăn trong việc quản lý nó, SQL đang nóng hơn bao giờ hết.

Nói chung, SQL có liên quan một lần nữa bởi vì nó cần để quản lý và phân tích (nhưng không lưu trữ) big data. Cộng đồng các nhà phát triển thậm chí còn dự đoán một số loại hợp nhất SQL và NoQuery. SQL có ở khắp mọi nơi, trong tương lai SQL sẽ tiếp tục có liên quan đến các ứng dụng.

12. Visual Basic

Đó là một hệ thống lập trình máy tính do Microsoft phát triển và sở hữu . Visual Basic ban đầu được tạo ra để giúp việc viết chương trình cho hệ điều hành máy tính Windows trở nên dễ dàng hơn. Cơ sở của Visual Basic là một ngôn ngữ lập trình trước đó có tên là BASIC được phát minh bởi các giáo sư John Kemeny và Thomas Kurtz của Đại học Dartmouth.

Ngoài ra còn các ngôn ngữ lập trình khác, như Fortran,…

1.5. Câu hỏi ôn tập chương 1

Câu 1.1. Vai trò của các bộ phận trong máy tính?

Câu 1.2. Vai trò của hệ điều hành?

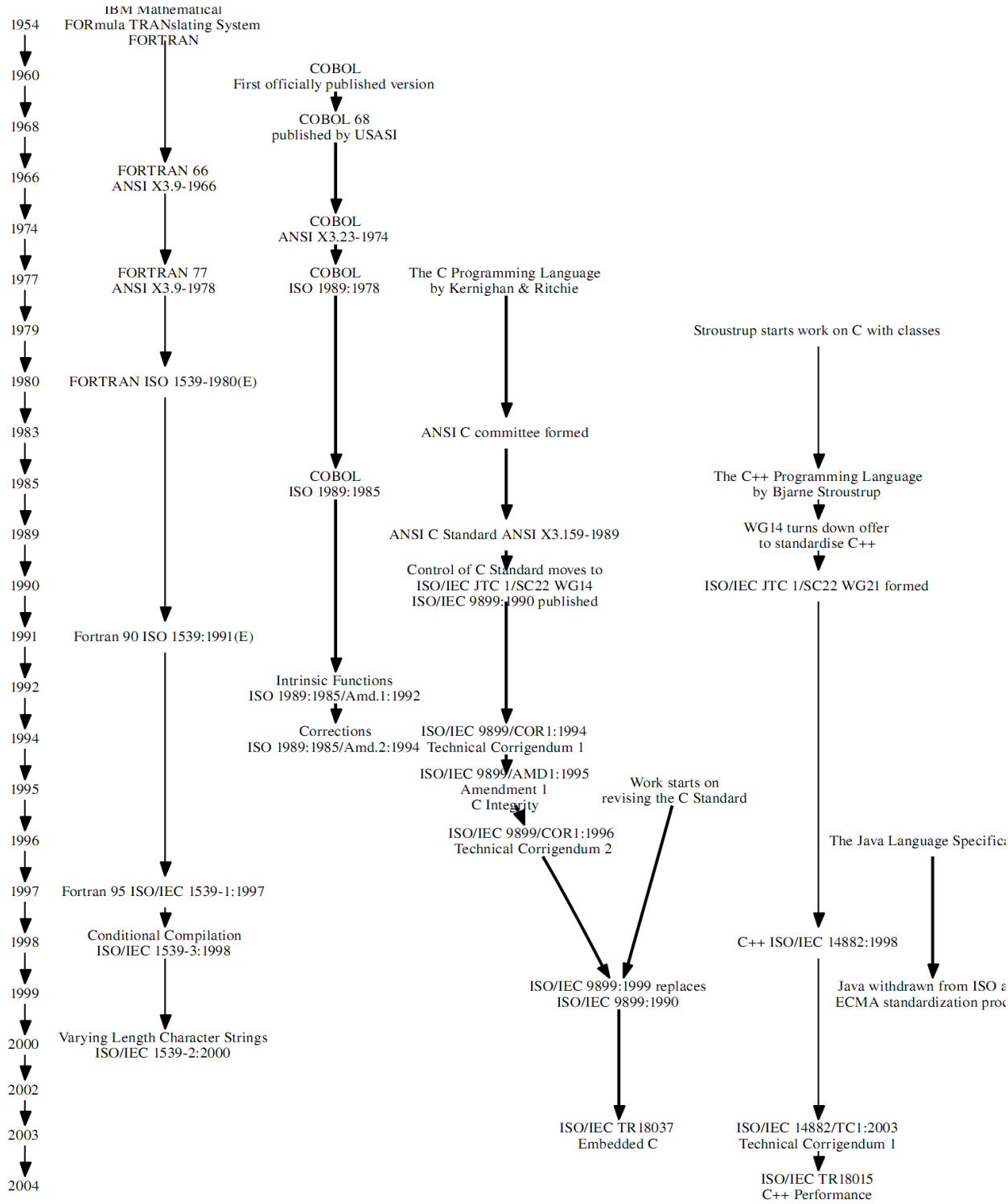
Câu 1.3. Với các số tronh hệ nhi phân: a = 10101011, b = 101, hãy tính các phép tính:
c = a + b; d = a – b; e = a*b; f = a/b.

Câu 1.4. Phân tích khả năng sử dụng C++ trong lập trình về các vấn đề kỹ thuật hiện nay?

Chương 2. CƠ BẢN VỀ LẬP TRÌNH C++

2.1. Khái Quát Về C, C++, Visual C++

2.1.1. Sự hình thành và phát triển của C/C++



Lịch sử phát triển C/C++ và một số ngôn ngữ lập trình khác [4].

2.1.1.1. Nguồn gốc của ngôn ngữ C

Ngôn ngữ lập trình C được phát minh và cài đặt lần đầu tiên bởi Dennis Ritchie và được sử dụng trong hệ điều hành Linux. C là kết quả của một quá trình phát triển bắt đầu bằng ngôn ngữ B (được phát minh bởi Ken Thompson). Từ ngôn ngữ B đã phát triển ngôn ngữ C vào năm 1970.

2.1.1.2. Nguồn gốc của C++

C++ bắt đầu như là mở rộng phiên bản của C. Mở rộng của C++ được đề cập lần đầu tiên bởi Bjarne Stroustrup vào năm 1979 tại phòng thí nghiệm Bell trong Murray Hill, New Jersey. Đầu tiên ông gọi ngôn ngữ này là C với lớp (C with class). Đến năm 1983 đổi thành tên là C++.

Danh sách các ngôn ngữ lập trình được yêu thích nhất năm 2017:

[Bình chọn trên IEEE Spectrum theo techtalk.vn]

2.2. Các trình biên dịch C/C++

2.2.2. Môi trường làm việc của C++ trên Turbo C hoặc Borland C

2.2.2.1. Khởi động - Thoát khỏi C++

Khởi động C++ cũng như mọi chương trình khác bằng cách nhấp đúp chuột lên biểu tượng của chương trình. Khi chương trình được khởi động sẽ hiện ra giao diện gồm có menu công việc và một khung cửa sổ bên dưới phục vụ cho soạn thảo. Một con trỏ nhấp nháy trong khung cửa sổ và chúng ta bắt đầu nhập nội dung chương trình. Đây là giao diện của các trình biên dịch quen thuộc là Turbo C hoặc Borland C. Ngoài ra còn có các trình biên dịch khác chúng ta tự tham khảo trong các tài liệu liên quan.

Để kết thúc làm việc với C++ quay về môi trường Windows chúng ta ấn **Alt-X.**

2.2.2.2. Giao diện và cửa sổ soạn thảo

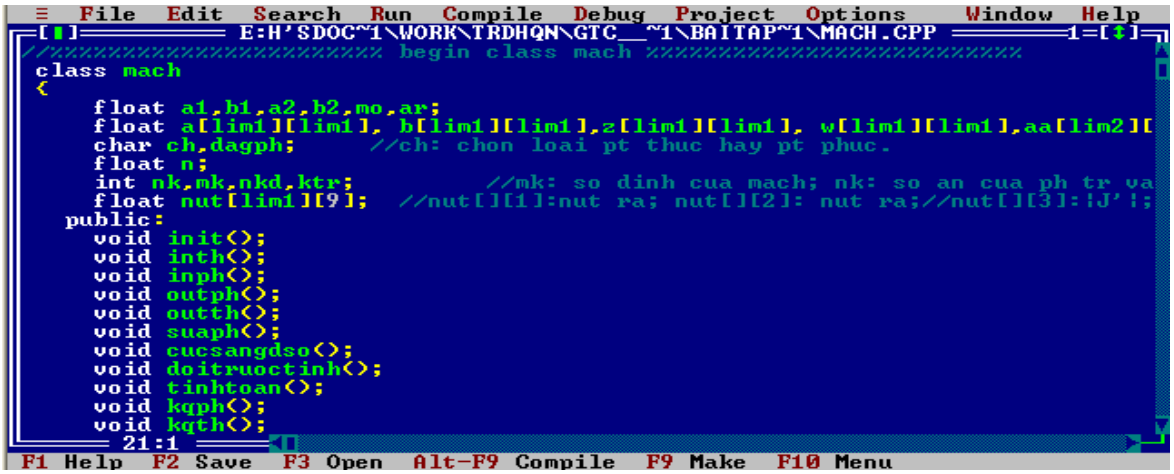
a. Mô tả chung

Khi gọi chạy C++ trên màn hình sẽ xuất hiện một menu xổ xuống và một cửa sổ soạn thảo. Trên menu gồm có các nhóm chức năng: **File, Edit, Search, Run, Compile, Debug, Project, Options, Window, Help**. Để kích hoạt các nhóm chức năng, có thể ấn **Alt+chữ cái** biểu thị cho menu của chức năng đó (là chữ cái có gạch dưới). Các bộ chương trình dịch hỗ trợ người lập trình một môi trường tích hợp, nghĩa là ngoài chức năng soạn thảo, nó còn cung cấp nhiều chức năng, tiện ích khác giúp người lập trình vừa có thể soạn thảo văn bản vừa gọi chạy chương trình vừa gỡ lỗi …

b. Các chức năng soạn thảo

Bộ soạn thảo của Turbo C hoặc Borland C cũng giống hầu hết các bộ soạn thảo văn bản Work, Notepad…, tuy nhiên có các phím chức năng sau là khác:

Ctrl + Insert -> copy
Shift + Delete -> Cut (cắt)
Shift + Insert -> Paste (dán)
Ctrl + Delete -> Delete (xóa)



Giao diện soạn thảo của Turbo C hoặc Borland C.

c. Chức năng dịch và chạy chương trình

+ Ctrl-F9: Khởi động chức năng dịch và chạy toàn bộ chương trình.
+ F4: Chạy chương trình từ đầu đến dòng lệnh hiện tại (đang chứa con trỏ).
+ F7: Chạy từng lệnh một của hàm main(), kể cả các lệnh trong hàm con.
+ F8: Chạy từng lệnh một của hàm main(). Khi đó mỗi lời gọi hàm được xem là một lệnh (không chạy từng lệnh trong các hàm được gọi).

Các chức năng liên quan đến dịch chương trình có thể được chọn thông qua menu Compile (Alt-C).

d. Tóm tắt một số phím nóng hay dùng

+ Các phím kích hoạt menu: Alt+chữ cái đại diện cho nhóm menu đó. Ví dụ Alt-F mở menu File để chọn các chức năng cụ thể trong nó như Open (mở file), Save (ghi file), Print (in nội dung văn bản ra máy in), … Alt-C mở menu Compile để chọn các chức năng dịch chương trình.

+ F1: mở cửa sổ trợ giúp. Đây là chức năng quan trọng giúp người lập trình nhớ tên

lệnh, cú pháp và cách sử dụng.

+ F2: ghi tệp lên đĩa.

+ F3: mở tệp tin ra sửa chữa.

+ F4: chạy chương trình đến vị trí con trỏ .

+ F5: Thu hẹp/mở rộng cửa sổ soạn thảo.

+ F6: Chuyển đổi giữa các cửa sổ soạn thảo.

+ F7: Chạy chương trình theo từng lệnh, kể cả các lệnh trong hàm con.

+ F8: Chạy chương trình theo từng lệnh trong hàm chính.

+ F9: Dịch và liên kết chương trình. Thường dùng chức năng này để tìm lỗi cú pháp của chương trình nguồn trước khi chạy.

+ Alt-F7: Chuyển con trỏ về nơi gây lỗi trước đó.

+ Alt-F8: Chuyển con trỏ đến lỗi tiếp theo.

+ Ctrl-F9: Chạy chương trình.

+ Ctrl-Insert: Lưu khối văn bản đuợc đánh dấu vào bộ nhớ đệm.

+ Shift-Insert: Dán khối văn bản trong bộ nhớ đệm vào văn bản tại vị trí con trỏ.

+ Shift-Delete: Cắt khối văn bản được đánh dấu, lưu nó vào bộ nhớ đệm.

+ Ctrl-Delete: Xoá khối văn bản được đánh dấu (không lưu vào bộ nhớ đệm).

+ Alt-F5: Chuyển sang cửa sổ xem kết quả của chương trình vừa chạy xong.

+ Alt-X: thoát C++ về lại Windows.

2.2.2.3. Các bước để tạo và thực hiện một chương trình C++

a. Quy trình viết và thực hiện chương trình

Trước khi viết và chạy một chương trình thông thường chúng ta cần:

1. Xác định yêu cầu của chương trình. Nghĩa là xác định dữ liệu đầu vào (input) cung cấp cho chương trình và tập các dữ liệu cần đạt được tức đầu ra (output). Các tập hợp dữ liệu này ngoài các tên gọi còn cần xác định kiểu của nó.Ví dụ để giải một phương trình bậc 2 dạng: $ax^2 + bx + c = 0$, cần báo cho chương trình biết dữ liệu đầu vào là a, b, c và đầu ra là nghiệm x_1 và x_2 của phương trình. Kiểu của a, b, c, x1, x2 là các số thực.

2. Xác định thuật toán giải.

3. Cụ thể hoá các khai báo kiểu và thuật toán thành dãy các lệnh, tức viết thành chương trình thông thường là trên giấy, sau đó bắt đầu soạn thảo vào trong máy. Quá trình này được gọi là soạn thảo chương trình nguồn.

4. Dịch chương trình nguồn để tìm và sửa các lỗi gọi là lỗi cú pháp.

5. Chạy chương trình, kiểm tra kết quả in ra trên màn hình. Nếu sai, sửa lại chương trình, dịch và chạy lại để kiểm tra. Quá trình này được thực hiện lặp đi lặp lại cho đến khi chương trình chạy tốt theo yêu cầu đề ra của NSD.

b. Soạn thảo tệp chương trình nguồn

Soạn thảo chương trình nguồn là một công việc đơn giản: gõ nội dung của chương trình (đã viết ra giấy) vào trong máy và lưu lại nó lên đĩa. Có thể soạn chương trình nguồn trên các bộ soạn thảo (editor) khác nhưng phải chạy trong môi trường tích hợp C++ (Borland C, Turbo C). Mục đích của soạn thảo là tạo ra một văn bản chương trình và đưa vào bộ nhớ của

máy. Văn bản chương trình cần được trình bày sáng sủa, rõ ràng. Các câu lệnh cần gióng thẳng cột theo cấu trúc của lệnh (các lệnh chứa trong một lệnh cấu trúc được trình bày thụt vào trong so với điểm bắt đầu của lệnh). Các chú thích nên ghi ngắn gọn, rõ nghĩa và phù hợp.

c. Dịch chương trình

Sau khi đã soạn thảo xong chương trình nguồn, bước tiếp theo thường là dịch (ấn tổ hợp phím Alt-F9) để tìm và sửa các lỗi gọi là lỗi cú pháp. Trong khi dịch C++ sẽ đặt con trỏ vào nơi gây lỗi (viết sai cú pháp) trong văn bản. Sau khi sửa xong một lỗi NSD có thể dùng Alt-F8 để chuyển con trỏ đến lỗi tiếp theo hoặc dịch lại. Để chuyển con trỏ về ngược lại lỗi trước đó có thể dùng Alt-F7. Quá trình sửa lỗi − dịch được lặp lại cho đến khi văn bản đã được sửa hết lỗi cú pháp.

Sản phẩm sau khi dịch là một tệp mới gọi là chương trình đích có đuôi EXE tức là tệp mã máy để thực hiện.

Trong và sau khi dịch, C++ sẽ hiện một cửa sổ chứa thông báo về các lỗi (nếu có), hoặc thông báo chương trình đã được dịch thành công (không còn lỗi). Các lỗi này được gọi là lỗi cú pháp.

Để dịch chương trình ta chọn menu \Compile\Compile hoặc \Compile\Make hoặc nhanh chóng hơn bằng cách ấn tổ hợp phím Alt-F9.

d. Chạy chương trình

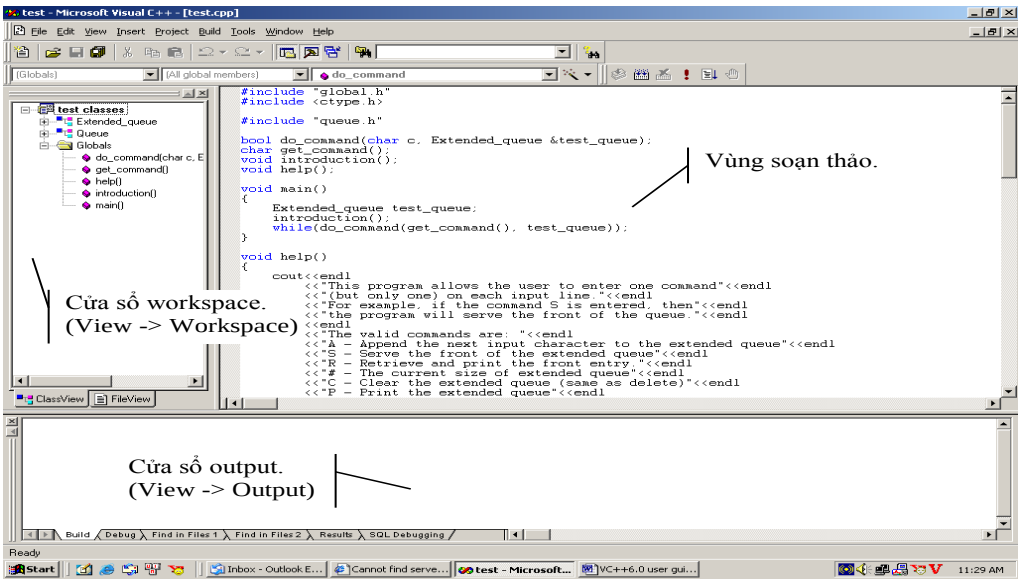
Ấn Ctrl-F9 để chạy chương trình, nếu chương trình chưa dịch sang mã máy, máy sẽ tự động dịch lại trước khi chạy. Kết quả của chương trình sẽ hiện ra trong một cửa sổ kết quả để NSD kiểm tra. Nếu kết quả chưa được như mong muốn, quay lại văn bản để sửa và lại chạy lại chương trình. Quá trình này được lặp lại cho đến khi chương trình chạy đúng như yêu cầu đã đề ra. Khi chương trình chạy, cửa sổ kết quả sẽ hiện ra tạm thời che khuất cửa sổ soạn thảo. Sau khi kết thúc chạy chương trình cửa sổ soạn thảo sẽ tự động hiện ra trở lại và che khuất cửa sổ kết quả. Để xem lại kết quả đã hiện ấn Alt-F5. Sau khi xem xong để quay lại cửa sổ soạn thảo ấn phím bất kỳ.

2.2.3. Môi trường làm việc của C++ trên Visual C++

2.2.3.1. Giới thiệu:

Visual C++ là phần mềm lập trình cho ngôn ngữ C++ với nhiều tính năng và hỗ trợ tốt. Trong phần thực hành này, chúng ta chỉ dùng nó để lập trình các ứng dụng không phải trên nền Windows (chỉ dùng Win32 Console Application). Các tính năng và những hỗ trợ khác sinh viên tự nghiên cứu và ứng dụng trong công việc sau này.

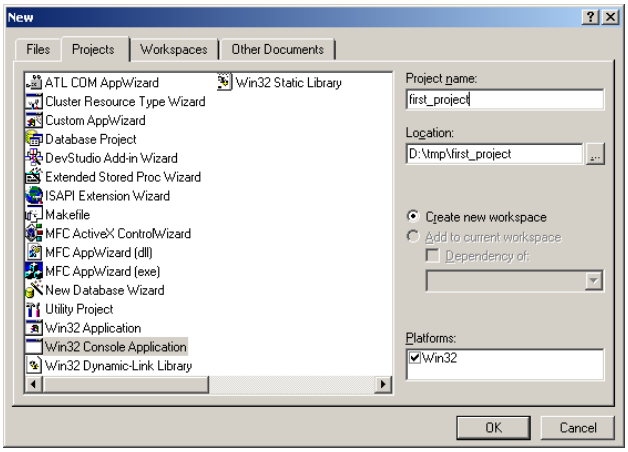
Một project trong VC++ chỉ dùng để lập trình cho một ứng dụng duy nhất.



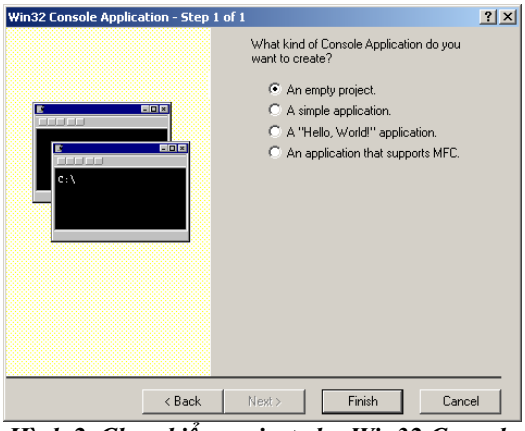
Giao diện Visual C++ 6.0.

3.2. Các bước tạo ra một project đơn giản

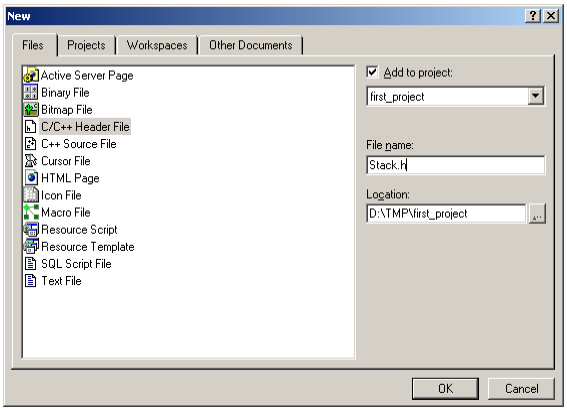
- *Bước 1:* Tạo một project đơn giản
  - Chọn File -> New, chọn tab Project (hình 1)
  - Chọn *Win32 Console Application*
  - Gõ tên của project muốn tạo vào mục *Project name*
  - Chọn thư mục cần lưu trữ trong mục *Location*. Chú ý, sau khi chọn thư mục xong thì phần mềm tự động thêm tên của project vào để thành tên đầy đủ của thư mục chứa project này. Toàn bộ các tập tin của project này đều nằm trong thư mục đã chỉ ra.
  - Nhấn nút OK
  - Một hội thoại khác xuất hiện (Win32 Console Application – Step 1 of 1) để giúp ta tạo ra một project bước ban đầu. Để cho đơn giản ta chọn *An empty project* (hình 2).
  - Nhấn nút *Finish* để tạo ra một project rỗng.



