TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

KHOA CÔNG NGHỆ THÔNG TIN

----------o0o----------

BÀI GIẢNG MÔN HỌC

LẬP TRÌNH NÂNG CAO

Tên môn học: **Lập trình nâng cao**

Số tin chỉ/ĐVHT: **3TC (2LT: 1TH)**

Hệ đào tạo: **Đại học** Ngành**: Các ngành**

Số tín chỉ thực hành: **1 TC**

Giảng viên biên soạn:

1. **Nguyễn Hải Minh**
2. **Nguyễn Tuấn Anh**
3. **Hà Mạnh Hùng**
4. **Nguyễn Quang Hiệp**
5. **Trần Lâm**

Bộ môn: **Công nghệ lập trình và Ứng dụng**

Thái Nguyên - 2016

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

KHOA CÔNG NGHỆ THÔNG TIN

----------o0o----------

BÀI GIẢNG MÔN HỌC

LẬP TRÌNH NÂNG CAO

KHOA PHÊ DUYỆT **BỘ MÔN PHÊ DUYỆT** **GIÁO VIÊN PHỤ TRÁCH**

Thái nguyên, Năm 2016

MỤC LỤC

CHƯƠNG 1: CON TRỎ 4
1.1 KHÁI NIỆM CON TRỎ 4
1.1.1 Khai báo con trỏ 4
1.1.2 Sử dụng con trỏ 4
1.2 CON TRỎ VÀ MẢNG 7
1.2.1 Con trỏ và mảng một chiều 7
1.2.2 Con trỏ và mảng nhiều chiều 10
1.3. CON TRỎ HÀM 11
1.4 CẤP PHÁT BỘ NHỚ ĐỘNG 14
1.4.1 Cấp phát bộ nhớ động cho biến 14
1.4.2 Cấp phát bộ nhớ cho mảng động một chiều 15
1.4.3 Cấp phát bộ nhớ cho mảng động nhiều chiều 17
CHƯƠNG 2: CÁC DÒNG NHẬP XUẤT VÀ TỆP TIN 23
2.1. NHẬP/XUẤT VỚI CIN/COUT 24
2.1.1. Toán tử nhập >> 24
2.1.2. Các hàm nhập kí tự và xâu kí tự 25
2.1.3. Toán tử xuất << 28
2.2. ĐỊNH DẠNG 28
2.2.1.Các phương thức định dạng 28
2.2.2. Các cờ định dạng 29
2.2.3. Các bộ và hàm định dạng 31
2.3. IN RA MÁY IN 32
2.4. LÀM VIỆC VỚI FILE 33
2.4.1. Tạo đối tượng gắn với file 33
2.4.2 Đóng file và giải phóng đối tượng 34
2.4.3. Kiểm tra sự tồn tại của file, kiểm tra hết file 37
2.4.4. Đọc ghi đồng thời trên file 38
2.4.5. Di chuyển con trỏ file 39

2.5. NHẬP/XUẤT NHỊ PHÂN 41
2.5.1. Khái niệm về 2 loại file: văn bản và nhị phân 41
2.5.2.Đọc, ghi kí tự 41
2.5.3. Đọc, ghi dãy kí tự 42
2.5.4. Đọc ghi đồng thời 43
CHƯƠNG 3: DỮ LIỆU KIỂU CẤU TRÚC VÀ HỢP 48
3.1. KIỂU CẤU TRÚC 48
3.1.1. Khai báo, khởi tạo 48
3.1.2.Truy nhập các thành phần kiểu cấu trúc 50
3.1.3. Phép toán gán cấu trúc 52
3.1.4. Các ví dụ minh hoạ 53
3.1.5.Hàm với cấu trúc 56
3.1.6.Cấu trúc với thành phần kiểu bit 67
3.1.7.Câu lệnh typedef 68
3.1.8.Hàm sizeof() 69
3.2. CẤU TRÚC TỰ TRỎ VÀ DANH SÁCH LIÊN KẾT 69
3.2.1.Cấu trúc tự trỏ 70
3.2.2. Khái niệm danh sách liên kết 72
3.2.3. Các phép toán trên danh sách liên kết 73
3.3. KIỂU HỢP 79
3.3.1. Khai báo 79
3.3.2. Truy cập 79
3.4. KIỂU LIỆT KÊ 80
CHƯƠNG 4: XỬ LÝ NGOẠI LỆ 86
4.1. Xử lý ngoại lệ 86
4.2. Ném Exception trong C++ 87
4.3. Bắt Exception trong C++ 87
4.4.Chuẩn Exception trong C++ 88
4.5. Định nghĩa Exception mới trong C++ 90

CHƯƠNG 5: MỘT SỐ VẤN ĐỀ 92
5.1. Một số quy tắc trong lập trình C++ 92
5.1.1. Tổ chức chương trình 92
5.1.2. Chuẩn tài liệu 93
5.1.3. Tên 93
5.1.4. Định dạng 95
5.1.5. Thiết kế 96
5.1.6. Code 97
5.2. Namespaces 103
5.1.1. Using namespace 104
5.1.2. Định nghĩa bí danh 106
5.1.3. Namespace std 106
Tài liệu tham khảo 108

CHƯƠNG 1: CON TRỎ

1.1 KHÁI NIỆM CON TRỎ

1.1.1 Khai báo con trỏ

Con trỏ là một biến đặc biệt chứa địa chỉ của một biến khác. Con trỏ có cùng kiểu dữ liệu với kiểu dữ liệu của biến mà nó trỏ tới. Cú pháp khai báo một con trỏ như sau:

<Kiểu dữ liệu> *<Tên con trỏ>;

Trong đó:

Kiểu dữ liệu: Có thể là các kiểu dữ liệu cơ bản của C++, hoặc là kiểu dữ liệu có cấu trúc, hoặc là kiểu đối tượng do người dùng tự định nghĩa.

Tên con trỏ: Tuân theo qui tắc đặt tên biến của C++:

- Chỉ được bắt đầu bằng một kí tự (chữ), hoặc dấu gạch dưới "_".
- Bắt đầu từ kí tự thứ hai, có thể có kiểu kí tự số.
- Không có dấu trống (space bar) trong tên biến.
- Có phân biệt chữ hoa và chữ thường.
- Không giới hạn độ dài tên biến.

Ví dụ, để khai báo một biến con trỏ có kiểu là int và tên là pointerInt, ta viết như sau:

```
int *pointerInt;
```

Lưu ý

Có thể viết dấu con trỏ "*" ngay sau kiểu dữ liệu, nghĩa là hai cách khai báo sau là tương đương:

```
int *pointerInt;
int* pointerInt;
```

Các cách khai báo con trỏ như sau là sai cú pháp:

```
*int pointerInt; // Khai báo sai con trỏ int pointer
int*; // Khai báo sai con trỏ
```

1.1.2 Sử dụng con trỏ

Con trỏ được sử dụng theo hai cách:

Dùng con trỏ để lưu địa chỉ của biến để thao tác

Lấy giá trị của biến do con trỏ trỏ đến để thao tác

Dùng con trỏ để lưu địa chỉ của biến

Bản thân con trỏ sẽ được trỏ vào địa chỉ của một biến có cùng kiểu dữ liệu với nó. Cú pháp của phép gán như sau:

<Tên con trỏ> = &<tên biến>;

Lưu ý

Trong phép toán này, tên con trỏ không có dấu "*".

Ví dụ:

```
int x, *px;
px = &x;
```

sẽ cho con trỏ px có kiểu int trỏ vào địa chỉ của biến x có kiểu nguyên. Phép toán &<Tên biến> sẽ cho địa chỉ của biến tương ứng.

Lấy giá trị của biến do con trỏ trỏ đến

Phép lấy giá trị của biến do con trỏ trỏ đến được thực hiện bằng cách gọi tên:

*<Tên con trỏ>;

Lưu ý

- Trong phép toán này, phải có dấu con trỏ "*". Nếu không có dấu con trỏ, sẽ trở thành phép lấy địa chỉ của biến do con trỏ trỏ tới.

Ví dụ:

```
int x = 12, y, *px;
px = &y;
*px = x;
```

Quá trình diễn ra như sau:

```
int x = 12, y, *px;  //x=12, y=0, px → null
px = &y;  //x=12, y=0  ← px
px = x;   // x=12, y=x=12 ← px
```

con trỏ px vẫn trỏ tới địa chỉ biến y và giá trị của biến y sẽ là 12.

Phép gán giữa các con trỏ

Các con trỏ cùng kiểu có thể gán cho nhau thông qua phép gán và lấy địa chỉ con trỏ:

<Tên con trỏ 1> = <Tên con trỏ 2>;

Lưu ý

- Trong phép gán giữa các con trỏ, bắt buộc phải dùng phép lấy **địa chỉ của biến** do con trỏ trỏ tới (không có dấu "*" trong tên con trỏ) mà không được dùng phép lấy **giá trị của biến** do con trỏ trỏ tới.

- Hai con trỏ phải cùng kiểu. Trong trường hợp hai con trỏ khác kiểu, phải sử dụng các phương thức ép kiểu tương tự như trong phép gán các biến thông thường có kiểu khác nhau.

Ví dụ:

```
int x = 12, *px, *py;
px = &x;
py = px;
int x = 12, *px, *py;  //x=12, px →null, py→ null
px = &x;       // x=12 ← px, py→ null
py = px; //  x=12 ←px, py→x=12
```

con trỏ py cũng trỏ vào địa chỉ của biến x như con trỏ px. Khi đó *py cũng có giá trị 12 giống như *px và là giá trị của biến x.

Chương trình 1.1 minh hoạ việc dùng con trỏ giữa các biến của một chương trình C++

Chương trình 1.1

```
#include <stdio.h>
#include <conio.h>
void main(void){
    int x = 12, *px, *py;
    cout << "x = " << x << endl;
    px = &x;          // Con trỏ px trỏ tới địa chỉ của x
    cout << "px = &x, *px = " << *px << endl;
  *px=*px+20; //Nộidungcủapxlà32
    cout << "*px = *px+20, x = " << x << endl;
    py = px; // Cho py trỏ tới chỗ mà px trỏ: địa chỉ của x
   *py += 15; // Nội dung của py là 47
   cout<<"py=px,*py+=15,x= "<<x<<endl;
}
```

Trong chương trình 1.1, ban đầu biến x có giá trị 12. Sau đó, con trỏ px trỏ vào địa chỉ của biến x nên con trỏ px cũng có giá trị 12. Tiếp theo, ta tăng giá trị của con trỏ px thêm 20, giá trị của con trỏ px là 32. Vì px đang trỏ đến địa chỉ của x nên x cũng có giá trị là 32. Sau đó, ta cho con trỏ py trỏ đến vị trí mà px đang trỏ tới (địa chỉ của biến x) nên py cũng có giá trị 32. Cuối cùng, ta tăng giá trị của con trỏ py thêm 15, py sẽ có giá trị 37. Vì py cũng đang trỏ đến địa chỉ của x nên x cũng có giá trị 37. Do đó, ví dụ 1.1 sẽ in ra kết quả như sau:

x = 12

px = &x, *px = 12

*px = *px + 20, x = 32

py = px, *py += 15, x = 47

1.2 CON TRỎ VÀ MẢNG

1.2.1 Con trỏ và mảng một chiều

Mảng một chiều

Trong C++, tên một mảng được coi là một kiểu con trỏ hằng, được định vị tại một vùng nhớ xác định và địa chỉ của tên mảng trùng với địa chỉ của phần tử đầu tiên của mảng.