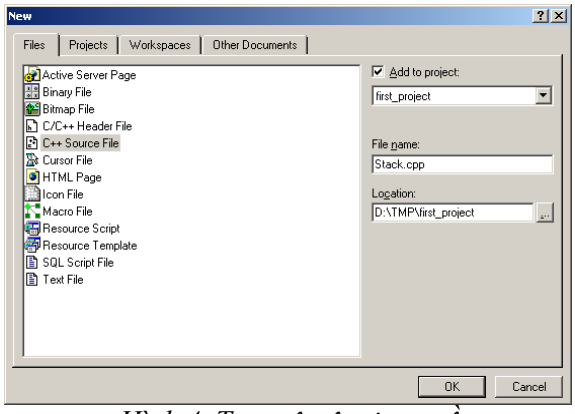
Hình 1. Tạo một project mới



Hình 2. Chọn kiểu project cho Win32 Console Application



Hình 3. Tạo một tập tin header



Hình 4. Tạo một tập tin nguồn

- *Bước 2:* Tạo một tập tin header *MyFirst.h*
  - Chọn File -> New, chọn tab Files (hình 3)
  - Chọn C/C++ Header Files
  - Gõ tên tập tin với phần mở rộng là **.h** vào mục *File name*. Chú ý rằng dấu đánh dấu ở mục *Add to project* cho biết là tập tin mới tạo này sẽ được thêm vào trong project có tên được hiển thị lên đó. Nếu tên project mà khác thì phải chọn lại trong danh sách.
  - Chú ý đến mục *Location* để đảm bảo là tập tin mới tạo ra cũng sẽ ở trong cùng thư mục với project vừa tạo ra.
  - Nhấn nút OK và bắt đầu soạn thảo
- *Bước 3:* Tạo một tập tin nguồn *MyFirst.cpp*
  - Chọn File -> New, chọn tab Files (hình 4)
  - Chọn C/C++ Source Files
  - Gõ tên tập tin với phần mở rộng là **.cpp** vào mục *File name*.
  - Nhấn OK và bắt đầu soạn thảo.
- *Bước 4:* Tạo tập tin nguồn dùng thực thi *MyMain.cpp*
  - Tạo một tập tin nguồn
  - Soạn thảo hàm chính của chương trình là **main**
- *Bước 5:* Thực thi chương trình
  - Chọn Build -> Build *first_project.exe*
  - Chọn Build -> Execute *first_project.exe*
  - Chú ý là tên của project đang tạo ra sẽ được thay thế vào các chữ in nghiêng ở trên

3.3. Ví dụ về một project đơn giản:

- Tạo ra một project đơn giản tên là V*idu*
- Tạo một tập tin header tên là *MyUtility.h* với nội dung như sau:

```
//file MyUtility.h
#ifndef _MYUTILITY_H_      //dde^? tra'nh bi. lo^~i redefination
#define _MYUTILITY_H_
#include <iostream.h>      //du`ng cho ca'c le^.nh cout, cin
#include <ctype.h>         //du`ng cho ca'c ha`m tolower
enum Error_code {success, overflow, underflow};
typedef int my_type;

#endif
```

- Tạo một tập tin header tên là *MyClass.h* với nội dung như sau:

```
//file MyClass.h
```

```
#ifndef _MYCLASS_H_        //dde^? tra'nh bị lo^~i redefination
#define _MYCLASS_H_
class MyClass {
public:
    MyClass();
    my_type input(const int &);
    my_type output(int &);
private:
    int data;
};
#endif
```

- Tạo một tập tin source tên là *MyClass.cpp* với nội dung như sau:

```
//file MyClass.cpp
#include "MyUtility.h"     //dde^? du`ng ca'c khai ba'o chung
#include "MyClass.h"               //dde^? du`ng ca'c khai ba'o cu?a MyClass

MyClass::MyClass()
{
    cout << "Khoi tao mot object cho MyClass" << endl << endl << flush;
    data = 0;
}

my_type MyClass::input(const int &x)
{
    data = x;
    return 1;
}

my_type MyClass::output(int &x)
{
    x = data;
    return 1;
}
```

- Tạo một tập tin source tên là *Vidu.cpp* với nội dung như sau:

```
//file Vidu.cpp
#include "MyUtility.h"     //dde^? du`ng ca'c khai ba'o chung
#include "MyClass.h"               //dde^? du`ng ca'c khai ba'o cu?a MyClass

int main() {
    MyClass obj;
    int x;
    cout << "Nhap vao mot so: " << flush;
    cin >> x;
    obj.input(x);
    cout << "Nhap vao mot so khac: " << flush;
    cin >> x;
    cout << "So moi nhap vao la " << x << endl << flush;
    obj.output(x);
    cout << "So da nhap vao truoc do la " << x << endl << flush;
    return 0;
}
```

- Thực thi chương trình sẽ được kết quả như sau:

3.4. Cách sửa lỗi trong chương trình:

Khi dịch chương trình mà gặp lỗi thì dùng thanh cuộn trong cửa sổ *Output* để tìm **đến lỗi đầu tiên**. Cố gắng tìm hiểu xem lỗi đó là gì và sửa lỗi. Thông thường khi double-click vào lỗi này thì phần mềm sẽ nhảy con trỏ đến vị trí xảy ra lỗi để giúp ta hiệu chỉnh nhanh chóng. **Chỉ sửa lỗi đầu tiên rồi dịch lại.**

Nếu phần mềm mở một tập tin có sẵn của VC++ (ví dụ như iostream.h) thì đó không phải là nơi gây ra lỗi mà chính là do chúng ta đã dùng sai lệnh nào đó hoặc là lỗi sai đó xảy ra ở những tập tin khác.

3.5. Dùng lại các chương trình đã viết sẵn:

Sau khi tạo ra project, chép các tập tin này vào thư mục của project đó. Chọn Project -> Add To Project -> Files. Chọn các tập tin cần thêm vào trong thư mục của project đó. Nhấn OK.

Trong cửa sổ Workspace, chọn tab FileView sẽ giúp ta nhìn thấy tất cả các tập tin của project này. Chọn ClassView sẽ cho ta nhìn thấy cấu trúc các class mà ta đang có.

2.3. Cấu Trúc Của Một Chương Trình C/C++

2.3.1. Cấu trúc đơn giản của một chương trình:

Một chương trình C++ có thể được đặt trong một hoặc nhiều file văn bản khác nhau. Mỗi file văn bản chứa một số phần nào đó của chương trình. Với những chương trình đơn giản và ngắn thường chỉ cần đặt chúng trên một file.

Một chương trình gồm nhiều hàm, mỗi hàm phụ trách một công việc khác nhau của chương trình. Đặc biệt trong các hàm này có một hàm duy nhất có tên hàm là main(). Khi chạy chương trình, các câu lệnh trong hàm main() sẽ được thực hiện đầu tiên. Trong hàm main() có thể có các câu lệnh gọi đến các hàm khác khi cần thiết, và các hàm này khi chạy lại có thể gọi đến các hàm khác nữa đã được viết trong chương trình (trừ việc gọi quay lại hàm main()). Sau khi chạy đến lệnh cuối cùng của hàm main() chương trình sẽ kết thúc.

Cụ thể, thông thường một chương trình gồm có các nội dung sau:

+ Phần khai báo các tệp nguyên mẫu: khai báo tên các tệp chứa những thành phần có sẵn (như các hằng chuẩn, kiểu chuẩn và các hàm chuẩn) mà người sử dụng (NSD) sẽ dùng trong chương trình.

+ Phần khai báo các kiểu dữ liệu, các biến, hằng ... do NSD định nghĩa và được dùng chung trong toàn bộ chương trình.

+ Danh sách các hàm của chương trình (do NSD viết, bao gồm cả hàm main()).

Một chương trình đơn giản có cấu trúc như ví dụ sau:

Ví dụ 1-2-3:

```
// my first program in C++
#include <iostream.h>
int main ()
{
  cout << "Hello World!";
  return 0;
}
```

```
/* my first program in C++
*/
#include <iostream.h>
void main ()
{
  cout << "Hello World!";
  return ;
}
```

```
Hello World!
```

Đây là một trong những chương trình đơn giản nhất có thể viết bằng C++ nhưng nó đã bao gồm những phần cơ bản mà mọi chương trình C++ có. Ta xem xét từng dòng một :

// my first program in C++ hoặc **/* my first program in C++ */**

Đây là dòng chú thích. Tất cả các dòng bắt đầu bằng hai dấu (//) hoặc đoạn nằm giữa (/*) và (*/) được coi là chú thích mà chúng không có bất kì một ảnh hưởng nào đến hoạt động của chương trình. Chúng có thể được các lập trình viên dùng để giải thích hay bình phẩm

bên trong mã nguồn của chương trình. Trong trường hợp này, dòng chú thích là một giải thích ngắn gọn những gì mà chương trình chúng ta làm.

#include <iostream.h>

Các câu bắt đầu bằng dấu (#) được dùng cho preprocessor (bộ xử lý trước). Ở đây câu lệnh #include <iostream.h> báo cho trình dịch biết cần phải sử dụng thư viện iostream. Đây là một thư viện vào ra cơ bản trong C++ và nó phải được khai báo vì nó sẽ được dùng trong chương trình. Đây là cách cổ điển để sử dụng thư viện iostream.

int main() hoặc **void main()**

Dòng này tương ứng với phần bắt đầu khai báo hàm main. Hàm main là điểm mà tất cả các chương trình C++ bắt đầu thực hiện. Nó không phụ thuộc vào vị trí của hàm này (ở đầu, cuối hay ở giữa của mã nguồn) mà nội dung của nó luôn được thực hiện đầu tiên khi chương trình bắt đầu. Thêm vào đó, do nguyên nhân nói trên, mọi chương trình C++ đều phải tồn tại một hàm main.

Theo sau main là một cặp ngoặc đơn bởi vì nó là một hàm. Trong C++, tất cả các hàm phải có một cặp ngoặc đơn (), có nghĩa là nó có thể có hoặc không có tham số (không bắt buộc). Nội dung của hàm main tiếp ngay sau phần khai báo chính thức được bao trong các ngoặc nhọn ({ }) như trong ví dụ của chúng ta.

cout << "Hello World";

Dòng lệnh này làm việc quan trọng nhất của chương trình. cout là một dòng xuất dữ liệu dạng ký tự, số chuẩn trong C++ được định nghĩa trong thư viện iostream và những gì mà dòng lệnh này làm là gửi chuỗi kí tự "Hello World" ra màn hình.

Chú ý rằng dòng này kết thúc bằng dấu chấm phẩy (;). Kí tự này được dùng để kết thúc một lệnh và bắt buộc phải có sau mỗi lệnh trong chương trình C++ (một trong những lỗi phổ biến nhất của những lập trình viên C++ là quên dấu chấm phẩy).

return 0;

Lệnh return kết thúc hàm main và trả về mã đi sau nó, trong trường hợp này là 0.

Trong C++, các dòng lệnh được phân cách bằng dấu chấm phẩy (;).Chương trình được cấu trúc thành những dòng khác nhau để nó trở nên dễ đọc hơn nhưng hoàn toàn không phải bắt buộc phải làm vậy. Ví dụ, thay vì viết

```
int main ()
{
  cout << " Hello World ";
  return 0;
}
```

ta có thể viết

```
int main () { cout << " Hello World "; return 0; }
```

cũng cho một kết quả như nhau.

2.3.2. Cấu trúc chương trình theo kiểu lập trình hướng đối tượng:

```
#include <thưviện>
//----------khai báo hàm con----------
kiểudữliệu tênhàm([danhsáchđốisố]); //dấu [] có nghĩa là có thể không.
//----------khai báo lớp 1----------
class tênlớp1
  {
    kiểudữliệu biến1,biến2, biếnN;
   public:
    kiểudữliệu tênhàm([danhsáchđốisố]);
```

```
  };
//----định nghĩa cac hàm trong lớp1-----
kiểudữliệu tênlớp1:: tênhàm([danhsáchđốisố])
  {
    nộidunghàm;
  }
//##########################################
//-------------khai báo lớp 2-------------
class tênlớp2
  {
    kiểudữliệu biến1,biến2, biếnN;
   public:
    kiểudữliệu tênhàm([danhsáchđốisố]);
  };
//----định nghĩa cac hàm trong lớp2-----
kiểudữliệu tênlớp2:: tênhàm([danhsáchđốisố])
  {
    nộidunghàm;
  }
//##########################################
//---------định nghĩa hàm chính---------
kiểudữliệu main ([danhsáchđốisố])
{
  nộidunghàmchính;
 }
```

Luôn phải có hàm main.

2.3.3. Cấu trúc chương trình tổng quát:

```
#include <thưviện>
#define tênhằng giátrịhằng
#define tênmacro biểuthứcmacro
//----------khai báo hàm con----------
kiểudữliệu tênhàm([danhsáchđốisố]); //dấu [] có nghĩa là có thể không.
//-----định nghĩa kiểu dữ liệu mới----
typedef môtảkiểu tênkiểumới;
//----khai báo hằng và biến toàn cục--
cosnt kiểudữliệu tênhằng = giátrịhằng;
kiểudữliệu tênbiến;
//-------------khai báo lớp-------------
class tênlớp
  {
    kiểudữliệu biến1,biến2, biếnN;
   public:
    kiểudữliệu tênhàm([danhsáchđốisố]);
  };
//----định nghĩa cac hàm trong lớp-----
kiểudữliệu tênlớp:: tênhàm([danhsáchđốisố])
  {
    nộidunghàm;
  }
//---------định nghĩa hàm chính---------
```

```
kiểudữliệu main ([danhsáchđốisố])
{
  nộidunghàmchính;                    Luôn phải có hàm main.
 }
//------định nghĩa cac hàm con-------
kiểudữliệu tênhàm([danhsáchđốisố])
  {
    nộidunghàmcon;
  }
```

Cũng có thể định nghĩa hàm con luôn tại vị trí khai báo hàm con mà không cần khai báo.

Ví dụ 1-2-4:

```
//(Bai 01.02) Tinh gia tri Rtd = R1//R2.
#include <iostream.h>
//====================
void gioithieu();
//====================
clas giaimach
    {       float r1,r2,rtd;
     public:
            void nhapR1R2();
            void tinhRtd();
            void xuatRtd();
            void maingm();
    };
//====================
void giaimach::nhapR1R2()
    {       cout<<"\tNhap     R1 = ";
            cin>>r1;
            cout<<"\tNhap     R2 = ";
            cin>>r2;
    }
//---------------------
void giaimach::tinhRtd()
    {       rtd=r1*r2/(r1+r2);
    }
//---------------------
void giaimach::xuatRtd()
    {       cout<<"\tTinh duoc Rtd = "<<rtd;
    }
//---------------------
void giaimach:: maingm()
    {       nhapR1R2();
            tinhRtd();
            xuatRtd();
    }
//====================
```

```
int main()
{
  giaimach gm;
  gioithieu();
  gm.maingm();
  getch();
  return 0;
}
//=====================
void gioithieu()
    {       clrscr();
            count<<"\n\n\tChuong trinh tinh gia tri Rtd tu R1 song song voi R2.\n";
    }
```

Chuong trinh tinh gia tri Rtd tu R1 song song voi R2.

Nhap R1 = 5
Nhap R2 = 5

Tinh duoc Rtd = 2.5

2.4. Một số lưu ý về lập trình C/C++

Trong lập trình C++ phải tuân thủ các nguyên tắc cơ bản sau:

2.4.1. Thuật toán và lưu đồ thuật toán

Phải xác định đúng thuật toán ứng với vấn đề cần giải quyết. Phải vẽ lưu đồ thuật toán cho những vấn đề khó.

2.4.2. Nguyên tắc địa phương

Các biến địa phương trong hàm, thủ tục hoặc chu trình cho dù có trùng tên với biến toàn cục thì khi xử lý biến đó trong hàm hoặc thủ tục vẫn không làm thay đổi giá trị của biến toàn cục.

2.4.3. Nguyên tắc nhất quán

Dữ liệu thế nào thì phải thao tác thế ấy. Cần sớm phát hiện những mâu thuẫn giữa cấu trúc dữ liệu và cách thao tác để kịp thời khắc phục.

2.4.4. Cấu trúc chương trình phải phù hợp với quy mô chương trình

Chọn cấu trúc chương trình phải phù hợp với quy mô của chương trình. Vấn đề đơn giản thì cần sử dụng cấu trúc đơn giản. Nhưng vấn đề phức tạp thì cần ứng dụng cấu trúc lập trình hướng đối tượng để giải quyết cho từng đối tượng riêng, nhằm dễ quản lý và sửa lỗi.

2.5. Câu hỏi ôn tập chương 2

Câu 2.1. Có thể lập trình C++ trên những chương trình biên dich nào?

Câu 2.2. Các chương trình biên dịch hoạt động trên những môi trường nào?

Câu 2.3. Có thể tổ chức chương trình C++ theo những dạng nào? Cho ví dụ minh họa?

Chương 3. BIẾN, HẰNG VÀ KIỂU DỮ LIỆU

3.2. Kiểu Dữ Liệu, Biến và Hằng Số

3.2.1. Tên biến, hằng, hàm…

Một tên hợp lệ do người lập trình đặt là một chuỗi gồm các chữ cái, chữ số hoặc kí tự gạch dưới. Chiều dài của một tên là không giới hạn.

Kí tự trống, các kí tự đánh dấu đều không thể có mặt trong một tên. Chỉ có chữ cái, chữ số và kí tự gạch dưới là được phép. Thêm vào đó, một tên biến luôn phải bắt đầu bằng một chữ cái. Chúng cũng có thể bắt đầu bằng kí tự gạch dưới (_) nhưng kí tự này thường được dành cho các liên kết bên ngoài (external link). Không bao giờ chúng bắt đầu bằng một chữ số.

Một luật nữa mà ta phải quan tâm đến khi tạo ra các tên của riêng mình là chúng không được trùng với bất kì từ khoá nào của ngôn ngữ hay của trình dịch, ví dụ các tên sau đây luôn luôn được coi là từ khoá theo chuẩn ANSI-C++ và do vậy chúng không thể được dùng để đặt tên

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

Thêm vào đó, một số biểu diễn khác của các toán tử (operator) cũng không được dùng làm tên vì chúng là những từ được dành riêng trong một số trường hợp.

and, and_eq, bitand, bitor, compl, not, not_eq, or, or_eq, xor, xor_eq.

Trình dịch của ta có thể thêm một từ dành riêng đặc trưng khác. Ví dụ, rất nhiều trình dịch 16 bit (như các trình dịch cho DOS) còn có thể các từ khoá far, huge và near.

Chú ý: Ngôn ngữ C++ phân biệt chữ hoa chữ thường. Do vậy biến RESULT khác với result cũng như Result.

3.2.2. Các kiểu dữ liệu

Khi lập trình, chúng ta lưu trữ các biến trong bộ nhớ của máy tính nhưng máy tính phải biết chúng ta muốn lưu trữ gì trong chúng vì các kiểu dữ liệu khác nhau sẽ cần lượng bộ nhớ khác nhau.

Bộ nhớ của máy tính chúng ta được tổ chức thành các byte. Một byte là lượng bộ nhớ nhỏ nhất mà chúng ta có thể quản lí. Một byte có thể dùng để lưu trữ một loại dữ liệu nhỏ như là kiểu số nguyên từ 0 đến 255 hay một kí tự. Nhưng máy tính có thể xử lý các kiểu dữ liệu phức tạp hơn bằng cách gộp nhiều byte lại với nhau, như số nguyên dài hay số thập phân.

Danh sách các kiểu dữ liệu cơ bản trong C++ cũng như miền giá trị mà chúng có thể biểu diễn:

Tên	Số byte	Mô tả	Miền giá trị	
char	1	Kí tự hay kiểu số nguyên 8-bit	unsigned char; char	-127 to 127
			signed char	0 to 255
short	2	kiểu số nguyên 16-bit	-32,763 to 32,762	
long	4	kiểu số nguyên 32-bit	-2,147,483,647 to 2,147,483,647	
int	*	Số nguyên. Độ dài của nó phụ thuộc vào hệ thống, như trong MS-DOS nó là 16-bit, trên Windows 9x/2000/NT là 32 bit...	int; signed int; short int; signed shhort int	-32,767 to 32,767
			unsigned int unsigned shhort int	0 to 65,535
			long int	-2,147,483,647 to 2,147,483,647
float	4	Dạng dấu phẩy động	3.4e + / - 38 (7 digits)	
double	8	Dạng dấu phẩy động với độ chính xác gấp đôi	1.7e + / - 308 (15 digits)	
long double	10	Dạng dấu phẩy động với độ chính xác hơn nữa	1.2e + / - 4932 (19 digits)	
bool	1	Giá trị logic. Nó mới được thêm vào chuẩn ANSI-C++. Bởi vậy không phải tất cả các trình dịch đều hỗ trợ nó.	**true** hoặc **false**	

Ngoài các kiểu dữ liệu cơ bản nói trên còn tồn tại các con trỏ và các tham số không kiểu (void) mà chúng ta sẽ xem xét sau.

3.2.3. Biến (Variable)

3.2.3.1. Khai báo một biến

Biến là một vị trí có tên trong bộ nhớ được sử dụng để giữ một giá trị và có thể thay đổi bởi chương trình. Tất cả các biến phải được khai báo trước khi sử dụng. Dạng tổng quát của một khai báo biến như sau:

type variable_list;

Khai báo trên type phải là một kiểu dữ liệu hợp lệ, variable_list bao gồm một hoặc nhiều tên định danh phân biệt bởi dấu phẩy. Ví dụ một vài cách khai báo.

int i,j,l; short int si; unsigned int ui; double balance, profit, loss;

Nếu muốn khai báo một vài biến có cùng một kiểu và ta muốn tiết kiệm công sức viết ta có thể khai báo chúng trên một dòng, ngăn cách các tên bằng dấu phẩy. Ví dụ

int a, b, c;

khai báo ba biến kiểu int (a,b và c) và hoàn toàn tương đương với :

int a;
int b;
int c;

Các kiểu số nguyên (char, short, long and int) có thể là số có dấu hay không dấu tuỳ theo miền giá trị mà chúng ta cần biểu diễn. Vì vậy khi xác định một kiểu số nguyên chúng ta đặt từ khoá signed hoặc unsigned trước tên kiểu dữ liệu. Ví dụ:

unsigned short x;

signed int y;

Nếu ta không chỉ rõ signed hay unsigned nó sẽ được coi là có dấu, vì vậy trong khai báo thứ hai chúng ta có thể viết :

int y;

cũng hoàn toàn tương đương với dòng khai báo ở trên. Trong thực tế, rất ít khi người ta dùng đến từ khoá signed. Ngoại lệ duy nhất của luật này kiểu char. Trong chuẩn ANSI-C++ nó là kiểu dữ liệu khác với signed char và unsigned char.

Để có thể thấy rõ hơn việc khai báo trong chương trình, chúng ta sẽ xem xét một đoạn mã C++ ví dụ như sau: (Ví dụ 3.1)

```
// Chuong trinh Khai bao bien
#include <iostream.h>
int main ()
{
  // Khai bao bien
  int a, b;
  int result;
  // tinh toan
  a = 5; b = 2;
  a = a + 1;
  result = a - b;
  // In ket qua
  cout<<"a = "<<a;
  cout<<"b = "<<b;
  cout<< "result="<<result;
  return 0;          // Ket thuc chuong trinh
}
```

```
a = 6
b = 2
result=4
```

3.2.3.2. Khởi tạo các biến

Khi khai báo một biến, giá trị của nó mặc nhiên là không xác định. Nhưng có thể ta sẽ muốn nó mang một giá trị xác định khi được khai báo. Để làm điều đó, ta chỉ cần viết dấu bằng và giá trị ta muốn biến đó sẽ mang:

type variable_name = value;

Ví dụ, nếu chúng ta muốn khai báo một biến int là a chứa giá trị 5 ngay từ khi khởi tạo, chúng ta sẽ viết :

int a = 5;

Bổ xung vào cách khởi tạo kiểu C này, C++ còn có thêm một cách mới để khởi tạo biến bằng cách bọc một cặp ngoặc đơn sau giá trị khởi tạo. Ví dụ :

int a (0);

Cả hai cách đều hợp lệ trong C++.

3.2.3.3. Vị trí khai báo biến:

Biến được khai báo tại bốn vị trí: bên trong hàm hoặc khối lệnh gọi là biến cục bộ (local variable), trong định nghĩa tham số hàm (tham số hình thức), biến toàn cục (global variable) và biến ngoài (external).

Tất cả các biến mà chúng ta sẽ sử dụng đều phải được khai báo trước. Một điểm khác biết giữa Cvà C++ là trong C++ chúng ta có thể khai báo biến ở bất kì nơi nào trong chương trình, thậm chí là ngay ở giữa các lệnh thực hiện chứ không chỉ là ở đầu khối lệnh như ở trong C. Mặc dù vậy chúng ta vẫn nên theo cách của ngôn ngữ C khi khai báo các biến bởi vì nó sẽ rất hữu dụng khi cần sửa chữa một chương trình có tất cả các phần khai báo được gộp lại với nhau.

a. Biến cục bộ (Local Variable)

Là biến được khai báo bên trong hàm hoặc trong khối lệnh. Biến cục bộ chỉ được tham chiếu bởi những câu lệnh bên trong hàm (khối lệnh), cụ thể là bên trong khối lệnh khai báo nó. Biến cục bộ chỉ tồn tại trong khối khai báo nó và nó sẽ tự động hủy khi chương trình thoát khỏi hàm hoặc khối lệnh. Biến cục bộ phải được khai báo đầu mỗi khối.

Ví dụ 3.2:

```
#include<iostream.h>
#include<conio.h>
void main()
{char s[15];
 clrscr();
 cout<<"Nhap ten: "; cin>>s;
 if((s!=""))
   { char s[20];
     cout<<"Nhap ho: "; cin>>s;
     cout<<"Ho va ten: "<<s;
   }
 cout<<"_"<<s;
 getch();
 return;
}
```

```
Nhap ten: A
Nhap ho: Nguyen_Van
Ho va ten: Nguyen_Van_A
```

Biến s[20] ở trên là biến cục bộ được khai báo trong khối lệnh if và nó chỉ được sử dụng trong khối lệnh if. Các tham chiếu ngoài khối lệnh này (mặc dầu vẫn còn trong hàm) không hợp lệ.

Điểm thuận lợi của khai báo biến cục bộ là bộ nhớ của biến chỉ được cấp phát khi cần. Khai báo biến bên trong mã lệnh giúp ngăn chặn những tác động không mong muốn. Ta có thể khởi tạo giá trị cho biến tại lúc khai báo.

b. Biến toàn cục (Global Variable)

Biến toàn cục được sử dụng trong toàn bộ chương trình. Biến toàn cục sẽ lưu trữ giá trị suốt chương trình. Ta có thể khai báo biến toàn cục bằng cách khai báo biến bên ngoài tất cả các hàm. Mọi biểu thức đều có thể truy cập đến biến này.

Ví dụ 3.3:

```
#include <iostream.h>
#include <stdio.h>
#include <conio.h>
int count;
void func1();
int func2();
void func3(int f3);
void main ()
 {     int a=200;
       clrscr();
       cout<<"KIEM TRA GIA TRI CAC BIEN\n";
       count=100;
       func1();
       count=func2();
       func3(a);
       cout<<"Gia tri cua bien count trong main: "<<count<<"\n";
       getch();
 }
void func1()
 {     int temp;
       temp=count;
       cout<<"Gia tri cua bien temp trong func1: "<<temp<<"\n";
 }
int func2()
 {     int count=222;
       cout<<"Gia tri cua bien count trong func2: "<<count<<"\n";
       return count;
 }
void func3(int f3)
 {     f3=f3-50;
       cout<<"Gia tri cua bien f3 trong func3: "<<f3<<"\n";
 }
```

```
KIEM TRA GIA TRI CAC BIEN
Gia tri cua bien temp trong func1: 100
Gia tri cua bien count trong func2: 222
Gia tri cua bien f3 trong func3: 150
Gia tri cua bien count trong main: 220
```

Hai hàm main và hàm func1 đều khai báo biến count và cả hai hàm này đều sử dụng nó. Tuy nhiên hàm func2 khai báo một biến cục bộ count . Nếu biến toàn cục và cục bộ có cùng tên, mọi thao tác sẽ thực hiện trên biến cục bộ và không có tác động nào trên biến toàn cục.