Ví dụ khai báo:

int A[5];

thì địa chỉ của mảng A (cũng viết là A) sẽ trùng với địa chỉ phần tử đầu tiên của mảng A (là &A[0]) nghĩa là:

A = &A[0];

Quan hệ giữa con trỏ và mảng

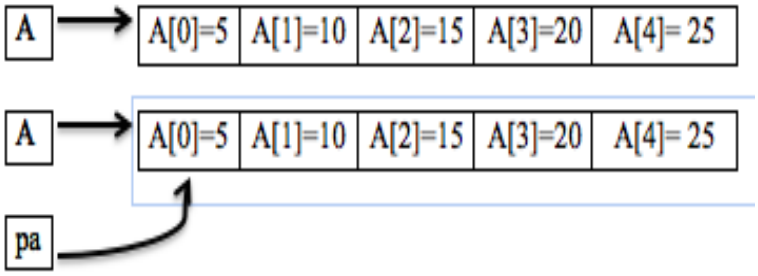
Vì tên của mảng được coi như một con trỏ hằng, nên nó có thể được gán cho một con trỏ có cùng kiểu.

Ví dụ khai báo:

int A[5] = {5, 10, 15, 20, 25};

int *pa = A;

int A[5] = {5, 10, 15, 20, 25};



int *pa = A;

thì con trỏ pa sẽ trỏ đến mảng A, tức là trỏ đến địa chỉ của phần tử A[0], cho nên hai khai báo sau là tương đương:

pa = A;

pa = &A[0];

Với khai báo này, thì địa chỉ trỏ tới của con trỏ pa là địa chỉ của phần tử A[0] và giá trị của con trỏ pa là giá trị của phần tử A[0], tức là *pa = 5;

Phép toán trên con trỏ và mảng

Khi một con trỏ trỏ đến mảng, thì các phép toán tăng hay giảm trên con trỏ

sẽ tương ứng với phép dịch chuyển trên mảng.

Ví dụ khai báo:

int A[5] = {5, 10, 15, 20, 25};

int *pa = &A[2];

thì con trỏ pa sẽ trỏ đến địa chỉ của phần tử A[2] và giá trị của pa là:

*pa = A[2] = 15.

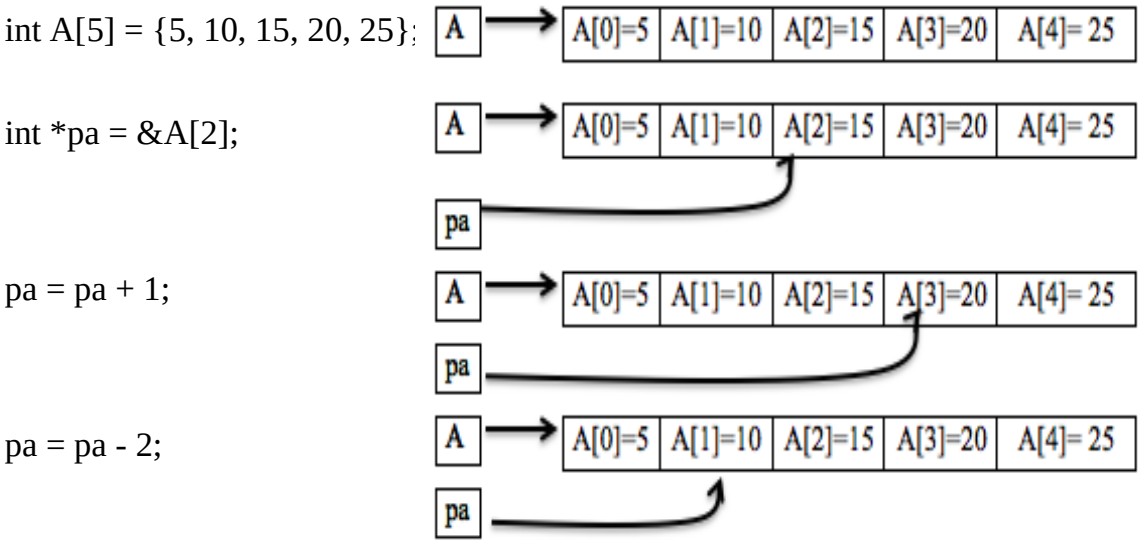
Khi đó, phép toán:

pa = pa + 1;

sẽ đưa con trỏ pa trỏ đến địa chỉ của phần tử tiếp theo của mảng A, đó là địa chỉ của A[3]. Sau đó, phép toán:

pa = pa – 2;

sẽ đưa con trỏ pa trỏ đến địa chỉ của phần tử A[1]



Lưu ý:

• Hai phép toán pa++ và *pa++ có tác dụng hoàn toàn khác nhau trên mảng, pa++ là thao tác trên con trỏ, tức là trên bộ nhớ, nó sẽ đưa con trỏ pa trỏ đến địa chỉ của phần tử tiếp theo của mảng. *pa++ là phép toán trên giá trị, nó tăng giá trị hiện tại của phần tử mảng lên một đơn vị.

Ví dụ:

int A[5] = {5, 10, 15, 20, 25};

int *pa = &A[2];

Thì pa++ là tương đương với pa = &A[3] và *pa = 20.

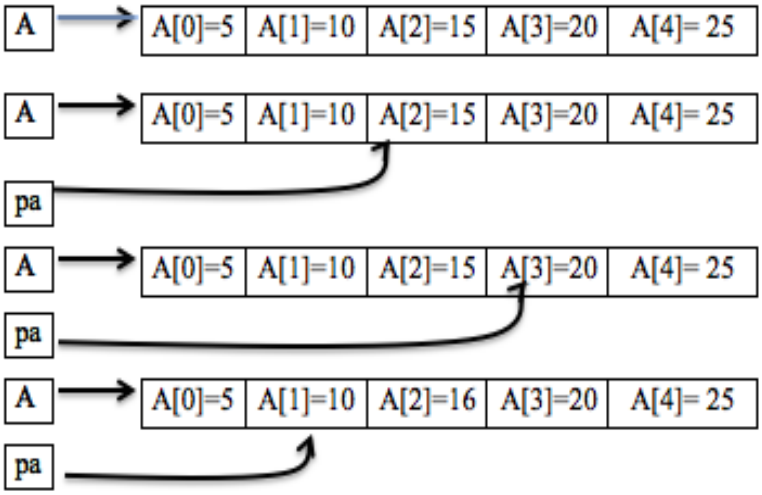
Nhưng *pa++ lại tương đương với pa = &A[2] và *pa = 15+1 = 16, A[2] = 16.

int A[5] = {5, 10, 15, 20, 25};

int *pa = &A[2];

pa ++;

Nhưng *pa ++;



Trong trường hợp:

int A[5] = {5, 10, 15, 20, 25};

int *pa = &A[4];

thì phép toán pa++ sẽ đưa con trỏ pa trỏ đến một địa chỉ không xác định. Lí do là A[4] là phần tử cuối của mảng A, nên pa++ sẽ trỏ đến địa chỉ ngay sau địa chỉ của A[4], địa chỉ này nằm ngoài vùng chỉ số của mảng A nên không xác định. Tương tự với trường hợp pa=&A[0], phép toán pa-- cũng đưa pa trỏ đến một địa chỉ không xác định

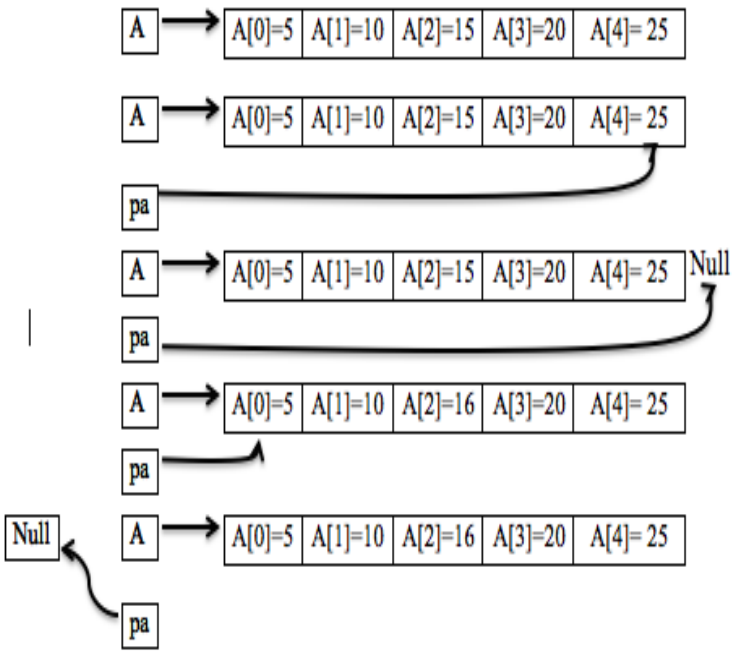
int A[5] = {5, 10, 15, 20, 25};

int *pa = &A[4];

pa ++;

pa = &A[0];

pa--;



• Vì mảng A là con trỏ hằng, cho nên không thể thực hiện các phép toán trên A mà chỉ có thể thực hiện trên các con trỏ trỏ đến A: các phép toán pa++ hoặc pa--là hợp lệ, nhưng các phép toán A++ hoặc A--là không hợp lệ.

Chương trình 2.2a minh hoạ việc cài đặt một thủ tục sắp xếp các phần tử của một mảng theo cách thông thường.

Chương trình 2.2a

```
void SortArray(int A[], int n){
    int temp;
    for(int i=0; i<n-1; i++)
    for(int j=i+1; j<n; j++)
}
if(A[i] > A[j]){
    temp = A[i];
    A[i] = A[j];
    A[j] = temp;
}
}
```

Chương trình 2.2b cài đặt một thủ tục tương tự bằng con trỏ. Hai thủ tục này có chức năng hoàn toàn giống nhau.

Chương trình 2.2b

```
void SortArray(int *A, int n){
    int temp;
    for(int i=0; i<n-1; i++)
    for(int j=i+1; j<n; j++)
}
if(*(A+i) > *(A+j)){
    temp = *(A+i);
    *(A+i) = *(A+j);
    *(A+j) = temp;
}
```

Trong chương trình 2.2b, thay vì dùng một mảng, ta dùng một con trỏ để trỏ đến mảng cần sắp xếp. Khi đó, ta có thể dùng các thao tác trên con trỏ thay vì các thao tác trên các phần tử mảng.

1.2.2 Con trỏ và mảng nhiều chiều

Con trỏ và mảng nhiều chiều

Một câu hỏi đặt ra là nếu một ma trận một chiều thì tương đương với một con trỏ, vậy một mảng nhiều chiều thì tương đương với con trỏ như thế nào?

Xét ví dụ:

```
int A[3][3] = {
```

```
        {5, 10, 15},
                        {20, 25, 30},
                        {35, 40, 45}
};
```

Khi đó, địa chỉ của ma trận A chính là địa chỉ của hàng đầu tiên của ma trận A, và cũng là địa chỉ của phần tử đầu tiên của hàng đầu tiên của ma trận A:

- Địa chỉ của ma trận A: A = A[0] = *(A+0) = &A[0][0];
- Địa chỉ của hàng thứ nhất: A[1] = *(A+1) = &A[1][0];
- Địa chỉ của hàng thứ i: A[i] = *(A+i) = &A[i][0];
- Địa chỉ phần tử &A[i][j] = (*(A+i)) + j;
- Giá trị phần tử A[i][j] = *((*(A+i)) + j);

Như vậy, một mảng hai chiều có thể thay thế bằng một mảng một chiều các con trỏ cùng kiểu:

int A[3][3];

có thể thay thế bằng:

int (*A)[3];

Con trỏ trỏ tới con trỏ

Vì một mảng hai chiều int A[3][3] có thể thay thế bằng một mảng các con trỏ int (*A)[3]. Hơn nữa, một mảng int A[3] lại có thể thay thế bằng một con trỏ int *A. Do vậy, một mảng hai chiều có thể thay thế bằng một mảng các con trỏ, hoặc một con trỏ trỏ đến con trỏ. Nghĩa là các cách viết sau là tương đương:

```
int A[3][3];
int (*A)[3];
int **A;
```

1.3. CON TRỎ HÀM

Mặc dù hàm không phải là một biến cụ thể nên không có một địa chỉ xác định. Nhưng trong khi chạy, mỗi một hàm trong C++ cũng có một vùng nhớ xác định, do vậy, C++ cho phép dùng con trỏ để trỏ đến hàm. Con trỏ hàm được dùng để truyền tham số có dạng hàm.