Sử dụng biến toàn cục trong trường hợp nhiều hàm trong chương trình sử dụng chung dữ liệu. Tuy nhiên ta tránh dùng biến toàn cục trong trường hợp không cần thiết bởi vì trình biên dịch sẽ dành bộ nhớ trong suốt quá trình thực thi cho biến toàn cục và sử dụng biến toàn cục làm mất tính tổng quát của chương trình. Hơn nữa, sử dụng biến toàn cục sẽ làm chương trình thường gặp lỗi bởi vì có tác động không biết hoặc không mong muốn đến nó.

c. Biến ngoài (external)

Biến ngoài (external). Các biến này không những được dùng trong một file mã nguồn mà còn trong tất cả các file được liên kết trong chương trình.

Trong C++ tầm hoạt động của một biến chính là khối lệnh mà nó được khai báo (một khối lệnh là một tập hợp các lệnh được gộp lại trong một bằng các ngoặc nhọn { }). Nếu nó được khai báo trong một hàm tầm hoạt động sẽ là hàm đó, còn nếu được khai báo trong vòng lặp thì tầm hoạt động sẽ chỉ là vòng lặp đó....

3.2.4. Hằng

3.2.4.1. Hằng số

Một hằng số là một giá trị bất kì mà chương trình không thay đổi được. Hằng có thể là kiểu dữ liệu bất kì.

Ngoài những số ở hệ cơ số 10 (cái mà tất cả chúng ta đều đã biết) C++ còn cho phép sử dụng các hằng số cơ số 8 và 16. Để biểu diễn một số hệ cơ số 8 chúng ta đặt trước nó kí tự 0, để biểu diễn số ở hệ cơ số 16 chúng ta đặt trước nó hai kí tự 0x. Ví dụ:

Các số thập phân (dạng dấu phẩy động), chúng biểu diễn các số với phần thập phân và có thể có số mũ. Chúng có thể bao gồm phần thập phân, kí tự e (biểu diễn 10 mũ...).

Cở số	Biểu diễn
Cơ số 10 (số nguyên)	1776
Cơ số 10 (thập phân)	3.14159
	6.02e23 // 6.02 x 10^{23}
	1.6e-19 // 1.6 x 10^{-19}
cơ số 8	0113
cơ số 16	0x4b

3.2.4.2. Hằng kí tự và xâu kí tự

Trong C++ còn tồn tại các hằng không phải kiểu số như:

'z' **Hằng ký tự**
"Hello world" **Hằng xâu ký tự**

Các kí tự được đặt trong dấu nháy đơn ('), các xâu kí tự được đặt trong dấu nháy kép (").

Khi viết các kí tự đơn hay các xâu kí tự cần phải đặt chúng trong các dấu nháy để phân biệt với các tên biến hay các từ khoá. Chú ý:

x 'x'

x là biến x trong khi 'x' là kí tự hằng 'x'.

Các kí tự đơn và các xâu kí tự có một tính chất riêng biệt là các mã điều khiển. Tất cả đều bắt đầu bằng dấu xổ ngược (\). Sau đây là danh sách các mã điều khiển đó:

n	xuống dòng
r	lùi về đầu dòng
t	kí tự tab
v	căn thẳng theo chiều dọc
b	backspace
f	sang trang
a	Kêu bíp
	dấu nháy đơn

'	dấu nháy kép
"	dấu hỏi
?	kí tự xổ ngược
\	

Thêm vào đó, để biểu diễn một mã ASCII ta cần sử dụng kí tự xổ ngược (\) tiếp theo đó là mã ASCII viết trong hệ cơ số 8 hay cơ số 16. Trong trường hợp đầu mã ASCII được viết ngay sau dấu sổ ngược, trong trường hợp thứ hai, để sử dụng số trong hệ cơ số 16 ta cần viết kí tự x trước số đó (ví dụ \x20 hay \x4A).

Các hằng chuỗi kí tự có thể được viết trên nhiều dòng nếu mỗi dòng được kết thúc bằng một dấu sổ ngược (\):

```
"chuoi tren \
 hai dong."
```

Ta có thể nối một vài hằng xâu kí tự ngăn cách bằng một hay vài dấu trống, kí tự tab, xuống dòng hay bất kì kí tự trống nào khác.

"cung" "mot" "chuoi"
"ki tu."

3.2.4.3. Định nghĩa các hằng (#define)

Ta có thể định nghĩa các hằng với tên mà mình muốn để có thể sử dụng thường xuyên mà không mất tài nguyên cho các biến bằng cách sử dụng chỉ thị #define. Đây là dạng của nó:

#define constname value

Trong thực tế việc duy nhất mà trình dịch làm khi nó tìm thấy một chỉ thị #define là thay thế các tên hằng tại bất kì chỗ nào chúng xuất hiện bằng giá trị mà chúng được định nghĩa. Vì vậy các hằng số #define được coi là các hằng số macro.

Chỉ thị #define không phải là một lệnh thực thi, nó là chỉ thị tiền xử lý (preprocessor), đó là lý do trình dịch coi cả dòng là một chỉ thị và dòng đó không cần kết thúc bằng dấu chấm phẩy (;). Nếu ta thêm dấu chấm phẩy vào cuối dòng, nó sẽ được coi là một phần của giá trị định nghĩa hằng.

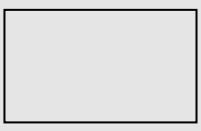
3.2.4.4. Khai báo các hằng (const)

Với tiền tố const ta có thể khai báo các hằng với một kiểu xác định như là ta làm với một biến

```
const int width = 100;
const to char tab = '\t';
const zip = 12440;
```

Ví dụ 3.4:

```
//Ve hinh chu nhat
#include<iostream.h>
#include<stdio.h>
#include<conio.h>
#define LEFT_TOP  0xDA
#define RIGHT_TOP 0xBF
#define HORIZ     0xC4
#define VERT      0xB3
#define LEFT_BOT  0xC0
```



aaaabbbcccc

```
#define RIGHT_BOT 0xD9
int main(void)
{
  clrscr();
  char i, j;
  /* draw the top of the box */
  putchar(LEFT_TOP);
  for (i=0; i<10; i++)   putchar(HORIZ);
  putchar(RIGHT_TOP);
  putchar('\n');
  /* draw the middle */
  for (i=0; i<4; i++)
  { putchar(VERT);
    for (j=0; j<10; j++) putchar(' ');
    putchar(VERT);
    putchar('\n');
  }
  /* draw the bottom */
  putchar(LEFT_BOT);
  for (i=0; i<10; i++)    putchar(HORIZ);
  putchar(RIGHT_BOT);
  putchar('\n');
  cout<<"aaaa"
  "bbb"  "cccc";
  getch();
  return 0;
}
```

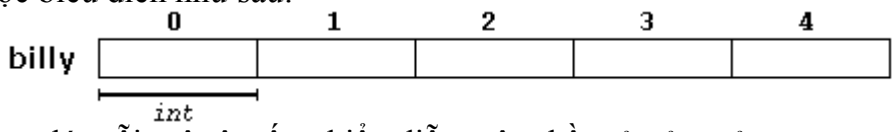
3.3. Dữ liệu dạng mảng

3.3.1. Mảng một chiều

3.3.1.1. Khái niệm mảng

Mảng là một dãy các phần tử có cùng kiểu được đặt liên tiếp trong bộ nhớ và có thể truy xuất đến từng phần tử bằng cách thêm một chỉ số vào sau tên của mảng.

Điều này có nghĩa là, ví dụ, chúng ta có thể lưu 5 giá trị kiểu **int** mà không cần phải khai báo 5 biến khác nhau.Ví dụ, một mảng chứa 5 giá trị nguyên kiểu **int** có tên là *billy* có thể được biểu diễn như sau:



trong đó mỗi một ô trống biểu diễn một phần tử của mảng, trong trường hợp này là các giá trị nguyên kiểu **int**. Chúng được đánh số từ **0** đến **4** vì phần tử đầu tiên của mảng luôn là **0** bất kể độ dài của nó là bao nhiêu.

Như bất kì biến nào khác, một mảng phải được khai báo trước khi có thể sử dụng. Một khai báo điển hình cho một mảng trong C++ như sau:

type name [*elements*];

trong đó *type* là một kiểu dữ liệu hợp lệ (**int**, **float**...), *name* là một tên biến hợp lệ và trường *elements* chỉ định mảng đó sẽ chứa bao nhiêu phần tử

Vì vậy, để khai báo *billy* như đã trình bày ở trên chúng ta chỉ cần một dòng đơn giản như sau:

int billy [5];

Chú ý: Trường *elements* bên trong cặp ngoặc [] phải là một giá trị hằng khi khai báo một mảng, vì mảng là một khối nhớ tĩnh có kích cỡ xác định và trình biên dịch phải có khả năng xác định xem cần bao nhiêu bộ nhớ để cấp phát cho mảng trước khi các lệnh có thể được thực hiện.

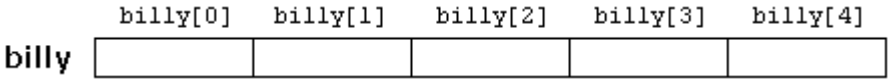
3.3.1.2. Khởi tạo một mảng.

Khi khai báo một mảng với tầm hoạt động địa phương (trong một hàm), theo mặc định nó sẽ không được khởi tạo, vì vậy nội dung của nó là không xác định cho đến khi chúng ra lưu các giá trị lên đó.

Nếu chúng ta khai báo một mảng toàn cục (bên ngoài tất cả các hàm) nó sẽ được khởi tạo và tất cả các phần tử được đặt bằng 0. Vì vậy nếu chúng ta khai báo mảng toàn cục:

int billy [5];

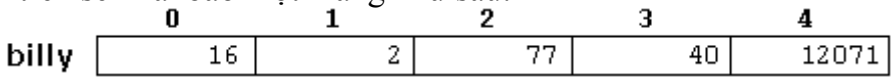
mọi phần tử của *billy* sẽ được khởi tạo là **0**:



Nhưng thêm vào đó, khi chúng ta khai báo một mảng, chúng ta có thể gán các giá trị khởi tạo cho từng phần tử của nó. Ví dụ:

int billy [5] = { 16, 2, 77, 40, 12071 };

lệnh trên sẽ khai báo một mảng như sau:



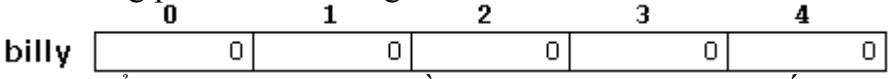
Số phần tử trong mảng mà chúng ta khởi tạo với cặp ngoặc nhọn { } phải bằng số phần tử của mảng đã được khai báo với cặp ngoặc vuông [ ]. Bởi vì điều này có thể được coi là một sự lặp lại không cần thiết nên C++ cho phép để trống giữa cặp ngoặc vuông, kích thước của mảng được xác định bằng số giá trị giữa cặp ngoặc nhọn.

3.3.1.3. Truy xuất đến các phần tử của mảng.

Ở bất kì điểm nào của chương trình trong tầm hoạt động của mảng, chúng ta có thể truy xuất từng phần tử của mảng để đọc hay chỉnh sửa như là đối với một biến bình thường. Cấu trúc của nó như sau:

name[*index*]

Như ở trong ví dụ trước ta có mảng *billy* gồm 5 phần tử có kiểu **int**, chúng ta có thể truy xuất đến từng phần tử của mảng như sau:



Ví dụ, để lưu giá trị 75 vào phần tử thứ ba của *billy* ta viết như sau:

billy[2] = 75;

và, ví dụ, để gán giá trị của phần tử thứ 3 của billy cho biến **a**, chúng ta viết:

a = billy[2];

Vì vậy, xét về mọi phương diện, biểu thức **billy[2]** giống như bất kì một biến kiểu **int.**

Chú ý rằng phần tử thứ ba của **billy** là **billy[2]**, vì mảng bắt đầu từ chỉ số 0. Vì vậy, phần tử cuối cùng sẽ là **billy[4]**. Vì vậy nếu chúng ta viết **billy[5]**, chúng ta sẽ truy xuất đến phần tử thứ 6 của mảng và vượt quá giới hạn của mảng.

Trong C++, việc vượt quá giới hạn chỉ số của mảng là hoàn toàn hợp lệ, tuy nhiên nó có thể gây ra những vấn đề thực sự khó phát hiện bởi vì chúng không tạo ra những lỗi trong quá trình dịch nhưng chúng có thể tạo ra những kết quả không mong muốn trong quá trình thực hiện. Nguyên nhân của việc này sẽ được nói đến kĩ hơn khi chúng ta bắt đầu sử dụng con trỏ.

Cần phải nhấn mạnh rằng chúng ta sử dụng cặp ngoặc vuông cho hai tác vụ: đầu tiên là đặt kích thước cho mảng khi khai báo chúng và thứ hai, để chỉ định chỉ số cho một phần tử cụ thể của mảng khi xem xét đến nó.

```
int billy[5];       // khai báo một mảng mới.
billy[2] = 75;      // truy xuất đến một phần tử của mảng.
```

Một vài thao tác hợp lệ khác với mảng:

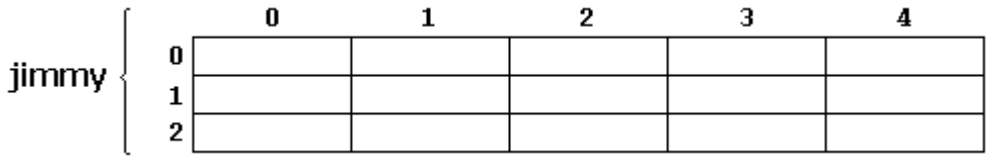
```
billy[0] = a;
billy[a] = 75;
b = billy [a+2];
billy[billy[a]] = billy[2] + 5;
```

```
// ví dụ về mảng
#include <iostream.h>

int billy [] = {16, 2, 77, 40, 12071};
int n, result=0;

int main ()
{
  for ( n=0 ; n<5 ; n++ )
  {
    result += billy[n];
  }
  cout << result;
  return 0;
}
```

12206

3.3.2. Mảng nhiều chiều.

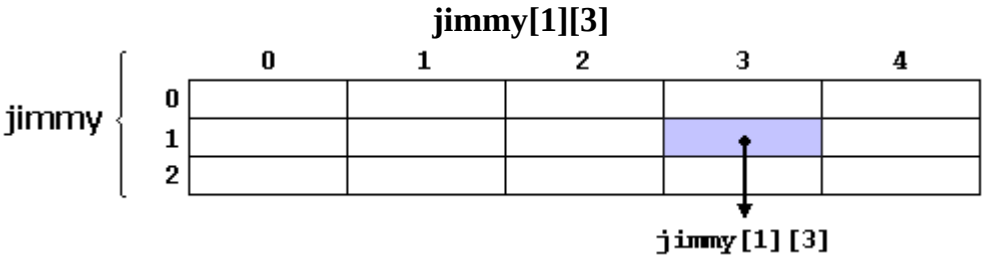
Mảng nhiều chiều có thể được coi như mảng của mảng, ví dụ, một mảng hai chiều có thể được tưởng tượng như là một bảng hai chiều gồm các phần tử có kiểu dữ liệu cụ thể và giống nhau.



jimmy biểu diễn một mảng hai chiều kích thước 3x5 có kiểu **int**. Cách khai báo mảng này như sau:

```
int jimmy [3][5];
```

và, ví dụ, cách để truy xuất đến phần tử thứ hai theo chiều dọc và thứ tư theo chiều ngang trong một biểu thức như sau:



(hãy nhớ rằng chỉ số của mảng luôn bắt đầu từ **0**).

Mảng nhiều chiều không bị giới hạn bởi hai chỉ số (hai chiều), Chúng có thể chứa bao nhiều chỉ số tùy thích mặc dù ít khí cần phải dùng đến mảng lớn hơn 3 chiều. Hãy thử xem xét lượng bộ nhớ mà một mảng có nhiều chỉ số cần đến. Ví dụ:

char century [100][365][24][60][60];

gán một giá trị **char** cho mỗi giây trong một thế kỉ, phải cần đến hơn 3 tỷ giá trị **chars**! Chúng ta sẽ phải cần khoảng 3GB RAM để khai báo nó.

Mảng nhiều chiều thực ra là một khái niệm trừu tượng vì chúng ta có thể có kết quả tương tự với mảng một chiều bằng một thao tác đơn giản giữa các chỉ số của nó:

int jimmy [3][5]; tương đương với

int jimmy [15]; (3 * 5 = 15)

Dưới đây là hai ví dụ với cùng một kết quả như nhau, một sử dụng mảng hai chiều và một sử dụng mảng một chiều:

```
// multidimensional array
#include <iostream.h>
#define WIDTH 5
#define HEIGHT 3
int jimmy [HEIGHT][WIDTH];
int n,m;

int main ()
{
  for (n=0;n<HEIGHT;n++)
    for (m=0;m<WIDTH;m++)
    {
      jimmy[n][m]=(n+1)*(m+1);
    }
  return 0;
}
```

```
// pseudo-multidimensional array
#include <iostream.h>
#define WIDTH 5
#define HEIGHT 3
int jimmy [HEIGHT * WIDTH];
int n,m;

int main ()
{
  for (n=0;n<HEIGHT;n++)
    for (m=0;m<WIDTH;m++)
    {
      jimmy[n * WIDTH + m]=(n+1)*(m+1);
    }
  return 0;
}
```

không một chương trình nào viết gì ra màn hình nhưng cả hai đều gán giá trị vào khối nhớ có tên **jimmy** theo cách sau:

jimmy

	0	1	2	3	4
0	1	2	3	4	5
1	2	4	6	8	10
2	3	6	9	12	15

Chúng ta đã định nghĩa hằng (**#define**) để đơn giản hóa những chỉnh sửa sau này của chương trình, ví dụ, trong trường hợp chúng ta quyết định tăng kích thước của mảng với chiều cao là 4 thay vì là 3, chúng ta chỉ cần thay đổi dòng:

#define HEIGHT 3

thành

#define HEIGHT 4

và không phải có thêm sự thay đổi nào nữa đối với chương trình.

3.3.3. Dùng mảng làm tham số.(đọc thêm)

Vào một lúc nào đó có thể chúng ta cần phải truyền một mảng tới một hàm như là một tham số. Trong C++, việc truyền theo tham số giá trị một khối nhớ là không hợp lệ, ngay cả khi nó được tổ chức thành một mảng. Tuy nhiên chúng ta lại được phép truyền địa chỉ của nó, việc này cũng tạo ra kết quả thực tế giống thao tác ở trên nhưng lại nhanh hơn nhiều và hiệu quả hơn.

Để có thể nhận mảng là tham số thì điều duy nhất chúng ta phải làm khi khai báo hàm là chỉ định trong phần tham số kiểu dữ liệu cơ bản của mảng, tên mảng và cặp ngoặc vuông trống. Ví dụ, hàm sau:

void procedure (int arg[])

nhận vào một tham số có kiểu "mảng của **char**" và có tên **arg**. Để truyền tham số cho hàm này một mảng được khai báo:

int myarray [40];

chỉ cần gọi hàm như sau:

procedure (myarray);

Dưới đây là một ví dụ cụ thể

```
// arrays as parameters
#include <iostream.h>

void printarray (int arg[], int length) {
  for (int n=0; n<length; n++)
    cout << arg[n] << " ";
  cout << "\n";
}

int main ()
{
  int firstarray[] = {5, 10, 15};
  int secondarray[] = {2, 4, 6, 8, 10};
  printarray (firstarray,3);
  printarray (secondarray,5);
  return 0;
}
```

5 10 15
2 4 6 8 10

Như ta có thể thấy, tham số đầu tiên (**int arg[]**) chấp nhận mọi mảng có kiểu cơ bản là **int**, bất kể độ dài của nó là bao nhiêu, vì vậy cần thiết phải có tham số thứ hai để báo cho hàm này biết độ dài của mảng mà chúng ta truyền cho nó.

Trong phần khai báo hàm chúng ta cũng có thể dùng tham số là các mảng nhiều chiều. Cấu trúc của mảng 3 chiều như sau:

*base_type***[][***depth***][***depth***]**

ví dụ, một hàm với tham số là mảng nhiều chiều có thể như sau:

void procedure (int myarray[][3][4])

chú ý rằng cặp ngoặc vuông đầu tiên để trống nhưng các cặp ngoặc sau thì không. Ta luôn luôn phải làm vậy vì trình biên dịch C++ phải có khả năng xác định độ lớn của các chiều thêm vào của mảng.

3.4. Xâu Ký Tự

3.4.1. Khái miện về xâu kí tự

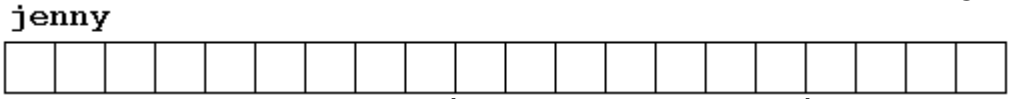
Trong tất cả các chương trình chúng ta đã thấy cho đến giờ, chúng ta chỉ sử dụng các biến kiểu số, chỉ dùng để biểu diễn các số. Nhưng bên cạnh các biến kiểu số còn có các xâu kí tự, chúng cho phép chúng ta biểu diễn các chuỗi kí tự như là các từ, câu, đoạn văn bản... Cho đến giờ chúng ta mới chỉ dùng chúng dưới dạng hằng chứ chứa quan tâm đến các biến có thể chứa chúng.

Trong C++ không có kiểu dữ liệu *cơ bản* để lưu các xâu kí tự. Để có thể thỏa mãn nhu cầu này, người ta sử dụng mảng có kiểu **char**. Hãy nhớ rằng kiểu dữ liệu này (**char**) chỉ có thể lưu trữ một kí tự đơn, bởi vậy nó được dùng để tạo ra xâu của các kí tự đơn.

Ví dụ, mảng sau (hay là xâu kí tự):

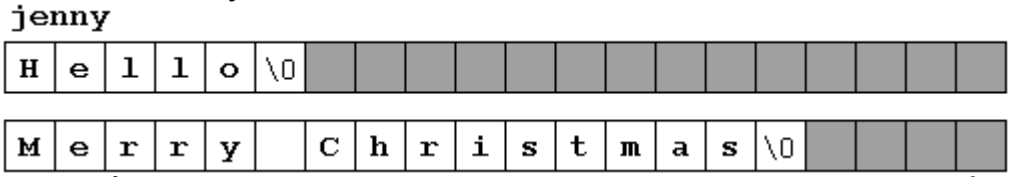
char jenny [20];

có thể lưu một xâu kí tự với độ dài cực đại là 20 kí tự. Ta có thể tưởng tượng nó như sau:



Kích thước cực đại này không cần phải luôn luôn dùng đến. Ví dụ, **jenny** có thể lưu xâu "Hello" hay "Merry christmas". Vì các mảng kí tự có thể lưu các xâu kí tự ngắn hơn độ dài của nó, trong C++ đã có một quy ước để kết thúc một nội dung của một xâu kí tự bằng một kí tự null, có thể được viết là '\0'.

Chúng ta có thể biểu diễn **jenny** (một mảng có 20 phần tử kiểu **char**) khi lưu trữ xâu kí tự "Hello" và "Merry Christmas" theo cách sau:



Chú ý rằng sau nội dung của xâu, một kí tự null (**'\0'**) được dùng để báo hiệu kết thúc xâu. Những ô màu xám biểu diễn những giá trị không xác định.

3.4.2. Khởi tạo các xâu kí tự.

Vì những xâu kí tự là những mảng bình thường nên chúng cũng như các mảng khác. Ví dụ, nếu chúng ta muốn khởi tạo một xâu kí tự với những giá trị xác định chúng ta có thể làm điều đó tương tự như với các mảng khác:

```
char mystring[] = { 'H', 'e', 'l', 'l', 'o', '\0' };
```

Tuy nhiên, chúng ta có thể khởi tạo giá trị cho một xâu kí tự bằng cách khác: sử dụng các *hằng xâu kí tự*.

Trong các biểu thức chúng ta đã sử dụng trong các ví dụ trong các chương trước các hằng xâu kí tự để xuất hiện vài lần. Chúng được biểu diễn trong cặp ngoặc kép ("), ví dụ:

"the result is: "

là một hằng xâu kí tự chúng ta sử dụng ở một số chỗ.

Không giống như dấu nháy đơn (') cho phép biểu diễn hằng kí tự, cặp ngoặc kép (") là hằng biểu diễn một chuỗi kí tự liên tiếp, và ở cuối chuỗi một kí tự null ('\0') luôn được tự động thêm vào.

Vì vậy chúng ta có thể khởi tạo xâu **mystring** theo một trong hai cách sau đây:

char mystring [] = { 'H', 'e', 'l', 'l', 'o', '\0' };

char mystring [] = "Hello";

Trong cả hai trường hợp mảng (hay xâu kí tự) **mystring** được khai báo với kích thước 6 kí tự: 5 kí tự biểu diễn **Hello** cộng với một kí tự null.

Việc gán nhiều hằng như việc sử dụng dấu ngoặc kép (") chỉ hợp lệ khi *khởi tạo* mảng, tức là lúc khai báo mảng. Các biểu thức trong chương trình như:

mystring = "Hello";

mystring[] = "Hello";

là không hợp lệ, cả câu lệnh dưới đây cũng vậy:

mystring = { 'H', 'e', 'l', 'l', 'o', '\0' };

Vậy hãy nhớ: Chúng ta chỉ có thể "gán" nhiều hằng cho một mảng vào lúc khởi tạo nó. Nguyên nhân là một thao tác gán (=) không thể nhận vế trái là cả một mảng mà chỉ có thể nhận một trong những phần tử của nó. Vào thời điểm khởi tạo mảng là một trường hợp đặc biệt, vì nó không thực sự là một lệnh gán mặc dù nó sử dụng dấu bằng (=).

3.4.3. Gán giá trị cho xâu kí tự

Vì vế trái của một lệnh gán chỉ có thể là một phần tử của mảng chứ không thể là cả mảng, chúng ta có thể gán một xâu kí tự cho một mảng kiểu **char** sử dụng một phương pháp như sau:

mystring[0] = 'H';
mystring[1] = 'e';
mystring[2] = 'l';
mystring[3] = 'l';
mystring[4] = 'o';
mystring[5] = '\0';

Nhưng rõ ràng đây không phải là một phương pháp thực tế. Để gán giá trị cho một xâu kí tự, chúng ta có thể sử dụng loạt hàm kiểu **strcpy** (**str**ing **c**o**py**), hàm này được định nghĩa trong string.h và có thể được gọi như sau:

strcpy (*string1***,** *string2***);**

Lệnh này copy nội dung của *string2* sang *string1*. *string2* có thể là một mảng, con trỏ hay một hằng xâu kí tự, bởi vậy lệnh sau đây là một cách đúng để gán xâu hằng **"Hello"** cho **mystring**:

strcpy (mystring, "Hello");

Ví dụ:

```
// setting value to string
#include <iostream.h>
#include <string.h>

int main ()
{
  char szMyName [20];
  strcpy (szMyName,"J. Soulie");
  cout << szMyName;
```

J. Soulie

```
  return 0;
}
```

Để ý rằng chúng ta phải include file **<string.h>** để có thể sử dụng hàm **strcpy**.

Mặc dù chúng ta luôn có thể viết một hàm đơn giản như hàm **setstring** dưới đây để thực hiện một thao tác giống như **strcpy**:

```
// setting value to string
#include <iostream.h>

void setstring (char szOut [], char szIn [])
{
  int n=0;
  do {
    szOut[n] = szIn[n];
    n++;
  } while (szIn[n] != 0);
}

int main ()
{
  char szMyName [20];
  setstring (szMyName,"J. Soulie");
  cout << szMyName;
  return 0;
}
```

J. Soulie

Một phương thức thường dùng khác để gán giá trị cho một mảng là sử dụng trực tiếp dòng nhập dữ liệu (**cin**). Trong trường hợp này giá trị của xâu kí tự được gán bởi người dùng trong quá trình chương trình thực hiện.

Khi **cin** được sử dụng với các xâu kí tự nó thường được dùng với phương thức **getline** của nó, phương thức này có thể được gọi như sau:

cin.getline (char *buffer***[], int** *length***, char** *delimiter* **= ' \n');**

trong đó ***buffer*** (bộ đệm) là địa chỉ nơi sẽ lưu trữ dữ liệu vào (như là một mảng chẳng hạn), ***length*** là độ dài cực đại của bộ đệm (kích thước của mảng) và ***delimiter*** là kí tự được dùng để kết thúc việc nhập, mặc định - nếu chúng ta không dùng tham số này - sẽ là kí tự xuống dòng ('**\n**').

Ví dụ sau đây lặp lại tất cả những gì ta gõ trên bàn phím. Nó rất đơn giản nhưng là một ví dụ cho thấy ta có thể sử dụng **cin.getline** với các xâu kí tự như thế nào:

```
// cin with strings
#include <iostream.h>

int main ()
{
  char mybuffer [100];
  cout << "What's your name? ";
  cin.getline (mybuffer,100);
  cout << "Hello " << mybuffer << ".\n";
```

What's your name? Juan
Hello Juan.
Which is your favourite team?
Inter Milan
I like Inter Milan too.

```
  cout << "Which is your favourite team? ";
  cin.getline (mybuffer,100);
  cout << "I like " << mybuffer << " too.\n";
  return 0;
}
```

Chú ý trong cả hai lời gọi **cin.getline** chúng ta sử dụng cùng một biến xâu (**mybuffer**). Những gì chương trình làm trong lời gọi thứ hai đơn giản là thay thế nội dung của **buffer** trong lời gọi cũ bằng nội dung mới.

Chúng ta đã biết sử dụng toán tử >> để nhận dữ liệu trực tiếp từ bàn phím. Phương thức này có thể được dùng với các xâu kí tự thay cho **cin.getline**. Ví dụ, trong chương trình của chúng ta, khi chúng ta muốn nhận dữ liệu từ người dùng chúng ta có thể viết:

cin >> mybuffer;

lệnh này sẽ làm việc như nó có những hạn chế sau mà **cin.getline** không có:

- Nó chỉ có thể nhận những từ đơn (không nhận được cả câu) vì phương thức này sử dụng kí tự trống(bao gồm cả dấu cách, dấu tab và dấu xuống dòng) làm dấu hiệu kết thúc..
- Nó không cho phép chỉ định kích thước cho bộ đệm. Chương trình của ta có thể chạy không ổn định nếu dữ liệu vào lớn hơn kích cỡ của mảng chứa nó.

Vì những nguyên nhân trên, khi muốn nhập vào các xâu kí tự ta nên sử dụng **cin.getline** thay vì **cin >>**.

3.4.4. Chuyển đổi xâu kí tự sang các kiểu khác.

Vì một xâu kí tự có thể biểu diễn nhiều kiểu dữ liệu khác như dạng số nên việc chuyển đổi nội dung như vậy sang dạng số là rất hữu ích. Ví dụ, một xâu có thể mang giá trị **"1977"**nhưng đó là một chuỗi gồm 5 kí tự (kể cả kí tự null) và không dễ gì chuyển thành một số nguyên. Vì vậy thư viện **cstdlib** (**stdlib.h**) đã cung cấp 3 macro/hàm hữu ích sau:

- **atoi:** chuyển xâu thành kiểu **int**.
- **atol:** chuyển xâu thành kiểu **long**.
- **atof:** chuyển xâu thành kiểu **float**.

Tất cả các hàm này nhận một tham số và trả về giá trị số (int, long hoặc float). Các hàm này khi kết hợp với phương thức **getline** của **cin** là một cách đáng tin cậy hơn phương thức **cin>>** cổ điển khi yêu cầu người sử dụng nhập vào một số:

```
// cin and ato* functions
#include <iostream.h>
#include <stdlib.h>

int main ()
{
  char mybuffer [100];
  float price;
  int quantity;
  cout << "Enter price: ";
  cin.getline (mybuffer,100);
  price = atof (mybuffer);
  cout << "Enter quantity: ";
```

Enter price: 2.75
Enter quantity: 21
Total price: 57.75

```
    cin.getline (mybuffer,100);
    quantity = atoi (mybuffer);
    cout << "Total price: " << price*quantity;
    return 0;
  }
```

Các hàm để thao tác trên chuỗi

Thư viện **cstring** (string.h) không chỉ có hàm **strcpy** mà còn có nhiều hàm khác để thao tác trên chuỗi. Dưới đây là giới thiệu lướt qua của các hàm thông dụng nhất:

strcat: char* strcat (char* *dest***, const char*** *src***);**

Gắn thêm chuỗi *src* vào phía cuối của *dest*. Trả về *dest.*

strcmp: int strcmp (const char* *string1***, const char*** *string2***);**

So sánh hai xâu *string1* và *string2*. Trả về 0 nếu hai xâu là bằng nhau.

strcpy: char* strcpy (char* *dest***, const char*** *src***);**

Copy nội dung của *src* cho *dest*. Trả về *dest.*

strlen: size_t strlen (const char* *string***);**

Trả về độ dài của *string*.

Chú ý: **char*** hoàn toàn tương đương với **char[];**

3.5. Các Toán Tử

Qua bài trước chúng ta đã biết đến sự tồn tại của các biến và các hằng. Trong C++, để xử lý, tính toán ta sử dụng các toán tử, đó là các từ khoá và các dấu không có trong bảng chữ cái nhưng lại có trên hầu hết các bàn phím trên thế giới. Hiểu biết về chúng là rất quan trọng vì đây là một trong những thành phần cơ bản của ngôn ngữ C++.

3.5.1. Toán tử tính toán giá trị số học

3.5.1.1. Toán tử gán (=).

Toán tử gán dùng để gán một giá trị nào đó cho một biến, như:

```
a = 5;
```

gán giá trị nguyên 5 cho biến a. Vế trái bắt buộc phải là một biến còn vế phải có thể là bất kì hằng, biến hay kết quả của một biểu thức hoặc giá trị trả về của một hàm.

Cần phải nhấn mạnh rằng toán tử gán luôn được thực hiện từ trái sang phải và không bao giờ đảo ngược.

```
a = b;
```

gán giá trị của biến a bằng giá trị đang chứa trong biến b. Chú ý rằng chúng ta chỉ gán giá trị của b cho a và sự thay đổi của b sau đó sẽ không ảnh hưởng đến giá trị của a.

Một thuộc tính của toán tử gán trong C++ góp phần giúp nó vượt lên các ngôn ngữ lập trình khác là việc cho phép vế phải có thể chứa các phép gán khác. Ví dụ:

```
a = 2 + (b = 5);
```

tương đương với

```
b = 5;
a = 2 + b;
```

Vì vậy biểu thức sau cũng hợp lệ trong C++

```
a = b = c = 5;
```

gán giá trị 5 cho cả ba biến a, b và c

*3.5.1.2. Các toán tử số học (+, -, *, /, %)*

Năm toán tử số học được hỗ trợ bởi ngôn ngữ là:

cộng

trừ

nhân

chia

lấy phần dư (trong phép chia)

Thứ tự thực hiện các toán tử này cũng giống như chúng được thực hiện trong toán học. Điều duy nhất có vẻ hơi lạ đối với ta là phép lấy phần dư, ký hiệu bằng dấu phần trăm (%). Đây chính là phép toán lấy phần dư trong phép chia hai số nguyên với nhau. Ví dụ, nếu a = 11 % 3; biến a sẽ mang giá trị 2 vì 11 = 3*3 +2.

*3.5.1.3. Các toán tử gán phức hợp (+=, -=, *=, /=, %=, >>=, <<=, &=, ^=, |=)*

Một đặc tính của ngôn ngữ C++ làm cho nó nổi tiếng là một ngôn ngữ súc tích chính là các toán tử gán phức hợp cho phép chỉnh sửa giá trị của một biến với một trong những toán tử cơ bản sau:

value += increase; tương đương với **value = value + increase;**

a -= 5; tương đương với **a = a - 5;**

a /= b; tương đương với **a = a / b;**

price *= units + 1; tương đương với **price = price * (units + 1);**

và tương tự cho tất cả các toán tử khác.

3.5.1.4. Tăng và giảm.

Một ví dụ khác của việc tiết kiệm khi viết mã lệnh là toán tử tăng (++) và giảm (--). Chúng tăng hoặc giảm giá trị chứa trong một biến đi 1. Chúng tương đương với +=1 hoặc -=1. Vì vậy, các dòng sau là tương đương:

a++;

a+=1;

a=a+1;

Một tính chất của toán tử này là nó có thể là tiền tố hoặc hậu tố, có nghĩa là có thể viết trước tên biến (++a) hoặc sau (a++) và mặc dù trong hai biểu thức rất đơn giản đó nó có cùng ý nghĩa nhưng trong các thao tác khác khi mà kết quả của việc tăng hay giảm được sử dụng trong một biểu thức thì chúng có thể có một khác biệt quan trọng về ý nghĩa:

+Trong trường hợp toán tử được sử dụng như là một tiền tố (++a) giá trị được tăng trước khi biểu thức được tính và giá trị đã tăng được sử dụng trong biểu thức;

+ Trong trường hợp ngược lại (a++) giá trị trong biến a được tăng sau khi đã tính toán. Hãy chú ý sự khác biệt :

Ví dụ 1	**Ví dụ 2**
B=3;	**B=3;**
A=++B;	**A=B++;**
// A = 4, B = 4	*// A = 3, B = 4*

3.5.2. Toán tử so sánh logic

3.5.2.1. Các toán tử quan hệ (==, !=, >, <, >=, <=)

Để có thể so sánh hai biểu thức với nhau chúng ta có thể sử dụng các toán tử quan hệ. Theo chuẩn ANSI-C++ thì giá trị của thao tác quan hệ chỉ có thể là giá trị logic - chúng chỉ có thể có giá trị true hoặc false, tuỳ theo biểu thức kết quả là đúng hay sai.

Sau đây là các toán tử quan hệ có thể sử dụng trong C++

==	Bằng
!=	Khác
>	Lớn hơn
<	Nhỏ hơn
>=	Lớn hơn hoặc bằng
<=	Nhỏ hơn hoặc bằng

Ví dụ:

(7 == 5) sẽ trả giá trị **false**

(6 >= 6) sẽ trả giá trị **true**

Thay vì sử dụng các số, chúng ta có thể sử dụng bất cứ biểu thức nào. Cho a=2, b=3 và c=6.

(a*b >= c) sẽ trả giá trị **true.**

(b+4 < a*c) sẽ trả giá trị **false**

Cần chú ý rằng = (một dấu bằng) thì hoàn toàn khác với == (hai dấu bằng). Dấu đầu tiên là một toán tử gán (gán giá trị của biểu thức bên phải cho biến ở bên trái) và dấu còn lại (==) là một toán tử quan hệ nhằm so sánh xem hai biểu thức có bằng nhau hay không.

Chú ý: Trong nhiều trình dịch có trước chuẩn ANSI-C++ cũng như trong ngôn ngữ C, các toán tử quan hệ không trả về giá trị logic true hoặc false mà trả về giá trị int với 0 tương ứng với false còn giá trị khác 0 (thường là 1) thì tương ứng với true.

3.5.2.2. Các toán tử logic (!, &&, ||).

Toán tử ! tương đương với toán tử logic NOT, nó chỉ có một đối số ở phía bên phải và việc duy nhất mà nó làm là đổi ngược giá trị của đối số từ true sang false hoặc ngược lại. Ví dụ:

!(5 == 5) trả về **false** vì biểu thức bên phải (5 == 5) có giá trị **true.**

!(6 <= 4) trả về **true** vì (6 <= 4)có giá trị **false.**

!true trả về **false.**

!false trả về **true.**

Toán tử logic && và || được sử dụng khi tính toán hai biểu thức để lấy ra một kết quả duy nhất. Chúng tương ứng với các toán tử logic AND và OR. Kết quả của chúng phụ thuộc vào mối quan hệ của hai đối số:

Đối số thứ nhất **a**	Đối số thứ hai **b**	Kết quả **a && b**	Kết quả **a || b**

true	true	**true**	**true**
true	false	**false**	**true**
false	true	**false**	**true**
false	false	**false**	**false**

Ví dụ:

((5 == 5) && (3 > 6)) trả về **false** (*true* && *false*).

((5 == 5) || (3 > 6)) trả về **true** (*true* || *false*).

3.5.2.3. Toán tử điều kiện (?).

Toán tử điều kiện tính toán một biểu thức và trả về một giá trị khác tuỳ thuộc vào biểu thức đó là đúng hay sai. Cấu trúc của nó như sau:

condition ***?*** *result1* ***:*** *result2*

Nếu condition là true thì giá trị trả về sẽ là result1, nếu không giá trị trả về là result2.

7==5 ? 4 : 3 trả về **3** vì **7** không bằng **5**.

7==5+2 ? 4 : 3 trả về **4** vì **7** bằng **5+2**.

5>3 ? a : b trả về **a**, vì **5** lớn hơn **3**.

a>b ? a : b trả về giá trị lớn hơn, **a** hoặc **b**.

3.5.3. Các toán tử khác

3.5.3.1. Các toán tử thao tác bit (&, |, ^, ~, <<, >>).

Các toán tử thao tác bit thay đổi các bit biểu diễn một biến, có nghĩa là thay đổi biểu diễn nhị phân của chúng.

toán tử	asm	Mô tả
&	**AND**	Logical AND
|	**OR**	Logical OR
^	**XOR**	Logical exclusive OR
~	**NOT**	Đảo ngược bit
<<	**SHL**	Dịch bit sang trái
>>	**SHR**	Dịch bit sang phải

3.5.3.2. Các toán tử chuyển đổi kiểu

Các toán tử chuyển đổi kiểu cho phép ta chuyển đổi dữ liệu từ kiểu này sang kiểu khác. Có vài cách để làm việc này trong C++, cách cơ bản nhất được thừa kế từ ngôn ngữ C là đặt trước biểu thức cần chuyển đổi tên kiểu dữ liệu được bọc trong cặp ngoặc đơn (), ví dụ:

```
int i;
float f = 3.14;
i = (int) f;
```

Đoạn mã trên chuyển số thập phân 3.14 sang một số nguyên (3). Ở đây, toán tử chuyển đổi kiểu là (int). Một cách khác để làm điều này trong C++ là

```
i = int ( f );
```

Cả hai cách chuyển đổi kiểu đều hợp lệ trong C++. Thêm vào đó ANSI-C++ còn có những toán tử chuyển đổi kiểu mới đặc trưng cho lập trình hướng đối tượng.

3.5.3.3. sizeof()

Toán tử này có một tham số, đó có thể là một kiểu dữ liệu hay là một biến và trả về kích cỡ bằng byte của kiểu hay đối tượng đó.

a = sizeof (char);

a sẽ mang giá trị 1 vì kiểu char luôn có kích cỡ 1 byte trên mọi hệ thống. Giá trị trả về của sizeof là một hằng số vì vậy nó luôn luôn được tính trước khi chương trình thực hiện.

Trong C++ còn có một số các toán tử khác, như các toán tử liên quan đến con trỏ hay lập trình hướng đối tượng. Chúng sẽ được nói đến cụ thể trong các phần tương ứng.

3.5.4. Thứ tự ưu tiên của các toán tử

Khi viết các biểu thức phức tạp với nhiều toán hạng chúng ta có thể tự hỏi toán hạng nào được tính trước, toán hạng nào được tính sau.

Ví dụ như trong biểu thức sau:

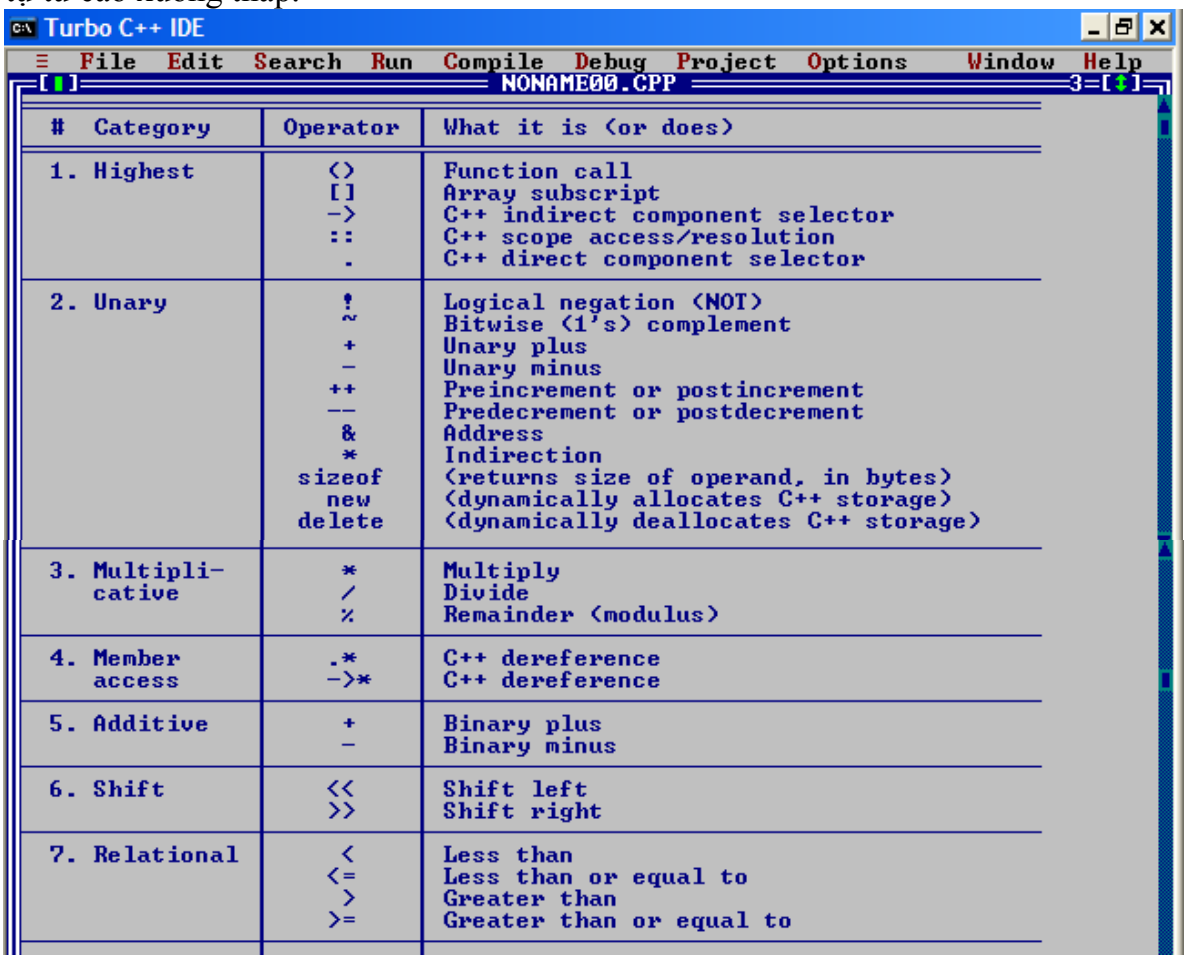
a = 5 + 7 % 2

có thể có hai cách hiểu sau:

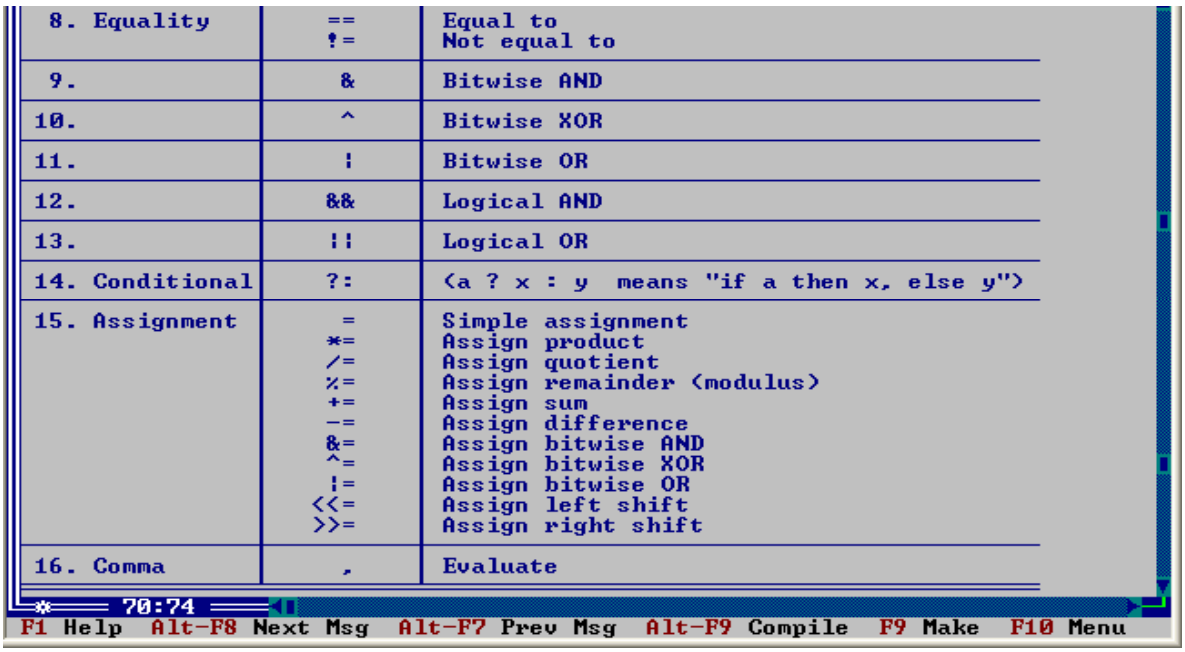
a = 5 + (7 % 2) với kết quả là **6**, hoặc

a = (5 + 7) % 2 với kết quả là **0.**

Câu trả lời đúng là biểu thức đầu tiên. Vì nguyên nhân nói trên, ngôn ngữ C++ đã thiết lập một thứ tự ưu tiên giữa các toán tử, không chỉ riêng các toán tử số học mà tất cả các toán tử có thể xuất hiện trong C++. Thứ tự ưu tiên của chúng được liệt kê trong bảng sau theo thứ tự từ cao xuống thấp.