Khai báo con trỏ hàm

Con trỏ hàm được khai báo tương tự như khai báo nguyên mẫu hàm thông thường trong C++, ngoại trừ việc có thêm kí hiệu con trỏ "*" trước tên hàm. Cú pháp khai báo con trỏ hàm như sau:

<Kiểu dữ liệu trả về> (*<Tên hàm>)([<Các tham số>]);

Trong đó:

- **Kiểu dữ liệu trả về**: là các kiểu dữ liệu thông thường của C++ hoặc kiểu do người dùng tự định nghĩa.
- **Tên hàm**: tên do người dùng tự định nghĩa, tuân thủ theo quy tắc đặt tên biến trong C++
- **Các tham số**: có thể có hoặc không (phần trong dấu "[]" là tuỳ chọn). Nếu có nhiều tham số, mỗi tham số được phân cách nhau bởi dấu phẩy.

Ví dụ khai báo: int (*Calcul)(int a, int b);

là khai báo một con trỏ hàm, tên là Calcul, có kiểu int và có hai tham số cũng là kiểu int.

Lưu ý:

Dấu "()" bao bọc tên hàm là cần thiết để chỉ ra rằng ta đang khai báo một con trỏ hàm. Nếu không có dấu ngoặc đơn này, trình biên dịch sẽ hiểu rằng ta đang khai báo một hàm thông thường và có giá trị trả về là một con trỏ.

Ví dụ, hai khai báo sau là khác nhau hoàn toàn:

```
// Khai báo một con trỏ hàm
int (*Calcul)(int a, int b);
// Khai báo một hàm trả về kiểu con trỏ
int *Calcul(int a, int b);
```

Sử dụng con trỏ hàm

Con trỏ hàm được dùng khi cần gọi một hàm như là tham số của một hàm khác. Khi đó, một hàm được gọi phải có khuôn mẫu giống với con trỏ hàm đã được khai báo.

Ví dụ, với khai báo :

```
int (*Calcul)(int a, int b);
```

thì có thể gọi các hàm có hai tham số kiểu int và trả về cũng kiểu int như sau:

```
int add(int a, int b);
int sub(int a, int b);
```

nhưng không được gọi các hàm khác kiểu tham số hoặc kiểu trả về như sau:

```
int add(float a, int b);
int add(int a);
char* sub(char* a, char* b):
```

Chương trình 2.3

```
#include <ctype.h>
#include <string.h>
// Hàm có sử dụng con trỏ hàm như tham số
void Display(char[] str, int (*Xtype)(int c)){
    int index = 0;
    while(str[index] != '\0'){
        cout << (*Xtype)(str[index]); // Sử dụng con trỏ hàm
index ++; }
return; }
// Hàm main, dùng lời gọi hàm đến con trỏ hàm
void main(){
    char input[500];
    cout << "Enter the string: ";
    cin >> input;
char reply;
    cout << "Display the string in uppercase or lowercase (u,l): ";
    cin >> reply;
    if(reply == 'l')  // Hiển thị theo dạng lowercase
        Display(str, tolower);
    else            // Hiển thị theo dạng uppercase
        Display(str, toupper);
    return;
}
```

Chương trình 2.3 khai báo hàm Display() có sử dụng con trỏ hàm có khuôn mẫu

```
int (*Xtype)(int c);
```

Trong hàm main, con trỏ hàm này được gọi bởi hai thể hiện là các hàm tolower() và hàm toupper(). Hai hàm này được khai báo trong thư viện ctype.h với mẫu như sau:

```
int tolower(int c);
int toupper(int c);
```

Hai khuôn mẫu này phù hợp với con trỏ hàm Xtype trong hàm Display() nên lời gọi hàm Display() trong hàm main là hợp lệ.

1.4 CẤP PHÁT BỘ NHỚ ĐỘNG

Xét hai trường hợp sau đây:

Trường hợp 1, khai báo một con trỏ và gán giá trị cho nó: int *pa = 12;

Trường hợp 2, khai báo con trỏ đến phần tử cuối cùng của mảng rồi tăng thêm một đơn vị cho nó:

```
int A[5] = {5, 10, 15, 20, 25};
int *pa = &A[4];
pa++;
```

Trong cả hai trường hợp, ta đều không biết thực sự con trỏ pa đang trỏ đến địa chỉ nào trong bộ nhớ: trường hợp 1 chỉ ra rằng con trỏ pa đang trỏ tới một địa chỉ không xác định, nhưng lại chứa giá trị là 12 do được gán vào. Trường hợp 2, con trỏ pa đã trỏ đến địa chỉ ngay sau địa chỉ phần tử cuối cùng của mảng A, đó cũng là một địa chỉ không xác định. Các địa chỉ không xác định này là các địa chỉ nằm ở vùng nhớ tự do còn thừa của bộ nhớ. Vùng nhớ này có thể bị chiếm dụng bởi bất kì một chương trình nào đang chạy.

Do đó, rất có thể các chương trình khác sẽ chiếm mất các địa chỉ mà con trỏ pa đang trỏ tới. Khi đó, nếu các chương trình thay đổi giá trị của địa chỉ đó, giá trị pa cũng bị thay đổi theo mà ta không thể kiểm soát được. Để tránh các rủi ro có thể gặp phải, C++ yêu cầu phải cấp phát bộ nhớ một cách tường minh cho con trỏ trước khi sử dụng chúng.