# Category	Operator	What it is (or does)
1. Highest	() [] ->> :: .	Function call Array subscript C++ indirect component selector C++ scope access/resolution C++ direct component selector
2. Unary	! ~ + - ++ -- & * sizeof new delete	Logical negation (NOT) Bitwise (1's) complement Unary plus Unary minus Preincrement or postincrement Predecrement or postdecrement Address Indirection (returns size of operand, in bytes) (dynamically allocates C++ storage) (dynamically deallocates C++ storage)
3. Multiplicative	* / %	Multiply Divide Remainder (modulus)
4. Member access	.* ->*	C++ dereference C++ dereference
5. Additive	+ -	Binary plus Binary minus
6. Shift	<< >>	Shift left Shift right
7. Relational	< <= > >=	Less than Less than or equal to Greater than Greater than or equal to



Nếu ta muốn viết một biểu thức phức tạp mà lại không chắc lắm về thứ tự ưu tiên của các toán tử thì nên sử dụng các ngoặc đơn. Ta nên thực hiện điều này vì nó sẽ giúp chương trình dễ đọc hơn.

3.6. Bài tập:

Bài 3.1. Viết chương trình tính giá trị điện trở tương đương của mạch có: (R1//R2) nt (R3//R4) nt (R5//R6)?

Bài 3.2.. Cho 2 số nguyên dương a, b. Tìm USCLN của a và b?

Bài 3.3. Cho 2 số nguyên dương a, b. Tìm BSCNN của a và b?

Chương 4. CÁC VẤN ĐỀ VỀ NHẬP, XUẤT

4.1. Hàm scanf

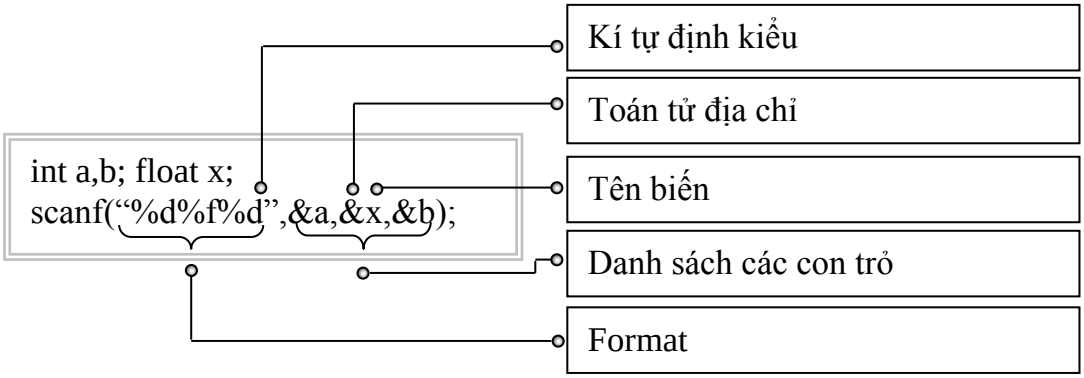
• ***Chức năng:***

Hàm scanf cho phép chúng ta nhập dữ liệu từ bàn phím theo khuôn dạng xác định. Đây là một trong những hàm nhập dữ liệu phổ biến nhất của C, nó cho phép nhập nhiều loại dữ liệu (có các kiểu khác nhau).

• ***Cú pháp:***

int scanf(const char * format, ds_các_con_trỏ);

@ ***Minh họa:***



Hàm scanf cho phép chúng ta nhập dữ liệu từ bàn phím theo khuôn dạng được xác định bởi xâu kí tự **format**, dữ liệu nhập vào sẽ lưu vào các biến hoặc vùng nhớ có địa chỉ tương ứng là các con trỏ trong **ds_các_con_trỏ** (có thể có nhiều con trỏ, mỗi con trỏ cách nhau bởi dấu phẩy).

Trong cú pháp trên **format** là một xâu quy định quy cách dữ liệu cần nhập, gồm nhiều đặc tả dữ liệu tương ứng với các kiểu của con trỏ trong phần **ds_các_con_trỏ**, có bao nhiêu con trỏ thì cần đúng bấy nhiêu đặc tả, đặc tả thứ nhất quy định khuôn dạng dữ liệu cho con trỏ thứ nhất, đặc tả thứ 2 quy định khuôn dạng dữ liệu cho con trỏ thứ 2,...

Mỗi đặc tả bắt đầu bằng dấu **%** có dạng sau (*các thành phần trong* [] *là tuỳ chọn*):

%[*][n]<ký_tự_định_kiểu>

Trong đó:

- **n** là một số nguyên dương quy định độ dài tối đa (tính theo số kí tự) được nhập cho thành phần tương ứng.

- **<ký_tự_định_kiểu>** là kí tự quy định kiểu dữ liệu cần nhập. Các kí tự định kiểu ở bảng sau:

Kí tự định kiểu	dữ liệu nhập	kiểu con trỏ của đối nhập liệu
d	integer	int *arg
D, ld	integer	long *arg
e, E	Float	float *arg
f	Float	float *arg
g, G	Float	float *arg
o	Octal	int *arg
O	Octal	long *arg
i	Decimal,octal, hex	int *arg
I	Decimal,octal, hex	long *arg
u	Unsigned int	unsigned int *arg
U	Unsigned int	unsigned long *arg
x	Hexadecimal	int *arg
X	Hexadecimal	int *arg
s	Character string	char arg[]
c	Character	char *arg

- * đây cũng là thành phần tuỳ chọn, nếu có thì tác dụng của nó là sẽ bỏ qua một thành phần dữ liệu được xác định bởi đặc tả này, như vậy sẽ không có con trỏ tương ứng với đặc tả này.

Ví dụ:

scanf("%d%*c%d",&a,&b);

trong dòng này chúng ta sẽ nhập 1 thành phần (gọi là 1 trường) số nguyên vào a, sau đó bỏ qua một thành phần là kí tự, và tiếp theo là một số nguyên vào b.

• ***Quy cách nhập dữ liệu:***

Khi chúng ta nhập dữ liệu từ bàn phím, kết thúc nhập bằng Enter (↵), thì tất cả những kí tự chúng ta gõ trên bàn phím đều được lưu trong vùng đệm nhập dữ liệu (gọi là dòng vào-stdin) -dòng vào kết thúc bởi (↵), dữ liệu trên dòng vào này sẽ được cắt thành từng trường tuần tự từ trái qua phải và gán vào các biến (hoặc vùng nhớ) xác định tương ứng bởi các con trỏ, các phần đã tách được sẽ bị loại khỏi dòng vào.

Trước khi tách giá trị một trường thì các khoảng trắng phía trước của trường nếu có sẽ bị loại bỏ. Nếu trong đặc tả không có thành phần [**n**] quy định độ dài tối đa một trường thì các trường được xác định bởi các ký tự dấu cách, tab, enter (gọi chung là khoảng trắng ký hiệu là ␣) hoặc khi gặp ký tự không phù hợp với đặc tả hiện tại.

Nếu trên dòng vào có nhiều hơn các thành phần yêu cầu của hàm nhập thì các thành phần chưa được nhận vẫn còn lưu trên dòng vào.

Ví dụ:

```
int a,b; float x;
scanf("%d%%d%f",&a,&b, &x);
```

với dòng vào là: ␣143 ␣ ␣ 535 ␣ 34 ↵

thì:

- khoảng trắng đầu tiên bị loại bỏ, 143 là trường thứ nhất được gán vào a,
- hai khoảng trắng bị loại bỏ, 535 là trường thứ hai được gán vào b,
- một khoảng trắng bị loại bỏ, 34 được gán vào x.

Nếu trong đặc tả có thành phần xác định độ rộng tối đa [**n**] thì một trường sẽ kết thúc khi gặp khoảng trống hoặc kí tự không phù hợp hoặc đã đủ độ dài **n.**

Ví dụ

```
int a,b; float x;
scanf("%d%2d%3f",&a,&b, &x);
```

với dòng vào là:

␣143 ␣ ␣ 537 ␣ 34 ↵

thì :

- khoảng trắng đầu tiên bị loại bỏ, 143 là trường thứ nhất được gán vào a,
- hai khoảng trắng bị loại bỏ, 53 là trường thứ hai được gán vào b,
- một khoảng trắng bị loại bỏ, 7 được gán vào x (còn lại 34↵ trong dòng vào)

• ***Lưu ý:***

- *Số các đặc tả phải tương ứng với số con trỏ trong danh sách con trỏ*
- *Ký tự định kiểu trong đặc tả phải phù hợp với kiểu của con trỏ cần nhập liệu.*
- *Dữ liệu nhập từ bàn phím phải phù hợp với các đặc tả.*
- *Hàm scanf trả về số nguyên là số trường được nhập dữ liệu*

4.2. Hàm printf

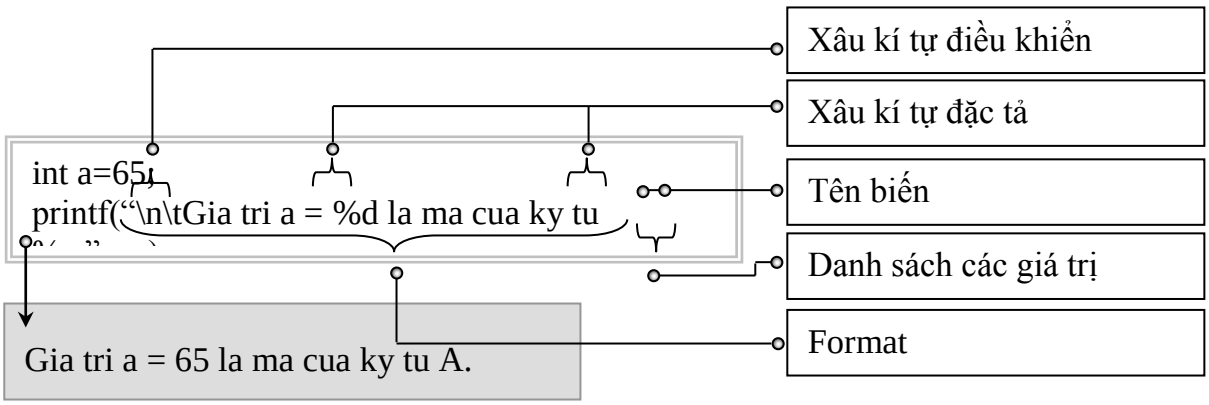
• ***Chức năng:***

Hàm printf cho phép chúng ta in dữ liệu từ biến (vùng nhớ) ra màn hình theo khuôn dạng được xác định. Đây là hàm in dữ liệu ra màn hình rất đa dạng của ngôn ngữ C. Cũng như hàm scanf, hàm printf cũng yêu cầu chúng ta phải cung cấp các giá trị và định dạng của dữ liệu cần in.

• ***Cú pháp:***

int printf (const char * format [, <danh_sách_các_giá trị>]);

@ Minh họa:



Trong cú pháp trên, **<danh_sách_các_giá trị>** là phần tuỳ chọn, nếu có thì đó là các giá trị cần in, các giá trị (có thể là biến, hằng, lời gọi hàm, hay biểu thức nói chung) cách nhau bởi dấu phẩy(,).

Lưu ý: số giá trị trong ***<danh_sách_các_giá trị>*** *có thể nhiều hơn số các đặc tả, khi đó các giá trị cuối (không có đặc tả tương ứng) sẽ bị bỏ qua.*

format là xâu ký tự điều khiển, nhiệm vụ chính của nó là điều khiển khuôn dạng thông tin được in ra màn hình.

Trong format gồm ba loại: các kí tự điều khiển, các đặc tả, các kí tự thường.

• *Các ký tự điều khiển*

Đây là các kí tự đặc biệt, bắt đầu bằng kí tự \ tiếp theo là 1 kí tự dùng để điều khiển: chuyển con trỏ màn hình, vị trí in dữ liệu,..

- \n : chuyển con trỏ màn hình xuống dòng mới
- \t : dấu tab
- \b : (backspace) lùi một kí tự (xoá kí tự trước vị trí con trỏ hiện tại)

• *Các ký tự thường*

Là những kí tự không thuộc loại điều khiển và đặc tả, các kí tự này được in ra màn hình đúng như nó xuất hiện trong format. Ngoài ra còn có một vài kí tự đặc biệt mà khi muốn in ra màn hình chúng ta phải đặt nó ngay sau kí tự \, đó là:

- \\ : để in chính dấu \
- \' : in dấu nháy đơn(')
- \" : in dấu nháy kép (")

• *Các đặc tả*

Trong format có thể có nhiều đặc tả, các đặc tả quy định khuôn dạng dữ liệu cần in ra, mỗi đặc tả có dạng như sau :

%[+][-][n[.m]]<ký_tự_định_kiểu>

Ý nghĩa các thành phần

Thành phần <ký_tự_định_kiểu> đây là kí tự quy định kiểu của dữ liệu cần in. Các kí tự định kiểu được cho trong bảng sau:

Kí tự định kiểu	Kiểu của giá trị cần in	Khuôn dạng in ra
d	int	số nguyên hệ 10
i	int	số nguyên hệ 10
o	int	số nguyên không dấu hệ 8
u	unsigned int	số nguyên không dấu hệ 10
x,X	int,unsigned	số nguyên không dấu hệ 16
f	float	số thực (dạng dấu phẩy tĩnh)
e, E	float	số thực (dấu phẩy tĩnh hoặc kí pháp khoa học)
c	char	kí tự
s	char *	xâu kí tự
p	con trỏ	in giá trị con trỏ dạng Segment:Offset hoặc Offset tuỳ mô hình bộ nhớ được sử dụng

Lưu ý: Có thể dùng kết hợp ld, lu, lx,.. để định kiểu dữ liệu in ra là số nguyên dài (long), số nguyên dài không dấu (unsigned long),..

Thành phần [n[.m]] : n, m là các số nguyên dương, n quy định độ rộng của thông tin (tính theo số ký tự) được in ra màn hình; m là số chữ số cho phần thập phân (chỉ dùng cho số thực), nếu có m thì số thực được làm tròn với m chữ số thập phân. Nếu độ rộng thực sự của giá trị cần in < độ rộng được dành cho nó (n) thì các dấu trống được thêm vào (bên trái hay bên phải tuỳ vào sự có mặt của thành phần [-] hay không).

*Lưu ý: có thể thay số n bằng kí tự *, khi đó thông tin sẽ được in ra theo đúng độ rộng thực sự của nó.*

ví dụ printf("%5.1f",1.37); sẽ in ra 1.4 và chiếm 5 vị trí trên màn hình, bên trái của số được điền 2 dấu cách.

Thành phần [-]: Xác định kiểu căn bên trái hay bên phải. Khi một giá trị được in ra trên màn hình, nếu độ rộng thực sự của nó nhỏ hơn độ rộng xác định bởi thành phần n, ngầm định chúng được căn bên phải (trong vùng n kí tự trên màn hình), nếu có dấu - thì dữ liệu được căn trái.

Thành phần [+]: In dấu (+) trước số dương.

Giá trị trả về của hàm printf là tổng độ dài thông tin được in (tính theo ký tự).

4.3. Hàm cin

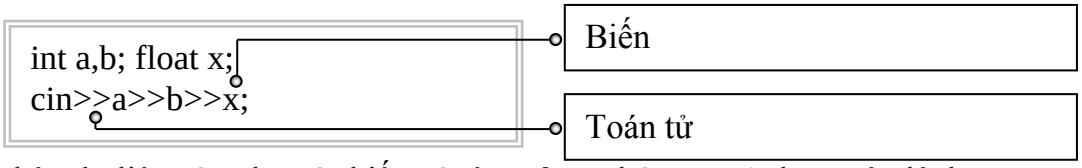
• ***Chức năng:***

Hàm cin là hàm nhập dữ liệu từ bàn phím phổ biến nhất của C++, nó cho phép nhập nhiều loại dữ liệu.

• ***Cú pháp:***

cin>>ten_bien1[>>ten_bien2];

*@ **Minh họa:***



Để nhập dữ liệu vào cho các biến có tên a, b, c chúng ta sử dụng câu lệnh:

cin >> a;

cin >> b ;

cin >> c ;

hoặc:

cin >> a >> b>> c;

a, b, c là các biến được sử dụng để lưu trữ các giá trị NSD nhập vào từ bàn phím. Khái niệm biến sẽ được mô tả cụ thể sau, ở đây a, b, c được hiểu là các tên gọi để chỉ 3 giá trị khác nhau. Hiển nhiên có thể nhập dữ liệu nhiều hơn 3 biến bằng cách tiếp tục viết tên biến vào bên phải sau dấu >> của câu lệnh.

Khi chạy chương trình nếu gặp các câu lệnh trên chương trình sẽ "tạm dừng" để chờ NSD nhập dữ liệu vào cho các biến. Sau khi NSD nhập xong dữ liệu, chương trình sẽ tiếp tục chạy từ câu lệnh tiếp theo sau của các câu lệnh trên.

Cách thức nhập dữ liệu của NSD phụ thuộc vào loại giá trị của biến cần nhập mà ta gọi là kiểu, ví dụ nhập một số có cách thức khác với nhập một chuỗi kí tự. Giả sử cần nhập độ dài hai cạnh của một hình chữ nhật, trong đó cạnh dài được quy ước bằng tên biến cd và chiều rộng được quy ước bởi tên biến cr. Câu lệnh nhập sẽ như sau:

cin >> cd >> cr ;

Khi máy dừng chờ nhập dữ liệu NSD sẽ gõ giá trị cụ thể của các chiều dài, rộng theo đúng thứ tự trong câu lệnh. Các giá trị này cần cách nhau bởi ít nhất một dấu trắng (ta quy ước gọi dấu trắng là một trong 3 loại dấu được nhập bởi các phím sau: phím spacebar (dấu cách), phím tab (dấu tab) hoặc phím Enter (dấu xuống dòng)). Các giá trị NSD nhập vào cũng được hiển thị trên màn hình để NSD dễ theo dõi.

Ví dụ nếu NSD nhập vào 23 11 ↵ thì chương trình sẽ gán giá trị 23 cho biến cd và 11 cho biến cr.

@ Chú ý:

Giả sử NSD nhập 2311 ↵ (không có dấu cách giữa 23 và 11) thì chương trình sẽ xem 2311 là một giá trị và gán cho cd. Máy sẽ tạm dừng chờ NSD nhập tiếp giá trị cho biến cr.

Toán tử nhập >> chủ yếu làm việc với dữ liệu kiểu số. Để nhập kí tự hoặc xâu kí tự, C++ cung cấp các phương thức (hàm) sau:

+ cin.get(c): cho phép nhập một kí tự vào biến kí tự c,

+ cin.getline(s,n): cho phép nhập tối đa n-1 kí tự vào xâu s.

Các hàm trên khi thực hiện sẽ lấy các kí tự còn lại trong bộ nhớ đệm (của lần nhập trước) để gán cho c hoặc s. Do toán tử cin >> x sẽ để lại kí tự xuống dòng trong bộ đệm nên kí tự này sẽ làm trôi các lệnh sau đó như cin.get(c), cin.getline(s,n) (máy không dừng để nhập cho c hoặc s). Vì vậy trước khi sử dụng các phương thức cin.get(c) hoặc cin.getline(s,n) nên sử dụng phương thức cin.ignore(1) để lấy ra kí tự xuống dòng còn sót lại trong bộ đệm. Ví dụ đoạn lệnh sau cho phép nhập một số nguyên x (bằng toán tử >>) và một kí tự c (bằng phương thức cin.get(c)):

```
int x;
char c;
cin >> x; cin.ignore(1);
cin.get(c);
```

4.4. Hàm cout

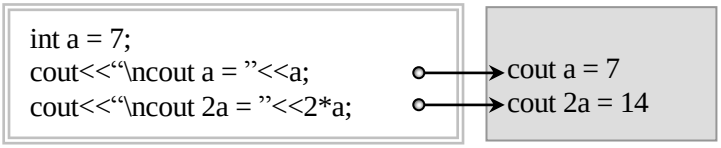
• ***Chức năng:***

Hàm cout là hàm in dữ liệu ra màn hình rất phổ biến của ngôn ngữ C++.

• ***Cú pháp:***

cout<<ten_bien1[<<ten_bien2];

@ Minh họa:



@ Định dạng thông tin cần in ra màn hình

Một số định dạng đơn giản được trình bày trước ở đây. Các định dạng chi tiết và phức tạp hơn sẽ được trình bày trong các phần sau của giáo trình. Để sử dụng các định dạng này cần khai báo file nguyên mẫu <iomanip.h> ở đầu chương trình bằng chỉ thị #include <iomanip.h>.

- endl: Tương đương với kí tự xuống dòng '\n'.
- setw(n): Bình thường các giá trị được in ra bởi lệnh cout << sẽ thẳng theo lề trái với độ rộng phụ thuộc vào độ rộng của giá trị đó. Phương thức này quy định độ rộng dành để in ra các giá trị là n cột màn hình. Nếu n lớn hơn độ dài thực của giá trị, giá trị sẽ in ra theo lề phải, để trống phần thừa (dấu cách) ở trước.
- setprecision(n): Chỉ định số chữ số của phần thập phân in ra là n. Số sẽ được làm tròn trước khi in ra.
- setiosflags(ios::showpoint): Phương thức setprecision chỉ có tác dụng trên một dòng in. Để cố định các giá trị đã đặt cho mọi dòng in (cho đến khi đặt lại giá trị mới) ta sử dụng phương thức setiosflags(ios::showpoint).

Ví dụ sau minh hoạ cách sử dụng các phương thức trên.

Ví dụ 3 :

```
#include <iostream.h>                    // để sử dụng cout <<
#include <iomanip.h>                     // để sử dụng các định dạng
#include <conio.h>                       // để sử dụng các hàm clrscr() và getch()
void main()
{
      clrscr();                          // xoá màn hình
      cout << "CHI TIÊU" << endl << "========" << endl ;
      cout << setiosflags(ios::showpoint) << setprecision(2) ;
      cout << "Sách vở" << setw(20) << 123.456 << endl;
      cout << "Thức ăn" << setw(20) << 2453.6 << endl;
      cout << "Quần áo lạnh" << setw(15) << 3200.0 << endl;
        getch();                         // tạm dừng (để xem kết quả)
      return ;                           // kết thúc thực hiện hàm main()
}
```

Chương trình này khi chạy sẽ in ra bảng sau:

```
CHI TIÊU
========
Sách vở                 123.46
Thức ăn                2453.60
Quần áo lạnh           3200.00
```

4.5. Các hàm nhập, xuất khác

4.5.1. Hàm getch, getche nhập 1 ký tự

• ***Cú pháp:***

int getch(); int getche();

• ***Chức năng:***

Hai hàm này thực hiện đợi người dùng nhập một ký tự từ bàn phím và trả về một số nguyên là mã của kí tự được bấm, ví dụ:

gõ phím a => *hàm sẽ trả về 97.*

Sự khác nhau giữa hai hàm là hàm getche() hiện kí tự được nhập lên màn hình, còn getch() thì không.

4.5.2. Hàm gets

• ***Chức năng:***

Hàm gets cho phép nhập một xâu kí tự từ bàn phím (bao gồm cả khoảng trống).

• ***Cú pháp:***

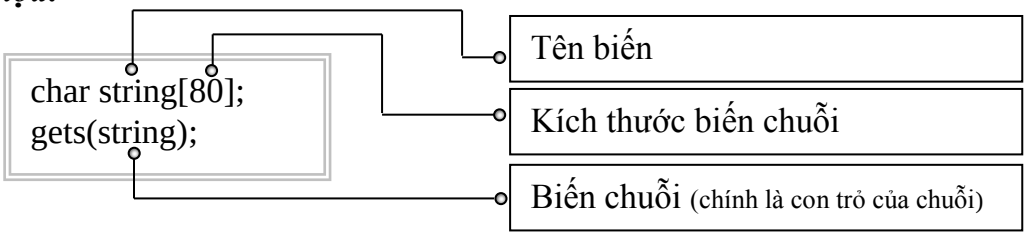
char * gets(char * s);

Khác với hàm scanf, hàm gets cho phép nhập một xâu kí tự chỉ kết thúc khi gặp enter (kí tự '\n') và lưu thông qua một biến con trỏ.

Xâu kí tự được ghi vào s (với s là mảng các kí tự hoặc con trỏ kí tự), dấu kết thúc xâu ('\0' - kí tự có mã 0) được tự động thêm vào cuối xâu kí tự. Hàm trả về địa chỉ của xâu được nhập.

Chú ý: hàm gets loại bỏ ký tự Enter('\n') trên dòng vào nhưng ký tự này không được đưa vào s mà tự động thêm ký tự kết thúc xâu ('\0') vào cuối của s.

@ Minh họa:



4.5.3. Hàm putch

• ***Chức năng*:**

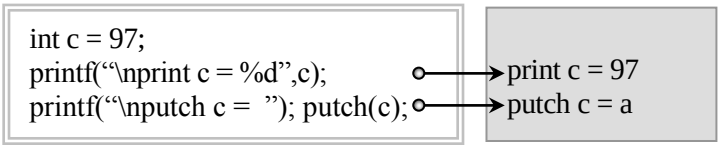
Hàm putch dùng để in một kí tự ra màn hình từ mã của nó theo bảng mã ASCII.

• ***Cú pháp*:**

int putch(int ch);

Hàm này in kí tự có mã là *ch* ra màn hình tại vị trí hiện tại của con trỏ, chuyển con trỏ sang phải 1 ký tự, hàm trả về số nguyên chính là mã kí tự in ra.

@ Minh họa:



4.6. Các thuật toán liên qua nhập xuất dữ liệu

4.6.1. Biểu diễn đồ thị trên máy tính

a) Ma trận kề, ma trận trọng số

Để lưu trữ đồ thị và thực hiện các thuật toán khác nhau, ta cần phải biểu diễn đồ thị trên máy tính, đồng thời sử dụng những cấu trúc dữ liệu thích hợp để mô tả đồ thị. Việc chọn cấu trúc dữ liệu nào để biểu diễn đồ thị có tác động rất lớn đến hiệu quả thuật toán. Vì vậy, lựa chọn cấu trúc dữ liệu thích hợp biểu diễn đồ thị sẽ phụ thuộc vào từng bài toán cụ thể.

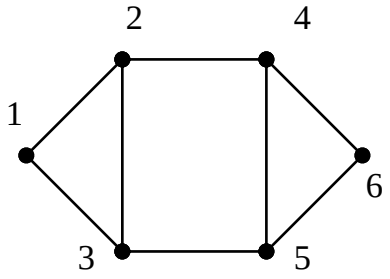
- Ma trận kề

Xét đơn đồ thị vô hướng G =<V, E>, với tập đỉnh V = {1, 2, . . ., n}, tập cạnh E = {e_1, e_2, ..., e_m}. Ta gọi ma trận kề của đồ thị G là ma trận có các phần tử hoặc bằng 0 hoặc bằng 1 theo quy định như sau:

$A = \{a_{ij}: a_{ij} = 1$ nếu $(i,j) \in E$, $a_{ij} = 0$ nếu $(i,j) \notin E$; $i, j = 1, 2, ..., n\}$.

Ví dụ 1. Biểu diễn đồ thị trong hình dưới đây bằng ma trận kề.

	1	2	3	4	5	6
1	0	1	1	0	0	0
2	1	0	1	1	0	0
3	1	1	0	0	1	0
4	0	1	0	0	1	1
5	0	0	1	1	0	1
6	0	0	0	1	1	0



Đồ thị vô hướng G

Ma trận kề có những tính chất sau:

a. Ma trận kề của đồ thị vô hướng là ma trận đối xứng A[i,j] = A[j, i]; i, j = 1, 2, . . . n. Ngược lại, mỗi ma trận (0,1) cấp *n* đẳng cấu với một đơn đồ thị vô hướng *n* đỉnh;

b. Tổng các phần tử theo dòng i (cột j) của ma trận kề chính bằng bậc đỉnh i (đỉnh j);

c. Nếu ký hiệu

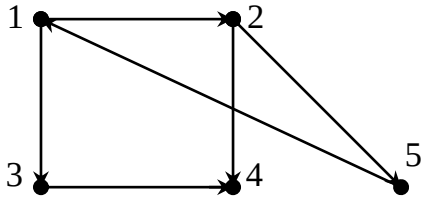
a_{ij}^p với i, j =1, 2, ..., n là các phần tử của ma trận

A^p = A.A. . . A (p lần) khi đó a_{ij}^p với *i, j =1, 2, ..., n* cho ta số đường đi khác nhau từ đỉnh *i* đến đỉnh *j* qua *p-1* đỉnh trung gian.

Ma trận kề của đồ thị có hướng cũng được định nghĩa hoàn toàn tương tự, chúng ta chỉ cần lưu ý tới hướng của cạnh. Ma trận kề của đồ thị có hướng là không đối xứng.

Ví dụ: Tìm ma trận kề của đồ thị có hướng trong hình sau.

	1	2	3	4	5
1	0	1	1	0	0
2	0	0	0	1	1
3	0	0	0	1	0
4	0	0	0	0	0
5	1	0	0	0	0



Đồ thị có hướng G

- Ma trận trọng số

Trong rất nhiều ứng dụng khác nhau của lý thuyết đồ thị, mỗi cạnh *e =(u,v)* của nó được gán bởi một số *c(e) = d(u,v)* gọi là trọng số của cạnh *e*. Đồ thị trong trường hợp như vậy gọi là đồ thị trọng số. Trong trường hợp đó, ma trận kề của đồ thị được thay bởi ma trận trọng số

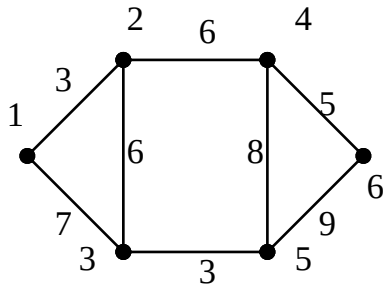
c= {c[i,j] với i, j= 1, 2, ..., n. c[i,j] = d(i,j) nếu (i, j) $\in$ E, c[i,j] = θ nếu (i, j) $\notin$E}.

Ưu điểm của phương pháp biểu diễn đồ thị bằng ma trận kề (hoặc ma trận trọng số) là ta dễ dàng trả lời được câu hỏi: Hai đỉnh u, v có kề nhau trên đồ thị hay không và chúng ta chỉ

mất đúng một phép so sánh. Nhược điểm lớn nhất của nó là bất kể đồ thị có bao nhiêu cạnh ta đều mất n^2 đơn vị bộ nhớ để lưu trữ đồ thị.

Ví dụ: Ma trận kề của đồ thị có trọng số trong hình sau.

	1	2	3	4	5	6
1	0	3	7	0	0	0
2	3	0	6	6	0	0
3	7	6	0	0	3	0
4	0	6	0	0	8	5
5	0	0	3	8	0	9
6	0	0	0	5	9	0



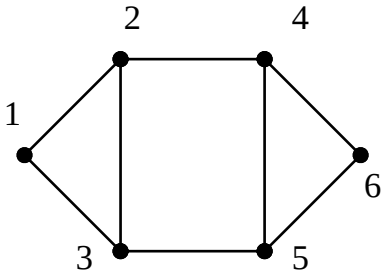
Đồ thị trọng số G

b) Danh sách cạnh (cung)

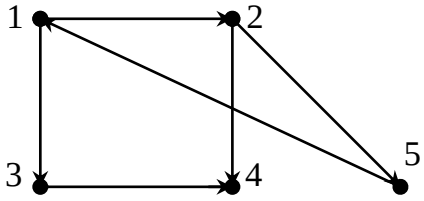
Trong trường hợp đồ thị thưa (đồ thị có số cạnh $m \leq 6n$), người ta thường biểu diễn đồ thị dưới dạng danh sách cạnh. Trong phép biểu diễn này, chúng ta sẽ lưu trữ danh sách tất cả các cạnh (cung) của đồ thị vô hướng (có hướng). Mỗi cạnh (cung) *e(x,y)* được tương ứng với hai biến *dau[e], cuoi[e]*. Như vậy, để lưu trữ đồ thị, ta cần *2m* đơn vị bộ nhớ.

Nhược điểm lớn nhất của phương pháp này là để nhận biết những cạnh nào kề với cạnh nào chúng ta cần *m* phép so sánh trong khi duyệt qua tất cả *m* cạnh (cung) của đồ thị. Nếu là đồ thị có trọng số, ta cần thêm *m* đơn vị bộ nhớ để lưu trữ trọng số của các cạnh.

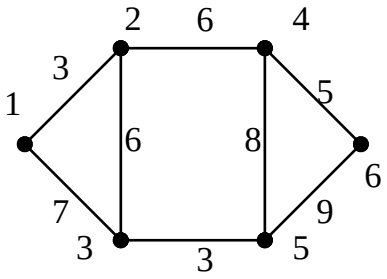
Ví dụ: Danh sách cạnh (cung) của đồ thị vô hướng, đồ thị có hướng, đồ thị trọng số:



dau	cuoi
1	2
1	3
2	3
2	4
3	5
4	5
4	6
5	6



dau	cuoi
1	2
1	3
2	4
2	5
3	4
5	1



dau	cuoi	trongso
1	2	3
1	3	7
2	3	6
2	4	6
3	5	3
4	5	8
4	6	5
5	6	9

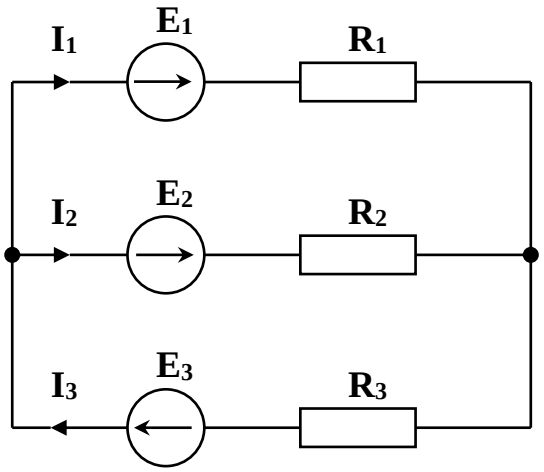
4.7. Bài tập

Bài 4.1. Viết chương trình giải hệ phương trình sau:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 = b_2 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 = b_3 \end{cases}$$

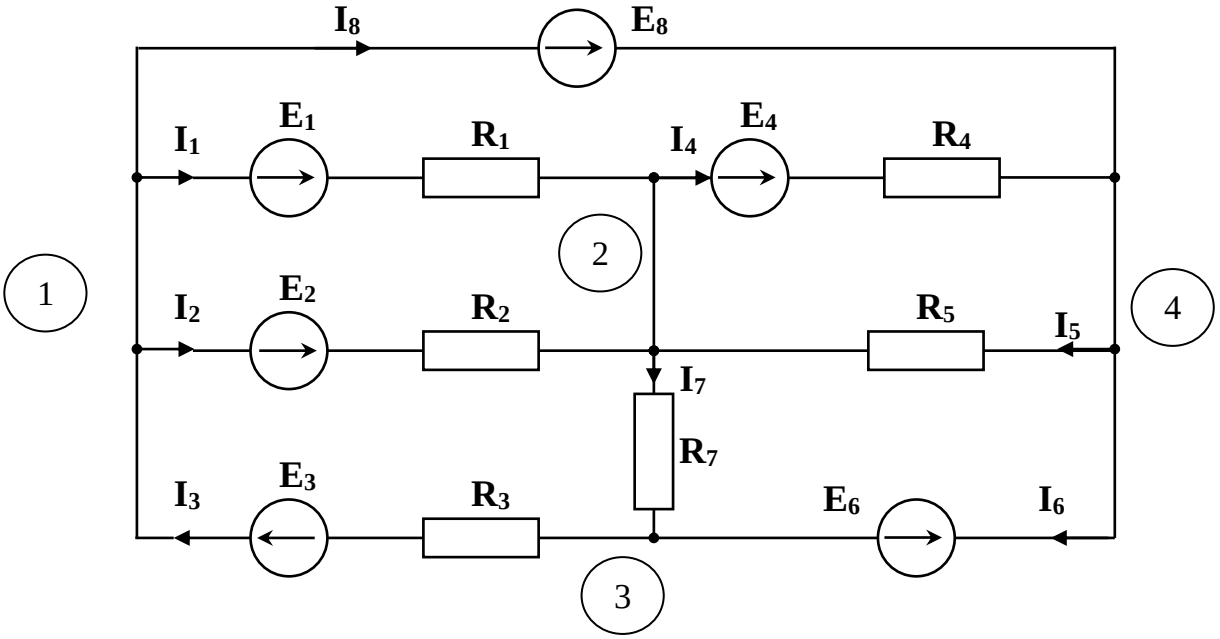
Biết: a_{ij}, b_i (với $i = \overline{1,3}$ và $j = \overline{1,3}$) được nhập từ bàn phím.

Bài 4.2. Viết chương trình tìm I_i (với $i = \overline{1,3}$) trên mạch điện sau?



Biết: R_i, E_i (với $i = \overline{1,3}$) được nhập từ bàn phím.

Bài 4.3. Viết chương trình nhập dữ liệu của mạch điện sau dưới dạng danh sách cạnh? Xuất kết quả đã nhập.



Chương 5. CÁC CẤU TRÚC LỆNH ĐIỀU KHIỂN

Một chương trình thường không chỉ bao gồm các lệnh tuần tự nối tiếp nhau. Trong quá trình chạy nó có thể rẽ nhánh hay lặp lại một đoạn mã nào đó. Để làm điều này chúng ta sử dụng các cấu trúc điều khiển.

Cùng với việc giới thiệu các cấu trúc điều khiển chúng ta cũng sẽ phải biết tới một khái niệm mới: **khối lệnh**, đó là một nhóm các lệnh được ngăn cách bởi dấu chấm phẩy (;) nhưng được gộp trong một khối giới hạn bởi một cặp ngoặc nhọn: **{** và **}.**

Hầu hết các cấu trúc điều khiển mà chúng ta sẽ xem xét trong chương này cho phép sử dụng một lệnh đơn hay một khối lệnh làm tham số, tuỳ thuộc vào chúng ta có đặt nó trong cặp ngoặc nhọn hay không.

5.1. Cấu trúc điều kiện: if - else

Cấu trúc này được dùng khi một lệnh hay một khối lệnh chỉ được thực hiện khi một điều kiện nào đó thoả mãn. Dạng của nó như sau:

if **(***điềukiện***)** *côngviệc*

trong đó *điềukiện* là true, công việc được thực hiện, nếu không *côngviệc* bị bỏ qua (không thực hiện) và chương trình tiếp tục thực hiện lệnh tiếp sau cấu trúc điều kiện.

Ví dụ, đoạn mã sau đây sẽ viết **x is 100** chỉ khi biến **x** chứa giá trị 100:

```
if (x == 100)    cout << "x is 100";
```

Nếu chúng ta muốn có hơn một lệnh được thực hiện trong trường hợp *điềukiện* là true chúng ta có thể chỉ định một khối lệnh bằng cách sử dụng một cặp ngoặc nhọn { }:

```
if (x == 100)
 {
   cout << "x is ";
   cout << x;
 }
```

Chúng ta cũng có thể chỉ định điều gì sẽ xảy ra nếu điều kiện không được thoả mãn bằng cách sử dụng từ khoá else. Nó được sử dụng cùng với if như sau:

if (*điềukiện***)** *côngviệc1* **else** *côngviệc2*

Ví dụ:

```
if (x == 100)
  cout << "x is 100";
else
```

```
  cout << "x is not 100";
```

Cấu trúc if + else có thể được móc nối để kiểm tra nhiều giá trị.

Ví dụ sau đây sẽ kiểm tra xem giá trị chứa trong biến x là dương, âm hay bằng không.

```
if (x > 0)
  cout << "x duong";
else if (x < 0)
  cout << "x am";
else
  cout << "x is 0";
```

5.2. Cấu trúc lựa chọn: switch

Mục đích của cấu trúc *switch* là kiểm tra một vài giá trị hằng (*const*) cho một *biểuthức*, tương tự với những gì chúng ta làm khi liên kết một số lệnh *if* và *else if* với nhau. Dạng thức của nó như sau:

switch (*biểuthức***) {**
case *hằng1***:**
công việc1
break;
case *hằng2***:**
côngviệc2
break;
.
.
.
default:
default côngviệcn
}

Nó hoạt động theo cách sau: **switch** tính *biểuthức* và kiểm tra xem nó có bằng *hằng1* hay không, nếu đúng thì nó thực hiện *côngviệc1* cho đến khi tìm thấy từ khoá **break**, sau đó nhảy đến phần cuối của cấu trúc lựa chọn *switch*.

Còn nếu không, switch sẽ kiểm tra xem *biểuthức* có bằng *hằng2* hay không. Nếu đúng nó sẽ thực hiện *côngviệc2* cho đến khi tìm thấy từ khoá **break.**

Cuối cùng, nếu giá trị biểu thức không bằng bất kì hằng nào được chỉ định ở trên, chương trình sẽ thực hiện các lệnh trong phần **default:** nếu nó tồn tại vì phần này không bắt buộc phải có.

Hai đoạn mã sau là tương đương:

ví dụ switch	***if-else* tương đương**

switch (x) {
case 1:
cout << "x is 1";
break;
case 2:
cout << "x is 2";
break;
default:
cout << "value of x unknown";
}
	if (x == 1) {
	cout << "x is 1";
	}
	else if (x == 2) {
	cout << "x is 2";
	}
	else {
	cout << "value of x unknown";
	}

Cấu trúc của lệnh ***switch*** hơi đặc biệt. Chú ý sự tồn tại của lệnh **break** ở cuối mỗi khối lệnh. Điều này là cần thiết vì nếu không thì sau khi thực hiện *côngviệc1* chương trình sẽ không nhảy đến cuối của lệnh switch mà sẽ thực hiện các khối lệnh tiếp theo cho đến khi nó tìm thấy lệnh **break** đầu tiên. Điều này khiến cho việc đặt cặp ngoặc nhọn **{ }** trong mỗi trường hợp là không cần thiết và có thể được dùng khi ta muốn thực hiện một khối lệnh cho nhiều trường hợp khác nhau, ví dụ:

```
switch (x) {
 case 1:
 case 2:
 case 3:
  cout << "x is 1, 2 or 3";
  break;
 default:
  cout << "x is not 1, 2 nor 3";
 }
```

Chú ý rằng lệnh **switch** chỉ có thể được dùng để so sánh một *biểu thức* với các *hằng*. Vì vậy chúng ta không thể đặt các biến (**case (n*2):**) hay các khoảng (**case (1..3):**) vì chúng không phải là các hằng hợp lệ.

Nếu ta cần kiểm tra các khoảng hay nhiều giá trị không phải là hằng số hãy kết hợp các lệnh **if** và **else if**

Ví dụ: Chương trình nhập xuất số ngày trong tháng.

```
#include <iostream.h>
using namespace std;
int main()
{
      int month, day;
      cout << "Month: " << endl;
      cin >> month;
```

```
        switch (month)
        {
           case 1:
           case 3:
           case 5:
           case 7:
           case 8:
           case 10:
           case 12:
              day = 30;
              break;
           case 4:
           case 6:
           case 9:
           case 11:
              day = 31;
              break;
           default:
              day = 28;
        }
        cout<<"\nThang "<< month<<" co "<< day<<" ngay.";
        return 0;
}
```

5.3. Các cấu trúc lặp

Mục đích của các vòng lặp là lặp lại một thao tác với một số lần nhất định hoặc trong khi một điều kiện nào đó còn thoả mãn.

5.3.1. Vòng lặp while

Dạng của nó như sau:
while (*điềukiện***)** *côngviệc*

Chức năng của nó đơn giản chỉ là lặp lại côngviệc khi điều kiện điềukiện còn thoả mãn.
Ví dụ, chúng ta sẽ viết một chương trình đếm ngược sử dụng vào lặp while:

```
// custom countdown using while
#include <iostream.h>
int main ()
{
  int n;
  cout << "Enter the starting number > ";
  cin >> n;
  while (n>0) {
    cout << n << ", ";
    --n;
  }
```

Enter the starting number >
8
8, 7, 6, 5, 4, 3, 2, 1, ZERO!

```
  cout << "ZERO!";
  return 0;
}
```

Khi chương trình chạy người sử dụng được yêu cầu nhập vào một số để đếm ngược. Sau đó, khi vòng lặp *while* bắt đầu nếu số mà người dùng nhập vào thoả mãn điều kiện n>0 khối lệnh sẽ được thực hiện một số lần không xác định chừng nào điều kiện (n>0) còn được thoả mãn.

Chúng ta cần phải nhớ rằng vòng lặp phải kết thúc ở một điểm nào đó, vì vậy bên trong vòng lặp chúng ta phải cung cấp một phương thức nào đó để buộc điềukiện trở thành sai nếu không thì nó sẽ lặp lại mãi mãi. Trong ví dụ trên vòng lặp phải có lệnh --n; để làm cho *condition* trở thành sai sau một số lần lặp.

5.3.2. Vòng lặp do-while

Dạng thức:

do *côngviệc* **while (***điềukiện***);**

Chức năng của nó là hoàn toàn giống vòng lặp *while* chỉ trừ có một điều là điều kiện điều khiển vòng lặp được tính toán *sau khi* *côngviệc* được thực hiện, vì vậy *côngviệc* sẽ được thực hiện ít nhất một lần ngay cả khi *điềukiện* không bao giờ được thoả mãn. Ví dụ, chương trình dưới đây sẽ viết ra bất kì số nào mà ta nhập vào cho đến khi ta nhập số 0.

```
#include <iostream.h>
int main ()
{
  unsigned long n;
  do {
    cout << "Enter number (0 to end): ";
    cin >> n;
    cout << "You entered: " << n << "\n";
  } while (n != 0);
  return 0;
}
```

Enter number (0 to end):
12345
You entered: 12345
Enter number (0 to end):
160277
You entered: 160277
Enter number (0 to end): 0
You entered: 0

Vòng lặp *do-while* thường được dùng khi điều kiện để kết thúc vòng lặp nằm trong vòng lặp, như trong ví dụ trên, số mà người dùng nhập vào là điều kiện kiểm tra để kết thúc vòng lặp. Nếu ta không nhập số 0 trong ví dụ trên thì vòng lặp sẽ không bao giờ chấm dứt.

5.3.3. Vòng lặp for

Dạng thức:

for (*khởitạo***;** *điềukiện***;** *tăng/giảm***)** *côngviệc*;

Chức năng chính của nó là lặp lại *côngviệc* chừng nào *điềukiện* còn mang giá trị đúng, như trong vòng lặp *while*. Nhưng thêm vào đó, **for** cung cấp chỗ dành cho lệnh khởi tạo và lệnh tăng. Vì vậy vòng lặp này được thiết kế đặc biệt lặp lại một hành động với một số lần xác định.

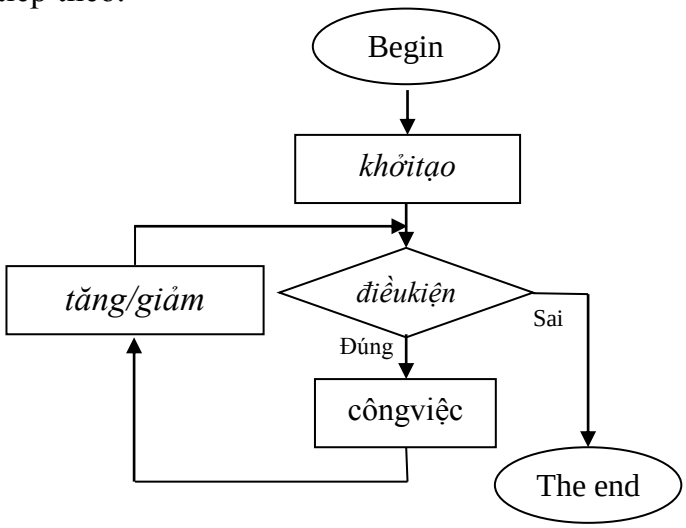
Cách thức hoạt động của nó như sau:

1, *khởitạo* được thực hiện. Nói chung nó đặt một giá trị ban đầu cho biến điều khiển. Lệnh này được thực hiện *chỉ một lần*.

2, *điềukiện* được kiểm tra, nếu nó là đúng vòng lặp tiếp tục còn nếu không vòng lặp kết thúc và *côngviệc* được bỏ qua.

3, *côngviệc* được thực hiện. Nó có thể là một lệnh đơn hoặc là một khối lệnh được bao trong một cặp ngoặc nhọn { }.

4, Cuối cùng, *tăng/giảm* được thực hiện để tăng hoặc giảm biến điều khiển và vòng lặp quay trở lại bước tiếp theo.



Hình 5. 1. Lưu đồ cấu trúc lặp for.

Sau đây là một ví dụ đếm ngược sử dụng vòng *for*.

```
// countdown using a for loop
#include <iostream.h>
int main ()
{
  for (int n=10; n>0; n--) {
    cout << n << ", ";
  }
  cout << "FIRE!";
  return 0;
}
```

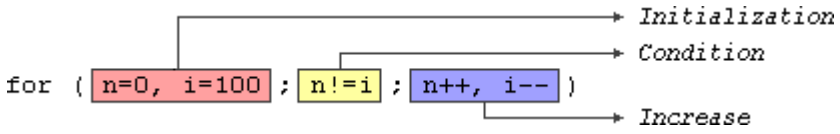
10, 9, 8, 7, 6, 5, 4, 3, 2, 1, FIRE!

Phần khởi tạo và lệnh tăng không bắt buộc phải có. Chúng có thể được bỏ qua nhưng vẫn phải có dấu chấm phẩy ngăn cách giữa các phần. Vì vậy, chúng ta có thể viết **for (;n<10;)** hoặc **for (;n<10;n++)**.

Bằng cách sử dụng dấu phẩy, chúng ta có thể dùng nhiều lệnh trong bất kì trường nào trong vòng *for*, như là trong phần khởi tạo. Ví dụ chúng ta có thể khởi tạo một lúc nhiều biến trong vòng lặp:

```
for ( n=0, i=100 ; n!=i ; n++, i-- )
{
  // cái gì ở đây cũng được...
}
```

Vòng lặp này sẽ thực hiện 50 lần nếu như n và i không bị thay đổi trong thân vòng lặp:



5.4. Các lệnh rẽ nhánh và lệnh nhảy

5.4.1. Lệnh break

Sử dụng *break* chúng ta có thể thoát khỏi vòng lặp ngay cả khi điều kiện để nó kết thúc chưa được thoả mãn. Lệnh này có thể được dùng để kết thúc một vòng lặp không xác định hay buộc nó phải kết thúc giữa chừng thay vì kết thúc một cách bình thường. Ví dụ, chúng ta sẽ dừng việc đếm ngược trước khi nó kết thúc:

```
// break loop example
#include <iostream.h>
int main ()
{
  int n;
  for (n=10; n>0; n--) {
    cout << n << ", ";
    if (n==3)
    {
      cout << "Bo qua!";
      break;
    }
  }
  return 0;
}
```

10, 9, 8, 7, 6, 5, 4, Bo qua!

5.4.2. Lệnh continue

Lệnh *continue* làm cho chương trình bỏ qua phần còn lại của vòng lặp và nhảy sang lần lặp tiếp theo. Ví dụ chúng ta sẽ bỏ qua số 5 trong phần đếm ngược:

```
// break loop example
#include <iostream.h>
int main ()
{
  for (int n=10; n>0; n--) {
    if (n==5) continue;
    cout << n << ", ";
  }
  cout << "ZERO!";
  return 0;
}
```

10, 9, 8, 7, 6, 4, 3, 2, 1, ZERO!