1.4.1 Cấp phát bộ nhớ động cho biến

Cấp phát bộ nhớ động

Thao tác cấp phát bộ nhớ cho con trỏ thực chất là gán cho con trỏ một địa chỉ xác định và đưa địa chỉ đó vào vùng đã bị chiếm dụng, các chương trình khác không thể sử dụng địa chỉ đó. Cú pháp cấp phát bộ nhớ cho con trỏ như sau:

<tên con trỏ> = new <kiểu con trỏ>;

Ví dụ, khai báo:

```
int *pa;
pa = new int;
```

sẽ cấp phát bộ nhớ hợp lệ cho con trỏ pa.

Lưu ý:

Ta có thể vừa cấp phát bộ nhớ, vừa khởi tạo giá trị cho con trỏ theo cú pháp sau:

```
int *pa;
pa = new int(12);
```

sẽ cấp phát cho con trỏ pa một địa chỉ xác định, đồng thời gán giá trị của con trỏ

*pa = 12.

Giải phóng bộ nhớ động

Địa chỉ của con trỏ sau khi được cấp phát bởi thao tác **new** sẽ trở thành vùng nhớ đã bị chiếm dụng, các chương trình khác không thể sử dụng vùng nhớ đó ngay cả khi ta không dùng con trỏ nữa. Để tiết kiệm bộ nhớ, ta phải huỷ bỏ vùng nhớ của con trỏ ngay sau khi không dùng đến con trỏ nữa. Cú pháp huỷ bỏ vùng nhớ của con trỏ như sau:

delete <tên con trỏ>;

Ví dụ:

```
int *pa= new int (12);
delete(pa);
```

Lưu ý:

- Một con trỏ, sau khi bị giải phóng địa chỉ, vẫn có thể được cấp phát một vùng nhớ mới hoặc trỏ đến một địa chỉ mới: int *pa = new int(12); // Khai báo con trỏ pa, cấp phát bộ nhớ // và gán giá trị ban đầu cho pa là 12. delete pa; // Giải phóng vùng nhớ vừa cấp cho pa. int A[5] = {5, 10, 15, 20, 25}; pa = A; // Cho pa trỏ đến địa chỉ của mảng A
- Nếu có nhiều con trỏ cùng trỏ vào một địa chỉ, thì chỉ cần giải phóng bộ nhớ của một con trỏ, tất cả các con trỏ còn lại cũng bị giải phóng bộ nhớ:

```
int *pa = new int(12); // *pa = 12
int *pb = pa;   // pb trỏ đến cùng địa chỉ pa.
*pb += 5; // *pa = *pb = 17
delete pa; // Giải phóng cả pa lẫn pb
```

Một con trỏ sau khi cấp phát bộ nhớ động bằng thao tác **new**, cần phải phóng bộ nhớ trước khi trỏ đến một địa chỉ mới hoặc cấp phát bộ nhớ mới:

```
int*pa=newint(12);   //pađượccấpbộnhớvà*pa=12
*pa = new int(15);// pa trỏ đến địa chỉ khác và *pa = 15.
                  // địa chỉ cũ của pa vẫn bị coi là bận
```

1.4.2 Cấp phát bộ nhớ cho mảng động một chiều

Cấp phát bộ nhớ cho mảng động một chiều

Mảng một chiều được coi là tương ứng với một con trỏ cùng kiểu. Tuy nhiên, cú pháp cấp phát bộ nhớ cho mảng động một chiều là khác với cú pháp cấp phát bộ nhớ cho con trỏ thông thường:

<Tên con trỏ> = new <Kiểu con trỏ>[<Độ dài mảng>];

Trong đó:

Tên con trỏ: tên do người dùng đặt, tuân thủ theo quy tắc đặt tên biến của C++.

Kiểu con trỏ: Kiểu dữ liệu cơ bản của C++ hoặc là kiểu do người dùng tự định nghĩa

Độ dài mảng: số lượng các phần tử cần cấp phát bộ nhớ của mảng.

Ví dụ:

int *A = new int[5];

sẽ khai báo một mảng A có 5 phần tử kiểu int được cấp phát bộ nhớ động.

Lưu ý:

Khi cấp phát bộ nhớ cho con trỏ có khởi tạo thông thường, ta dùng dấu "()", khi cấp phát bộ nhớ cho mảng, ta dùng dấu "[]". Hai lệnh cấp phát sau là hoàn toàn khác nhau:

```
// Cấp phát bộ nhớ và khởi tạo cho một con trỏ int
        int *A = new int(5);
// Cấp phát bộ nhớ cho một mảng 5 phần tử kiểu int
        int *A = new int[5];
```

Giải phóng bộ nhớ của mảng động một chiều

Để giải phóng vùng nhớ đã được cấp phát cho một mảng động, ta dùng cú pháp sau:

```
    delete [] <tên con trỏ>;
```

Ví dụ:

```
// Cấp phát bộ nhớ cho một mảng có 5 phần tử kiểu int
int *A = new int[5];
// Giải phóng vùng nhớ do mảng A đang chiếm giữ.
delete [] A;
```

Chương trình 2.4

```
void InitArray(int *A, int length){
    A = new int[length];
for(int i=0; i<length; i++)
A[i] = 0;
    Return }
void DeleteArray(int *A){
    delete [] A;
return;}
```

1.4.3 Cấp phát bộ nhớ cho mảng động nhiều chiều

Cấp phát bộ nhớ cho mảng động nhiều chiều

Một mảng hai chiều là một con trỏ đến một con trỏ. Do vậy, ta phải cấp phát bộ nhớ theo từng chiều theo cú pháp cấp phát bộ nhớ cho mảng động một chiều.

Ví dụ:

```
int **A;
const int length = 10;
A = new int*[length]; // Cấp phát bộ nhớ cho số dòng của ma trận A
for(int i=0; i<length; i++)
// Cấp phát bộ nhớ cho các phần tử của mỗi dòng
    A[i] = new int[length];
```

sẽ cấp phát bộ nhớ cho một mảng động hai chiều, tương đương với một ma trận có kích thước 10*10.