5.4.3. Lệnh goto

Lệnh này cho phép nhảy vô điều kiện tới bất kì điểm nào trong chương trình. Nói chung ta nên tránh dùng nó trong chương trình C++. Tuy nhiên chúng ta vẫn xét một ví dụ dùng lệnh **goto** để đếm ngược:

```
// goto loop example
#include <iostream.h>
int main ()
{
  int n=10;
  loop: ;
  cout << n << ", ";
  n--;
  if (n>0) goto loop;
  cout << "ZERO!";
  return 0;
}
```

10, 9, 8, 7, 6, 5, 4, 3, 2, 1, ZERO!

5.4.4. Hàm exit()

Mục đích của hàm exit() là kết thúc chương trình và trả về một mã xác định. Dạng thức của nó như sau

void exit (int *exit code***);**

exit code được dùng bởi một số hệ điều hành hoặc có thể được dùng bởi các chương trình gọi. Theo quy ước, mã trả về 0 có nghĩa là chương trình kết thúc bình thường còn các giá trị khác 0 có nghĩa là có lỗi.

5.6. Các thuật toán liên qua đến các cấu trúc lệnh điều khiển

5.6.1. Các thuật toán sắp xếp

Sắp xếp là quá trình bố trí lại các phần tử trong một tập hợp theo một trình tự nào đó nhằm mục đích giúp quản lý và tìm kiếm các phần tử dễ dàng và nhanh chóng hơn.

Các phương pháp sắp xếp thông dụng:

- Phương pháp Đổi chỗ trực tiếp (Interchange sort)
- Phương pháp Nổi bọt (Bubble sort)
- Phương pháp Chèn trực tiếp (Insertion sort)
- Phương pháp Chọn trực tiếp (Selection sort)

a) Sắp xếp chèn (Insertion Sort)

Ý tưởng: Insertion Sort lấy ý tưởng từ việc chơi bài, dựa theo cách người chơi "chèn" thêm một quân bài mới vào bộ bài đã được sắp xếp trên tay.

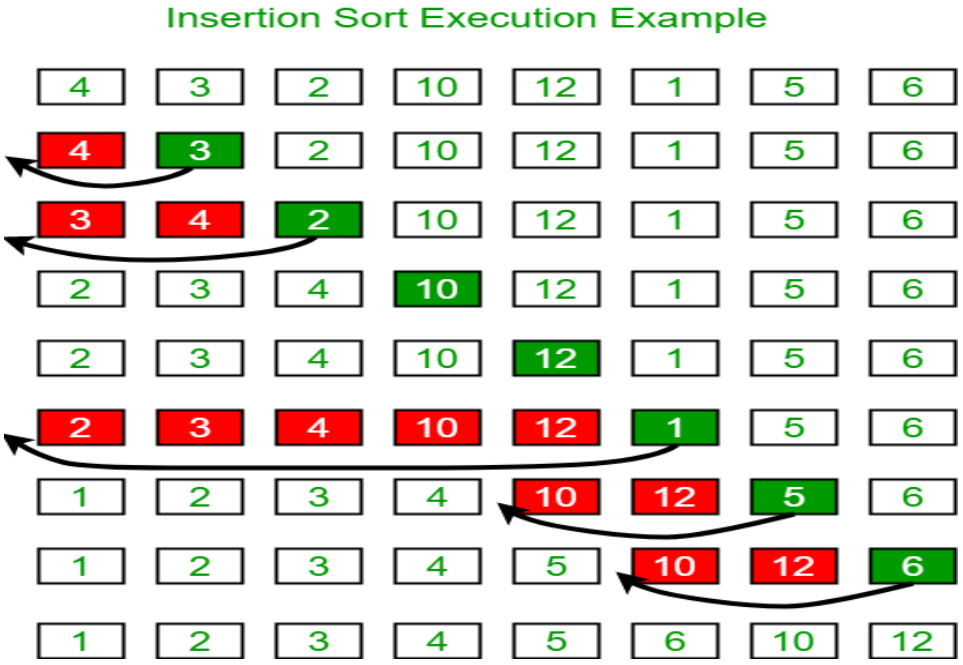
Thuật toán: Tại bước k = 1, 2, ..., n đưa phần tử thứ k trong mảng đã cho vào đúng vị trí trong dãy gồm k phần tử đầu tiên.

Kết quả là sau bước thứ k, sẽ có k phần tử đầu tiên được sắp xếp theo thứ tự.

Thuật toán dùng hàm con	Thuật toán không dùng hàm con
void insertionSort(int a[], int array_size)	// Begin insertionSort

{
int i, j, last;
for (i=1; i < array_size; i++)
{
last = a[i];
j = i;
while ((j > 0) && (a[j-1] > last))
{
a[j] = a[j-1];
j = j - 1;
}
a[j] = last;
} // end for
} // end of isort
	//Đã khai báo int a[ ], int array_size;
	int i, j, last;
	for (i=1; i < array_size; i++)
	{
	last = a[i];
	j = i;
	while ((j > 0) && (a[j-1] > last))
	{
	a[j] = a[j-1];
	j = j - 1;
	}
	a[j] = last;
	} // end for
	// The end of insertionSort

Ví dụ:



b) Sắp xếp lựa chọn (Selection Sort)

Ý tưởng của Selection sort là tìm từng phần tử cho mỗi vị trí của mảng hoán vị A' cần tìm.

Thuật toán:

Tìm phần tử nhỏ nhất đưa vào vị trí 1

Tìm phần tử nhỏ tiếp theo đưa vào vị trí 2

Tìm phần tử nhỏ tiếp theo đưa vào vị trí 3…

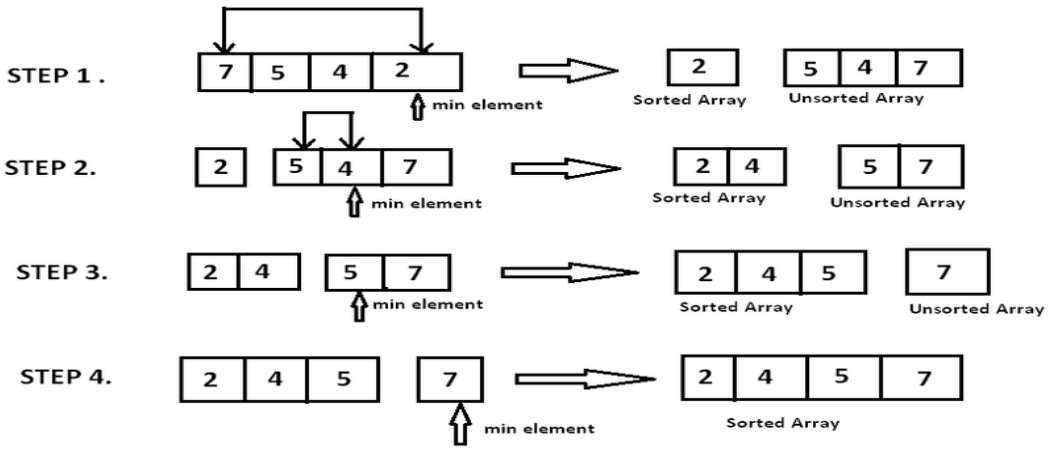
Thuật toán dùng hàm con	Thuật toán không dùng hàm con

void Swap(int &a, int &b)
{
int temp = a;
	//Begin SelectionSort
	//Đã khai báo int a[ ], int n;
	int i, j, min, temp;

```
   a = b;
   b = temp;
}//-------the end of Swap()---------

void SelectionSort(int a[], int n)
{
   int i, j, min, temp;
   for (i = 0; i < n-1; i++)
  {
      min = i;
      for (j = i+1; j < n; j++)
         if (a[j] < a[min]) min = j;
      Swap(a[i], a[min]);
   }
}
```

```
   for (i = 0; i < n-1; i++)
  {
      min = i;
      for (j = i+1; j < n; j++)
         if (a[j] < a[min]) min = j;
      if(i !=min)// nếu min ≠ i thì đổi chỗ
        {  temp =a[i];
           a[i]= a[min];
           a[min]=temp;
        }
   }
//The end of SelectionSort
```

Ví dụ :



c) <u>Sắp xếp nổi bọt (Bubble Sort)</u>

Ý tưởng: Bubble Sort, như cái tên của nó, là thuật toán đẩy phần tử lớn nhất xuống cuối dãy, đồng thời những phần tử có giá trị nhỏ hơn sẽ dịch chuyển dần về đầu dãy. Tựa như sự nổi bọt vậy, những phần tử nhẹ hơn sẽ nổi lên trên và ngược lại, những phần tử lớn hơn sẽ chìm xuống dưới.

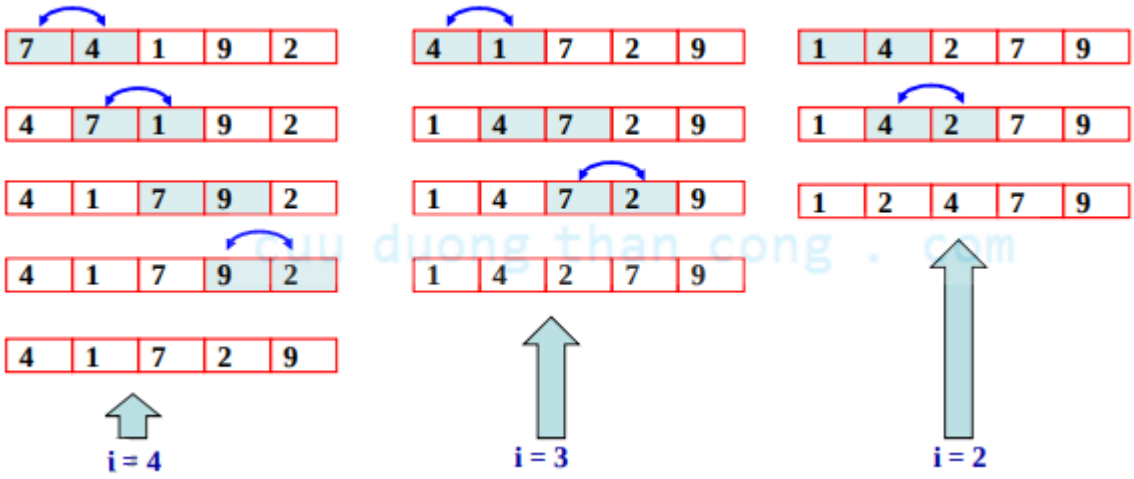
Thuật toán: Duyệt mảng từ phần tử đầu tiên. Ta sẽ so sánh mỗi phần tử với phần tử liền trước nó, nếu chúng đứng sai vị trí, ta sẽ đổi chỗ chúng cho nhau.

Thuật toán dùng hàm con	Thuật toán không dùng hàm con

void BubbleSort(int a[], int n)
{
  int i, j;
  for (i = (n-1); i >= 0; i--)
  {
  for (j = 1; j <= i; j++)
  {
  if (a[j-1] > a[j])
  Swap(a[j-1],a[j]);
	//Begin BubbleSort
	//Đã khai báo int a[ ], int n;
	  int i, j;
	  for (i = (n-1); i >= 0; i--)
	  {
	  for (j = 1; j <= i; j++)
	  {
	  if (a[j-1] > a[j])
	  {temp = a[j-1];

}
}
}
	a[j-1]= a[j];
	a[j]=temp;
	}
	}
	}
	//The end of BubbleSort

Ví dụ :



d) Đổi chỗ trực tiếp (Interchange Sort)

Khái niệm nghịch thế: Xét một mảng các số a[0], a[1], … a[n-1]. Nếu có i<j và a[i] > a[j], thì ta gọi đó là một nghịch thế.

Mảng chưa sắp xếp sẽ có nghịch thế. Ngược lại, mảng có thứ tự sẽ không chứa nghịch thế.

a[0] <= a[1] <=… <=[n -1]

Nhận xét:

Để sắp xếp một dãy số, ta có thể xét các nghịch thế có trong dãy và làm triệt tiêu dần chúng đi.

Thuật toán: xuất phát từ đầu dãy, tìm tất cả nghịch thế chứa phần tử này, triệt tiêu chúng bằng cách đổi chỗ phần tử này với phần tử tương ứng trong cặp nghịch thế. Lặp lại xử lý trên với các phần tử tiếp theo trong dãy.

Thuật toán dùng hàm con	Thuật toán không dùng hàm con

void InterchangeSort(int a[], int n)
{
for (int i = 0; i < n - 1; i++)
for (int j = i + 1; j < n; j++)
if(a[i] > a[j]) //nếu đúng thì đổi chỗ
Swap(a[i], a[j]);
}
	//Bắt đầu InterchangeSort
	//với int a[], int n ; đã khai báo và có dữ liệu.
	int temp ;
	for (int i = 0; i < n - 1; i++)
	for (int j = i + 1; j < n; j++)
	if(a[i] > a[j]) //nếu đúng thì đổi chỗ
	{ temp =a[i];
	a[i]= a[j];
	a[j]=temp;
	}

	//Kết thúc InterchangeSort

Ví dụ:

	24	45	23	13	43	-1
Step 1	24	45	23	13	43	-1
	24	45	23	13	43	-1
	23	45	24	13	43	-1
	13	45	24	23	43	-1
	13	45	24	23	43	-1
	-1	45	24	23	43	13
Step 2	-1	45	24	23	43	13
	-1	24	45	23	43	13
	-1	23	45	24	43	13
	-1	23	45	24	43	13
	-1	13	45	24	43	23
Step 3	-1	13	45	24	43	23
	-1	13	24	45	43	23
	-1	13	24	45	43	23
	-1	13	23	45	43	24

Step 4	-1	13	23	45	43	24
	-1	13	23	43	45	24
	-1	13	23	24	45	43
Step 5	-1	13	23	24	45	43
	-1	13	23	24	43	45

5.6.2. Thuật toán sinh kế tiếp

Áp dụng thuật toán này để liệt kê các tập con của một tập gồm n phần tử.

- Mã hóa tập biến: Tập biến gồm n biến ký tự theo thứ tự các phần tử → mảng n ký tự.
- Miền trị của mỗi biến {'0', '1'}. '0' mô tả cho tình huống phần tử này không có trong tập con, '1': mô tả cho tình huống phần tử này có mặt trong tập con.
- Với tập cha là 4 (n = 4) phần tử X={ a, b, c, d }, có thể dùng mảng "0111" mô tả cho tập con {b,c,d}.

→ Mỗi tập con được biểu diễn là một chuỗi (xâu) nhị phân.

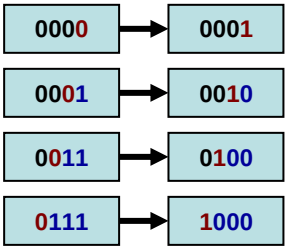
- Trạng thái khởi tạo: "0000" mang ý nghĩa tập trống.
- Trạng thái kết thúc: "1111" mang ý nghĩa là tập cha.

Với tập cha gồm 4 phần tử, có 2^4 tập con với các biểu diễn:

vars	p(b)
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7

vars	p(b)
1000	8
1001	9
1010	10
1011	11
1100	12
1101	13
1110	14
1111	15

Với thứ tự từ điển, tập con sau lớn hơn tập con trước 1 đơn vị theo cách tính chuỗi nhị phân.

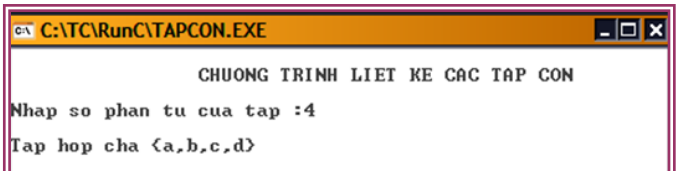


Gọi i : vị trí bit 0 đầu tiên từ bên phải.
Cho các bit 1 bên phải vị trí i thành 0
Cho bit i mang trị 1

```
i= n-1;
while (i>=0 && vars[i]=='1') vars[i--] = '0';
vars[i] = '1';
```

Mã lệnh:

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
#define TRUE 1
#define FALSE 0
```