Lưu ý:

• Trong lệnh cấp phát A = new int*[length], cần phải có dấu "*" để chỉ ra rằng cần cấp phát bộ nhớ cho một mảng các phần tử có kiểu là con trỏ int (int*), khác với kiểu int bình thường.

Giải phóng bộ nhớ của mảng động nhiều chiều

Ngược lại với khi cấp phát, ta phải giải phóng lần lượt bộ nhớ cho con trỏ tương ứng với cột và hàng của mảng động.

Ví dụ:

```
int **A;
...;      // cấp phát bộ nhớ
...
for(int i=0; i<length; i++)
delete [] A[i];   // Giải phóng bộ nhớ cho mỗi dòng
delete [] A;          // Giải phóng bộ nhớ cho mảng các dòng sẽ giải phóng bộ nhớ cho một mảng động hai chiều.
```

Chương trình 2.5

```
#include<stdio.h>
#include<conio.h>
/* Khai báo nguyên mẫu hàm */
void InitArray(int **A, int row, int colum);
void AddArray(int **A, int **B, int row, int colum);
void DisplayArray(int **A, int row, int colum);
void DeleteArray(int **A, int row);
```

```
void InitArray(int **A, int row, int colum){
    A = new int*[row];
    for(int i=0; i<row; i++){
        A[i] = new int[colum];
        for(int j=0; j<colum; j++){
            cout << "Phan tu [" << i << "," << j << "] = ";
            cin >> A[i][j];
} return;
}
void AddArray(int **A, int **B, int row, int colum){
    for(int i=0; i<row; i++)
    for(int j=0; j<colum; j++)
        A[i][j] += B[i][j];
return; }
void DisplayArray(int **A, int row, int colum){
    for(int i=0; i<row; i++){
        for(int j=0; j<colum; j++)
            cout << A[i][j] << " ";
        cout << endl; // Xuống dòng
    return;
}
void DeleteArray(int **A, int row){
    for(int i=0; i<row; i++)
        delete [] A[i];
    delete [] A;
return; }
void main(){
    clrscr();
    int **A, **B, row, colum;
    cout << "So dong: ";
    cin >> row;
    cout << "So cot: ";
cin >> colum;
/* Khởi tạo các ma trận */
    cout << "Khoi tao mang A:" << endl;
    InitArray(A, row, colum);
```

```
    cout << "Khoi tao mang B:" << endl;
    InitArray(B, row, colum);
// Cộng hai ma trận
    AddArray(A, B, row, colum);
    // Hiển thị ma trận kết quả
    cout << "Tong hai mang A va mang B:" << endl;
    DisplayArray(A, row, colum);
    // Giải phóng bộ nhớ
    DeleteArray(A, row);
    DeleteArray(B, row);
return;
}
```

TỔNG KẾT CHƯƠNG

Nội dung chương 1 đã trình bày các vấn đề liên quan đến việc khai báo và sử dụng con trỏ và mảng trong ngôn ngữ C++:

. Con trỏ là một kiểu biến đặc biệt, nó trỏ đến địa chỉ của một biến khác. Có hai cách truy nhập đến con trỏ là truy nhập đến địa chỉ hoặc truy nhập đến giá trị của địa chỉ mà con trỏ trỏ đến.

. Con trỏ có thể tham gia vào các phép toán như các biến thông thường bằng phép lấy giá trị.

. Một con trỏ có sự tương ứng với một mảng một chiều có cùng kiểu.

. Một ma trận hai chiều có thể thay thế bằng một mảng các con trỏ hoặc một con trỏ trỏ đến con trỏ.

. Một con trỏ có thể trỏ đến một hàm, khi đó, nó được dùng để gọi một hàm như là một tham số cho hàm khác.

. Một con trỏ cần phải trỏ vào một địa chỉ xác định hoặc phải được cấp phát bộ nhớ qua phép toán **new** và giải phóng bộ nhớ sau khi dùng bằng thao tác **delete**.

Bài tập chương 1

Câu hỏi về con trỏ

1. Toán tử gì được dùng để xác định địa chỉ của một biến?
2. Toán tử gì được dùng để xác định giá trị ở vị trí được trỏ bởi một con trỏ?
địa chỉ của một biến?
3. Con trỏ là gì?
4. Truy cập gián tiếp là gì?
5. Mảng được lưu trữ trong bộ nhớ như thế nào?
6. Chỉ ra hai cách để nhận được địa chỉ phần tử đầu tiên của mảng data[].
7. Nếu mảng được truyền đến một hàm, hai cách gì để nhận biết mảng kết thúc ở đâu?
8. Sáu toán tử gì có thể thực hiện với con trỏ?
9. Giả sử bạn có hai con trỏ. Nếu con trỏ đầu tiên trỏ đến phần tử thứ ba trong một mảng kiểu int, con trỏ thứ hai trỏ đến phần tử thứ tư. Việc trừ con trỏ thứ hai cho con
trỏ đầu cho kết quả gì?
10. Giả sử cost là một tên biến.
Làm thế nào để khai báo và khởi tạo một con trỏ có tên p_cost trỏ đến biến đó.
Làm thế nào để gán giá trị 100 cho biến cost bằng cách dùng cả truy cập trực tiếp và gián tiếp.
Làm thế nào để in giá trị của con trỏ p_cost và giá trị con trỏ p_cost trỏ đến.
11. Làm thế nào để gán địa chỉ của biến thực có tên là radius cho một viến con trỏ.
12. Hai cách để gán giá trị 100 cho phần tử thứ ba của mảng data[].
13. Viết một hàm có tên là sumarrays() có đối số là hai mảng, tính tổng giá trị cả hai mảng và trả về tổng đó. Viết chương trình minh họa.
14. Viết lệnh khai báo biến con trỏ, khai báo và khởi gán con trỏ trỏ tới biến, khai báo
và khởi gán con trỏ trỏ đến con trỏ.
15. Xét khai báo

```
float x;
float *px = &x;
float **ppx = &px;
```