```
#define lim 20
void Output(char vars[], int n,int count) //Xuất KQ
{  int d=0,sum=0;
   printf("\nChuoi thu %3d: ",count);
   for(int i=0; i<n; i++) putchar(vars[i]);
   printf("\   => Tap con thu %3d: {",count);
   for(i=0; i<n; i++)  if(vars[i]=='1') sum++;
   for(i=0; i<n; i++)
    { if(vars[i]=='1')
       {   printf("%c",97+i); //97 là a trong mã ASCII
           d++;
           if (d>0 && d<sum) printf(",");
       }
   }
  printf("}");
}
////////////////////////////////////////////////////////////////////
void main()
{int n;
  printf("\n\t\tCHUONG TRINH LIET KE CAC TAP CON");
  printf("\n\nNhap so phan tu cua tap :");
  scanf("%d",&n);
  printf("\nTap hop cha {");
  for(int i=0; i<n; i++)  //In tập hợp cha
  {   printf("%c",97+i); //97 là a trong mã ASCII
      if (i<(n-1)) printf(",");
  }
  printf("}\n\n");
//-----------------------------------
  char vars[lim];
//---Generate(vars,n);---------
   int Stop=FALSE;
   for(int i=0; i<n; i++) vars[i]='0'; //InitConfigure(vars,n);
   int count=1;
   Output(vars,n,count);// Xuất kết quả
   while(!Stop)
   {
      i=n-1;  //Begin of NextConfigure(vars,n);
      while(i>=0 && vars[i]=='1') vars[i--]='0';  //Cho cac bit 1 ben phai ve 0.
      vars[i]='1';  //The end of NextConfigure(vars,n)
      count++;
      Output(vars,n,count); // Xuất kết quả
      for(i=0; i<n; i++) //Kiểm tra xem đã là cấu hình cuối chưa
     {   if(vars[i]=='0') Stop= FALSE;
         else Stop= TRUE;
      }
   }
//----The end of Generate(vars,n)-------
  getch();
}////////////////////////////////////////////////////////////////////
```

5.5. Bài tập

Bài 5.1. Viết chương trình tính các bài toán sau:

a) Tính n!

b) Tính a^n với mọi số thực a và số tự nhiên n.

c) Tính tổng n số tự nhiên.

d) Nhập 2 số và nhập 1 phép tính từ bàn phím. Tính kết quả? (Gợi ý: dùng switch – case)

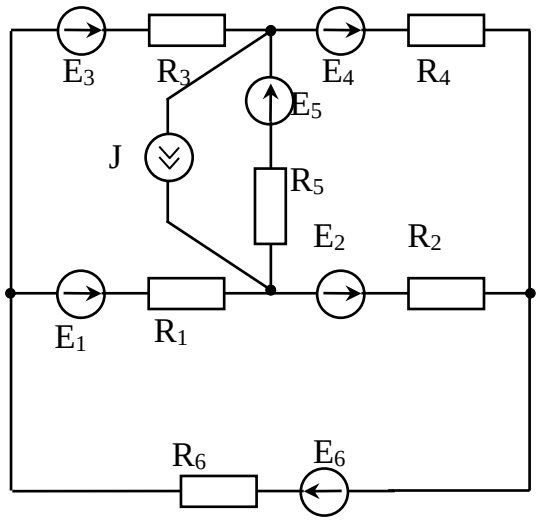
Bài 5.2. Nhập số liệu công suất của 1 nhà máy phát điện trong 24 giờ (mỗi giờ một số liệu tổng công suất đã phát). Sau đó tìm giờ có công suất phát thấp nhất, xuất kèm giá trị công suất? Tìm giờ có công suất phát lớn nhất, xuất kèm giá trị công suất?

Bài 5.3. Viết chương trình giải hệ phương trình <u>số thực</u> ***n*** ẩn.

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + ... + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + ... + a_{2n}x_n = b_2 \\ ... \\ a_{n1}x_1 + a_{n2}x_2 + ... + a_{nn}x_n = b_n \end{cases}$$

Biết các hệ số được nhập từ bàn phím.

Bài 5.4. Viết chương trình giải mạch điện sau:

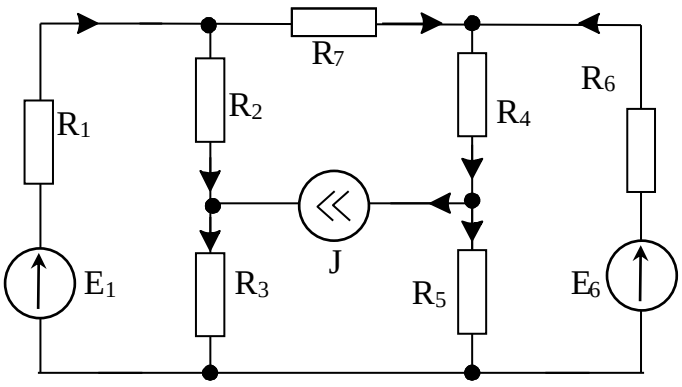


Biết: R_i, E_i (với $i = \overline{1,6}$), J nhập từ bàn phím.

Yêu cầu: 1. Tìm I_i (với $i = \overline{1,7}$)?

2. Chỉ cho phép nhập R > 0?
3. Tính công suất các nguồn điện?
4. Sắp xếp thứ tự các nguồn theo mức độ tăng dần công suất?

Bài 5.5. Viết chương trình giải giải mạch điện sau:



Biết: R_i, E_i (với $i = \overline{1,7}$), J nhập từ bàn phím.

Yêu cầu: 1. Tìm I_i (với $i = \overline{1,7}$)?

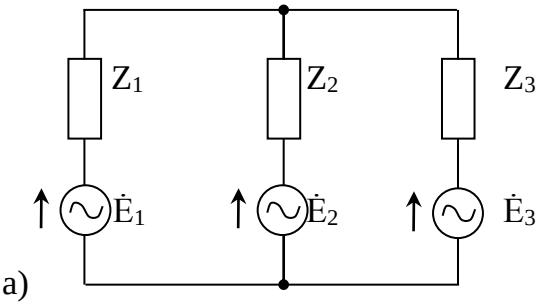
2. Có thể giải nhiều lần cho các thông số khác nhau, chỉ thoát thông qua lệnh "thoát"?

Bài 5.6. Viết chương trình giải hệ phương trình số phức ***n*** ẩn.

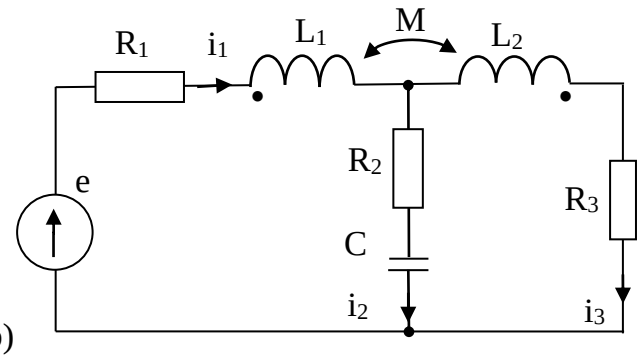
$$\begin{cases} (a_{11} + jb_{11}).(x_1 + jy_1) + (a_{12} + jb_{12}).(x_2 + jy_2) + ... + (a_{1n} + jb_{1n}).(x_n + jy_n) = c_1 + jd_1 \\ (a_{21} + jb_{21}).(x_1 + jy_1) + (a_{22} + jb_{22}).(x_2 + jy_2) + ... + (a_{2n} + jb_{2n}).(x_n + jy_n) = c_2 + jd_2 \\ ... \\ (a_{n1} + jb_{n1}).(x_1 + jy_1) + (a_{n2} + jb_{n2}).(x_2 + jy_2) + ... + (a_{nn} + jb_{nn}).(x_n + jy_n) = c_n + jd_n \end{cases}$$

Các thông số $[a_{ij} + j\, b_{ij}]$, $[c_i + j\, d_i]$ $(i, j \in N \setminus \{0\})$ nhập từ bàn phím.

Bài 5.7. Viết chương trình giải các mạch điện sau. Biết các thông số của được nhập từ bàn phím.

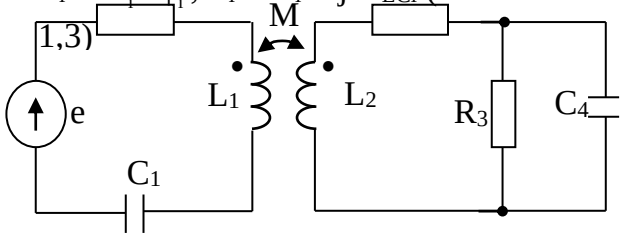


a)



b)

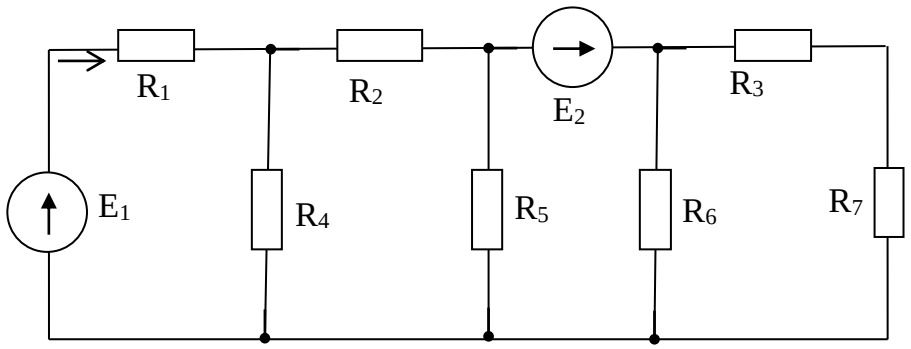
$\dot{E}_i = E_i \angle \varphi_i$, $Z_i = R_i + j\, X_{LCi}$ $(i = \overline{1,3})$



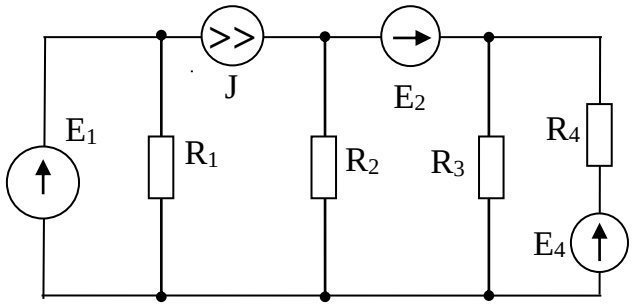
c)

Bài 5.8. Viết chương trình giải các mạch điện sau để tính công suất của các nguồn áp. Các thông số E_i, R_i nhập từ bàn phím.

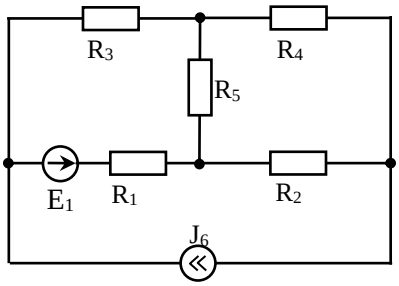
a)



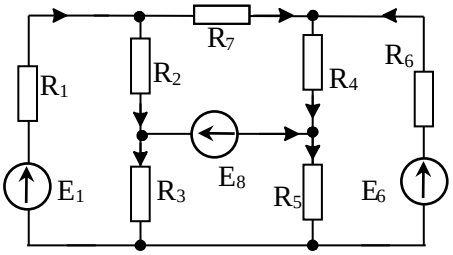
b)



c)



d)



Bài 5.9. Viết chương trình liệt kê các trường hợp có thể xảy ra hỏng hóc các thiết bị trong một trạm điện. Biết số lượng thiết bị và tên từng thiết bị được nhập từ bàn phím.

Yêu cầu:

1. Liệt kê được các trường hợp kèm theo danh sách tên thiết bị;
2. Trình bày rõ ràng, dễ giao tiếp.

Bài 5.10. Viết chương trình xếp lịch trực của nhà máy điện gồm n nhóm nhân viên vào 3 ca vận hành. Biết n được nhập từ bàn phím?

PHẦN 2. CÁC BÀI THỰC HÀNH

(Xem quyển tài liệu thực hành Tin học cơ sở)

Bài 1. Lập trình cơ bản

+ Vận dụng kiến thức đã học để viết một chương trình cơ bản;

+ Hoàn thiện kỹ năng tổ chức một chương trình C++, kỹ năng khai báo biến, kỹ năng tổ chức dữ liệu và kỹ năng thao tác với dữ liệu.

Bài 2. Lập trình ứng dụng các lệnh nhập xuất

+ Áp dụng thành thạo cấu trúc chương trình.

+ Áp dụng tốt việc khai báo biến, hằng.

+ Hoàn thiện kỹ năng sử dụng các lệnh nhập xuất để áp dụng cho các bài toán kỹ thuật trong lĩnh vực ngành học.

Bài 3. Lập trình ứng dụng cấu trúc if - else

+ Áp dụng thành thạo cấu trúc chương trình, khai báo biến, hằng và các lệnh nhập xuất.

+ Hoàn thiện kỹ năng sử dụng cấu trúc if – else để áp dụng cho các bài toán kỹ thuật trong lĩnh vực ngành học.

Bài 4. Lập trình ứng dụng cấu trúc switch – case

+ Áp dụng thành thạo cấu trúc chương trình, khai báo biến, hằng và các lệnh nhập xuất.

+ Hoàn thiện kỹ năng sử dụng cấu trúc switch – case để áp dụng cho các bài toán kỹ thuật trong lĩnh vực ngành học.

Bài 5. Lập trình ứng dụng cấu trúc for

+ Áp dụng thành thạo cấu trúc chương trình, khai báo biến, hằng và các lệnh nhập xuất.

+ Hoàn thiện kỹ năng sử dụng cấu trúc for để áp dụng cho các bài toán kỹ thuật trong lĩnh vực ngành học.

Bài 6. Lập trình ứng dụng cấu trúc while

+ Áp dụng thành thạo cấu trúc chương trình, khai báo biến, hằng và các lệnh nhập xuất.

+ Hoàn thiện kỹ năng sử dụng cấu trúc while để áp dụng cho các bài toán kỹ thuật trong lĩnh vực ngành học.

Bài 7. Lập trình ứng dụng cấu trúc do - while

+ Áp dụng thành thạo cấu trúc chương trình, khai báo biến, hằng và các lệnh nhập xuất.

+ Hoàn thiện kỹ năng sử dụng cấu trúc do - while để áp dụng cho các bài toán kỹ thuật trong lĩnh vực ngành học.

Bài 8. Lập trình ứng dụng các lệnh rẽ nhánh và lệnh nhảy

+ Áp dụng thành thạo cấu trúc chương trình, khai báo biến, hằng và các lệnh nhập xuất.

+ Hoàn thiện kỹ năng sử dụng các lệnh rẽ nhánh và lệnh nhảy để áp dụng cho các bài toán kỹ thuật trong lĩnh vực ngành học.

Bài 9. Lập trình ứng dụng cơ bản

+ Áp dụng thành thạo cấu trúc chương trình, khai báo biến, hằng và các lệnh nhập xuất.

+ Hoàn thiện kỹ năng chọn lựa và sử dụng phù hợp các cấu trúc lệnh if – else, switch – case, for, while, do – while, các lệnh rẽ nhánh và lệnh nhảy để áp dụng cho các bài toán kỹ thuật trong lĩnh vực ngành học.

Bài 10. Lập trình ứng dụng tổng hợp

+ Áp dụng thành thạo cấu trúc chương trình, khai báo biến, hằng và các lệnh nhập xuất.

+ Áp dụng thành thạo các cấu trúc lệnh if – else, switch – case, for, while, do – while, các lệnh rẽ nhánh và lệnh nhảy.

+ Hoàn thiện các kỹ năng lập trình cơ bản để giải quyết được các bài toán kỹ thuật trong lĩnh vực ngành học.

TÀI LIỆU THAM KHẢO

[1] Ngô Diện Tập (2005), *Vi điều khiển với lập trình C*, NXB Khoa học – Kỹ thuật.

[2] Ngô Diện Tập (2005), *Lập trình ghép nối máy tính trong Windows*, NXB Khoa học – Kỹ thuật.

[3] Robert Sedgewick (Trần Đoàn Thư, Bùi Thị Ngọc Nga, Nguyễn Hiệp Đoàn, Đặng Đức Trọng, Trần Hạnh Nhi dịch) (1995), *Cẩm nang thuật toán*, NXB Khoa học – Kỹ thuật.

[4] Derek M. Jones (2005),*The new C standard*.

[5] ISO/IEC (2010), *Strandard for programming language C++*.

[6] ISO/IEC (2010), *Programming language C++*.

[7] Đoàn Đức Tùng (chủ biên), Đoàn Thanh Bảo, Lê Thái Hiệp (2016), *Bài tập lý thuyết mạch điện (tập 1)*, NXB Xây Dựng, Hà Nội.

Phụ lục 1. Mã ASCII

Kí tự ASCII in được

Hệ 2 (Nhị phân)	Hệ 10 (Thập phân)	Đồ hoạ (Hiển thị ra được)
010 0000	32	Khoảng trống (SP)
010 0001	33	!
010 0010	34	"
010 0011	35	#
010 0100	36	$
010 0101	37	%
010 0110	38	&
010 0111	39	'
010 1000	40	(
010 1001	41	)
010 1010	42	*
010 1011	43	+
010 1100	44	,
010 1101	45	-
010 1110	46	.
010 1111	47	/
011 0000	48	0
011 0001	49	1
011 0010	50	2
011 0011	51	3
011 0100	52	4
011 0101	53	5
011 0110	54	6
011 0111	55	7
011 1000	56	8
011 1001	57	9
011 1010	58	:
011 1011	59	;
011 1100	60	<
011 1101	61	=
011 1110	62	>
011 1111	63	?
100 0000	64	@
100 0001	65	A
100 0010	66	B
100 0011	67	C
100 0100	68	D
100 0101	69	E
100 0110	70	F
100 0111	71	G
100 1000	72	H
100 1001	73	I
100 1010	74	J
100 1011	75	K
100 1100	76	L
100 1101	77	M
100 1110	78	N
100 1111	79	O
101 0000	80	P
101 0001	81	Q
101 0010	82	R
101 0011	83	S
101 0100	84	T
101 0101	85	U
101 0110	86	V
101 0111	87	W
101 1000	88	X
101 1001	89	Y
101 1010	90	Z
101 1011	91	[
101 1100	92	\
101 1101	93	]
101 1110	94	^
101 1111	95	_
110 0000	96	`
110 0001	97	a
110 0010	98	b
110 0011	99	c
110 0100	100	d
110 0101	101	e
110 0110	102	f
110 0111	103	g
110 1000	104	h
110 1001	105	i
110 1010	106	j
110 1011	107	k
110 1100	108	l
110 1101	109	m
110 1110	110	n

110 1111	111	o
111 0000	112	p
111 0001	113	q
111 0010	114	r
111 0011	115	s
111 0100	116	t
111 0101	117	u
111 0110	118	v
111 0111	119	w

111 1000	120	x
111 1001	121	y
111 1010	122	z
111 1011	123	{
111 1100	124	|
111 1101	125	}
111 1110	126	~

Phụ lục 2. TIỀN XỬ LÝ VÀ THƯ VIỆN TRONG C/C++

1. Preprocessor

#
#define
#elif
#else
#endif
#error
#if
#ifdef
#ifndef
#include
#line
#pragma
#undef

==========

Pound sign (#)

==========

The # (pound sign) indicates a preprocessor directive when it occurs as the first non-whitespace character on a line.

It signifies a compiler action, not necessarily associated with code generation.

and ## (double pound signs) are also used as operators to perform token replacement and merging during the preprocessor scanning phase.

#include (directive)

==========

Treats text in the file specified by filename as if it appeared in the current file.

Syntax:

#include "filename"

#include <filename>

#include "filename" searches the source path first, then the include path.

#include <filename> does not search the source directory.

Examples:

#include <stdio.h>

#include "main.h"

==========

#define (directive)

==========

Defines a macro

Syntax: #define <id1>[(<id2>, ...)] <token string>

The #define directive defines a macro.

Macros provide a mechanism for token replacement with or without a set of formal, function-line parameters.

All subsequent instances of the identifier <id1> in the source text will be replaced by the text defined by <token string>.

If <id1> is followed IMMEDIATELY by a "(", the identifiers following the "(" (<id2>, etc.) are treated like parameters to a function instead of as part of <token string>.

All instances of <id2> in <token string> will be replaced with the actual text defined for <id2> when <id1> is referenced in the source.

To continue the definitions on another line, end the current line with \.

Example:

#define MAXINT 32767

#define ctrl(x) ((x) \ - 64)

/* continued from preceding line */

It is legal but ill-advised to use Turbo C++ keywords as macro identifiers.

For example,

#define int long

is legal, but possibly catastrophic.

You can't use any of these global identifiers immediately after a #define or

#undef directive:

__DATE__ __FILE__ __LINE__

__STDC__ __TIME__

#undef (directive)

==========

Undefines a symbol

Syntax: #undef <identifier>

Undefines a symbol specified by <identifier> which was previously defined with a #define directive.

Example:

#undef DEBUG

#if, #ifdef, #ifndef, #else,

#elif, #endif (directives)

==========

Conditional compilation directives

Syntax:

#if <constant-expression>

#else

#endif

#if <constant-expression>

#elif <constant-expression>

#endif

The compiler only compiles the lines that follow the #if directive when <constant expression> evaluates to non-zero.

Otherwise, the compiler skips the lines that follow until it encounters the matching #else or #endif.

If the expression evaluates to false and there is a matching #else, the lines between the #else and the #endif are compiled.

#if directives can be nested, but matching #else and #endif directives must be in the same file as the #if.

#if ... #elif ... #endif

==========

#elif is like #else except the else block is only compiled if the expression after #elif is non-zero.

Alternate forms (#ifdef and #ifndef)

==========

The alternate form

#ifdef <identifier>

evaluates to 1 if the symbol specified by <identifier> has been previously defined with a #define directive.

The alternate form

#ifndef <identifier>

evaluates to 1 if the symbol specified by <identifier> has not been defined.

Example 1:

```
#if defined(NEARPOINTERS)
  space = farcoreleft();
```

```
#elif defined(FARPOINTERS)
  space = coreleft();
#else
#error Unsupported memory model
#endif
```

Example 2:

```
#ifdef DEBUG
  printf("Total space %d\n", space);
#endif
```

Example 3:

```
#ifndef Quiet
  PrintDiagnostics();
#endif
```

#error (directive)

==========

Issues error message

Syntax: #error <message>

If this line of code is compiled by the compiler, a fatal error message will be issued for this line and include the text defined by <message>.

Example:

```
#if !defined(MODEL)
#error Building model not defined
#endif
```

#line (directive)

==========

Causes the compiler to think that the line number of the next source line is given by <constant>, and the current input file is given by <identifier>.

Syntax: #line <constant> [<identifier>]

If <identifier> is missing, the current file name remains unchanged.

Example:

```
#line 55 main.cpp
```

#pragma (directive)

==========

Implementation-specific directives

Syntax: #pragma <directive name>

With #pragma, Turbo C++ can define whatever directives it desires without interfering with other compilers that support #pragma.

If the compiler doesn't recognize <directive name>, it ignores the #pragma directive without an error or warning message.

Turbo C++ supports the following #pragma directives:

```
#pragma argsused   #pragma exit      #pragma hdrfile
#pragma hdrstop    #pragma inline    #pragma option
#pragma saveregs   #pragma startup   #pragma warn
```

==========

2. Header Files

==========

ALLOC.H	ASSERT.H	BCD.H	BIOS.H
COMPLEX.H	CONIO.H	CTYPE.H	DIR.H
DIRENT.H	DOS.H	ERRNO.H	FCNTL.H
FLOAT.H	FSTREAM.H	GENERIC.H	GRAPHICS.H
IO.H	IOMANIP.H	IOSTREAM.H	LIMITS.H
LOCALE.H	MALLOC.H	MATH.H	MEM.H
PROCESS.H	SETJMP.H	SHARE.H	SIGNAL.H
STDARG.H	STDDEF.H	STDIO.H	STDIOSTR.H
STDLIB.H	STREAM.H	STRING.H	STRSTREA.H
SYS\STAT.H	SYS\TIMEB.H	SYS\TYPES.H	TIME.H
VALUES.H			

==========

4.1. ALLOC.H

Functions

brk calloc coreleft farcalloc
farcoreleft farfree farheapcheck farheapcheckfree
farheapchecknode farheapfillfree farheapwalk farmalloc
farrealloc free heapcheck heapcheckfree
heapchecknode heapfillfree heapwalk malloc
realloc sbrk

4.2. ASSERT.H

Includes STDIO.H STDLIB.H

Functions

assert

BCD.H

Declares the C++ class bcd, plus the overloaded operators for class bcd and for bcd math functions.

Functions

abs acos asin atan cos cosh exp log log10
pow pow10 real sin sinh sqrt tan tanh

Overloaded operators

```
!=    +=    +
-=    -     *=
*     ==    <
<=    >     >=
/=    /
```

4.3. BIOS.H

Functions

_bios_disk _bios_equip _bios_keybrd _bios_memsize
_bios_printer _bios_serialcom _bios_timeofday bioscom
biosdisk biosequip bioskey biosmemory
biosprint biostime

4.4. COMPLEX.H

Declares the C++ complex math functions.

All function names, member names, and operators have been borrowed from AT&T C++, except for the addition of acos, asin, atan, log10, tan, and tanh.

Includes IOSTREAM.H MATH.H

Functions

abs acos arg asin atan conj cos cosh exp
imag log log10 norm polar pow pow10 real sin
sinh sqrt tan tanh

Overloaded operators

```
+     +=    -     -=
*     *=    /     /=
==    !=    <<    >>
```

4.5. CONIO.H

Functions

cgets clreol clrscr cprintf
cputs cscanf delline getch
getche getpass gettext gettextinfo
gotoxy highvideo insline inp
inport inportb inpw kbhit
lowvideo movetext normvideo outp
outport outportb outpw putch
puttext _setcursortype textattr textbackground
textcolor textmode ungetch wherex
wherey window

4.6. CTYPE.H

Functions

_ftolower _ftoupper isalnum isalpha isascii iscntrl
isdigit isgraph islower isprint ispunct isspace
isupper isxdigit toascii tolower toupper

4.7. DIR.H

Functions

chdir findfirst findnext fnmerge fnsplit
getcurdir getcwd getdisk mkdir mktemp
rmdir searchpath setdisk

4.8. DIRENT.H

Functions

closedir opendir readdir rewinddir

4.9. DOS.H

Functions

absread abswrite allocmem bdos
bdosptr _chain_intr country ctrlbrk
delay disable _dos_allocmem _dos_close
_dos_creat _dos_creatnew dosexterr _dos_findfirst
_dos_findnext _dos_freemem _dos_getdate
_dos_getdiskfree
_dos_getdrive _dos_getfileattr _dos_getftime _dos_gettime

_dos_getvect _dos_keep _dos_open _dos_read
_dos_setblock _dos_setdate _dos_setdrive _dos_setfileattr
_dos_setftime _dos_settime _dos_setvect dostounix
_dos_write _emit_ enable FP_OFF
FP_SEG freemem geninterrupt getcbrk
getdate getdfree _getdrive getdta
getfat getfatd getftime getpsp
gettime getvect getverify _harderr
harderr _hardresume hardresume _hardretn
hardretn inp inport inportb
int86 int86x intdos intdosx
intr keep MK_FP nosound
outp outport outportb _OvrInitEms
_OvrInitExt parsfnm peek peekb
poke pokeb randbrd randbwr
segread setblock setcbrk setdate
setdta settime setvect setverify
sleep sound unixtodos unlink

4.10. ERRNO.H

No function

FCNTL.H

Defines "open flags" for open and similar library functions.

_fmode O_APPEND O_BINARY O_CHANGED
O_CREAT O_DENYALL O_DENYNONE O_DENYREAD
O_DENYWRITE O_DEVICE O_EXCL O_NOINHERIT
O_RDONLY O_RDWR O_TEXT O_TRUNC
O_WRONLY

4.11. FLOAT.H

Functions

_clear87 _fpreset _control87 _status87

FSTREAM.H

Declares the C++ stream classes that support file input and output.
Replaces the older, now outdated STDIOSTR.H.

Includes IOSTREAM.H

Classes

To get more Help on the classes in the FSTREAM.H header file (and their attendant member functions and data members), choose one of these Help links:

filebuf fstream fstreambase ifstream ofstream

4.12. GENERIC.H

Contains macros for generic class declarations.

GRAPHICS.H

Functions

arc bar bar3d
circle cleardevice clearviewport
closegraph detectgraph drawpoly
ellipse fillellipse fillpoly
floodfill getarccoords getaspectratio
getbkcolor getcolor getdefaultpalette
getdrivername getfillpattern getfillsettings
getgraphmode getimage getlinesettings
getmaxcolor getmaxmode getmaxx
getmaxy getmodename getmoderange
getpalette getpalettesize getpixel
gettextsettings getviewsettings getx
gety graphdefaults grapherrormsg
_graphfreemem _graphgetmem graphresult
imagesize initgraph installuserdriver
installuserfont line linerel
lineto moverel moveto
outtext outtextxy pieslice
putimage putpixel rectangle
registerbgidriver registerfarbgidriver registerbgifont
registerfarbgifont restorecrtmode sector
setactivepage setallpalette setaspectratio
setbkcolor setcolor setfillpattern
setfillstyle setgraphbufsize setgraphmode
setlinestyle setpalette setrgbpalette
settextjustify settextstyle setusercharsize
setviewport setvisualpage setwritemode
textheight textwidth

4.13. IO.H

Includes STDARG.H

Functions

access chmod _chmod chsize close
_close creat _creat creatnew creattemp
dup dup2 eof filelength getftime
ioctl isatty lock locking lseek
mktemp open _open read _read
remove rename setftime setmode sopen
tell umask unlink unlock write
_write

4.14. IOMANIP.H

Declares the C++ streams I/O manipulators and contains macros for creating paramaterized manipulators.

Includes GENERIC.H IOSTREAM.H

Classes

IAPP IOAPP OAPP SAPP
IMANIP IOMANIP OMANIP SMANIP

Overloaded operators

<< >>

4.15. IOSTREAM.H

Declares the C++ version 2.0 basic streams.

Replaces the older STREAM.H header file, which supports C++ version 1.2 streams.

Classes

To get more Help on the classes in the IOSTREAM.H header file (and their attendant member functions and data members), choose one of these Help links:

ios iostream iostream_withassign
istream istream_withassign ostream
ostream_withassign streambuf

4.16. LIMITS.H

No function

4.17. LOCALE.H

Declares functions that provide information specific to languages and countries.

Functions

localeconv setlocale

4.18. MALLOC.H

Includes ALLOC.H

Functions

stackavail

4.19. MATH.H

Functions

abs acos, acosl asin, asinl
atan, atanl atan2, atan2l atof, _atold
cabs, cabsl ceil, ceill cos, cosl
cosh, coshl exp, expl fabs, fabsl
floor, floorl fmod, fmodl frexp, frexpl
hypot, hypotl labs ldexp, ldexpl
log, logl log10, log101 matherr, _matherrl
modf, modfl poly, polyl pow, powl
pow10, pow10l sin, sinl sinh, sinhl
sqrt, sqrtl tan, tanl tanh, tanhl

4.20. MEM.H

Functions

_fmemccpy _fmemchr _fmemcmp _fmemcpy _fmemicmp
_fmemmove _fmemset _fmovmem _fsetmem memccpy
memchr memcmp memcpy memicmp memmove
memset movedata movmem setmem

4.21. PROCESS.H

Functions

abort _cexit _c_exit execl execle execlp
execlpe execv execve execvp execvpe exit
_exit getpid spawnl spawnle spawnlp spawnlpe
spawnv spawnve spawnvp spawnvpe system

4.22. SETJMP.H

Functions

longjmp setjmp

4.23. SHARE.H

No function

4.24. SIGNAL.H

Functions

raise signal

4.25. STDARG.H

Macros

va_arg va_end va_start

4.26. STDDEF.H

No function

4.27. STDIO.H

Functions

clearerr fclose fcloseall fdopen feof ferror
fflush fgetc fgetchar fgetpos fgets fileno
flushall fopen fprintf fputc fputchar fputs
fread freopen fscanf fseek fsetpos ftell
fwrite getc getchar gets getw perror
printf putc putchar puts putw remove
rename rewind rmtmp scanf setbuf setvbuf
sprintf sscanf strerror _strerror tempnam tmpfile
tmpnam ungetc unlink vfprintf vfscanf vprintf
vscanf vsprintf vsscanf

4.28. STDIOSTR.H

Declares the C++ (version 2.0) stream classes for use with stdio FILE structures. This file has been included for upward compatibility; use the standard classes and functions in FSTREAM.H instead.

4.29. STDLIB.H

Functions

abort abs atexit atof atoi
atol bsearch calloc div ecvt
exit _exit fcvt free _fullpath
gcvt getenv itoa labs ldiv
lfind _lrotl _lrotr lsearch ltoa
_makepath malloc max mblen mbtowc
mbstowcs min putenv qsort rand
random randomize realloc _rotl _rotr
_searchenv _splitpath srand strtod strtol
_strtold strtoul swab system time
ultoa wctomb wcstombs

4.30. STREAM.H

Declares the C++ (version 1.2) stream classes for use with stdio FILE structures. Provided for compatibility with C++ version 1.2.

If you plan to convert to C++ version 2.0, use IOSTREAM.H and read the "Using C++ Streams" chapter in the Programmer's Guide.

Includes STDIO.H

4.31. STRING.H

Functions

_fmemccpy _fmemchr _fmemcmp _fmemcpy _fmemicmp
_fmemset _fstrcat _fstrchr _fstrcmp _fstrcpy
_fstrcspn _fstrdup _fstricmp _fstrlen _fstrlwr
_fstrncat _fstrncmp _fstrnicmp _fstrncpy _fstrnset
_fstrpbrk _fstrrchr _fstrrev _fstrset _fstrspn
_fstrstr _fstrtok _fstrupr memccpy memchr
memcmp memcpy memicmp memmove memset
movedata movmem setmem stpcpy strcat
strchr strcmp strcmpi strcpy strcspn
strdup _strerror strerror stricmp strlen
strlwr strncat strncmp strncmpi strncpy
strnicmp strnset strpbrk strrchr strrev
strset strspn strstr strtok strxfrm
strupr

4.32. STRSTREA.H

Declares the C++ (version 2.0) stream classes for use with byte arrays in memory.

Includes IOSTREAM.H

Classes

To get more Help on the classes in the STRSTREA.H header file (and their attendant member functions and data members), choose one of these Help links:

istrstream ostrstream strstream strstreambase
strstreambuf

4.33. SYS\STAT.H

Functions

fstat stat

4.34. SYS\TIMEB.H

Functions

ftime

4.35. SYS\TYPES.H

No function

4.36. TIME.H

Functions

asctime clock ctime difftime gmtime localtime
mktime stime strftime _strdate _strtime time
tzset

4.37. VALUES.H

Defines UNIX compatible constants for limits to float and double values.

BITSPERBYTE DMAXEXP DMAXPOWTWO DMINEXP
DSIGNIF FMAXEXP FMAXPOWTWO FMINEXP
FSIGNIF _FEXPLEN HIBITI HIBITL
HIBITS _LENBASE MAXDOUBLE MAXFLOAT
MAXINT MAXLONG MAXSHORT MINDOUBLE
MINFLOAT