Để gán 100 cho biến x, ta có thể viết như sau hay không?

```
*ppx = 100;
```

16. Viết một nguyên mẫu hàm với đối là một mảng con trỏ kiểu char và trả về void.

17. Con trỏ trỏ đến hàm là gì?

18. Viết một khai báo con trỏ trỏ đến hàm trả về kiểu char và có đối là một mảng con trỏ kiểu char.

19. Khai báo sau có gì sai:

```
char *ptr(char *x[]);
```

20. Giải thích các khai báo sau:

```
a. int *var1;
b. int var2;
c. int **var3;
```

21. Giải thích các khai báo sau:

```
a. int a[3][12];
b. int (*b)[12];
c. int *c[12];
```

22. Giải thích các khai báo sau:

```
a. char *z[10];
b. char *y(int field);
c. char (*x)(int field);
```

23. Viết một khai báo con trỏ trỏ đến hàm có đối kiểu nguyên và trả về biến kiểu float.

24 Viết một khai báo mảng con trỏ trỏ đến hàm có đối là chuỗi ký tự và trả về số nguyên.

25. Viết lệnh khai báo mảng 10 con trỏ kiểu char.

26. Có điểm gì sai trong đoạn mã sau:

```
int x[3][12];
int *ptr[12];

ptr = x;
```

27. Hãy khai báo biến kí tự ch và con trỏ kiểu kí tự pc trỏ vào biến ch. Viết ra các cách gán giá trị ‘A’ cho biến ch.
28. Cho mảng nguyên cost. Viết ra các cách gán giá trị 100 cho phần tử thứ 3 của mảng.
29. Cho p, q là các con trỏ cùng trỏ đến kí tự c. Đặt *p = *q + 1. Có thể khẳng định: *q = *p - 1 ?
30. Cho p, q là các con trỏ trỏ đến biến nguyên x = 5. Đặt *p = *q + 1; Hỏi *q ?
31. Cho p, q, r, s là các con trỏ trỏ đến biến nguyên x = 10. Đặt *q = *p + 1; *r = *q + 1; *s = *r + 1. Hỏi giá trị của biến x ?

32. Chọn câu đúng nhất trong các câu sau:

A: Địa chỉ của một biến là số thứ tự của byte đầu tiên máy dành cho biến đó.

B: Địa chỉ của một biến là một số nguyên.

C: Số học địa chỉ là các phép toán làm việc trên các số nguyên biểu diễn địa chỉ của biến

D: a và b đúng

33.Chọn câu sai trong các câu sau:

A: Các con trỏ có thể phân biệt nhau bởi kiểu của biến mà nó trỏ đến.

B: Hai con trỏ trỏ đến các kiểu khác nhau sẽ có kích thước khác nhau.

C: Một con trỏ kiểu void có thể được gán bởi con trỏ có kiểu bất kỳ (cần ép kiểu).

D: Hai con trỏ cùng trỏ đến kiểu cấu trúc có thể gán cho nhau.

34. Cho con trỏ p trỏ đến biến x kiểu float. Có thể khẳng định ?

A: p là một biến và *p cũng là một biến

B: p là một biến và *p là một giá trị hằng

C: Để sử dụng được p cần phải khai báo float *p; và gán *p = x;

D: Cũng có thể khai báo void *p; và gán (float)p = &x;

35. Cho khai báo float x, y, z, *px, *py; và các lệnh px = &x; py = &y; Có thể khẳng định ?

A: Nếu x = *px thì y = *py

B: Nếu x = y + z thì *px = y + z

C: Nếu *px = y + z thì *px = *py + z

D: a, b, c đúng

36. Cho khai báo float x, y, z, *px, *py; và các lệnh px = &x; py = &y; Có thể khẳng định ?

A: Nếu *px = x thì *py = y

B: Nếu *px = *py - z thì *px = y - z

C: Nếu *px = y - z thì x = y - z

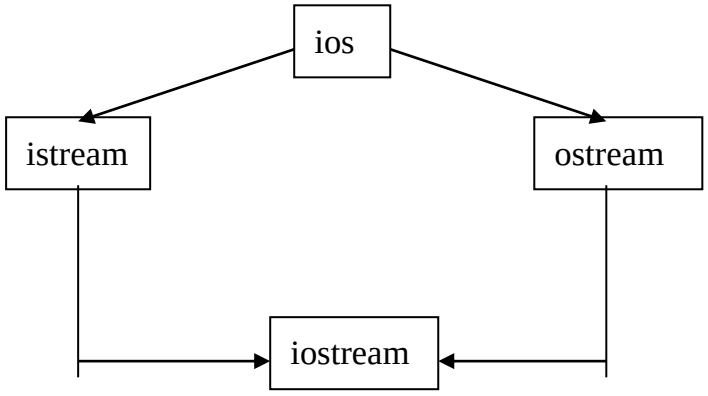
D: a, b, c đúng

37. Không dùng mảng, hãy nhập một dãy số nguyên và in ngược dãy ra màn hình.
38. Không dùng mảng, hãy nhập một dãy số nguyên và chỉ ra vị trí của số bé nhất, lớn nhất.
39. Không dùng mảng, hãy nhập một dãy số nguyên và in ra dãy đã được sắp xếp.
40. Không dùng mảng, hãy nhập một dãy kí tự. Thay mỗi kí tự ‘a’ trong dãy thành kí tự ‘b’ và in kết quả ra màn hình.

CHƯƠNG 2: CÁC DÒNG NHẬP XUẤT VÀ TỆP TIN

Nhập/xuất với cin/cout
Định dạng
In ra máy in
Làm việc với File
Nhập/xuất nhị phân

Trong C++ có sẵn một số lớp chuẩn chứa dữ liệu và các phương thức phục vụ cho các thao tác nhập/xuất dữ liệu của NSD, thường được gọi chung là ***stream*** (dòng). Trong số các lớp này, lớp có tên **ios** là lớp cơ sở, chứa các thuộc tính để định dạng việc nhập/xuất và kiểm tra lỗi. Mở rộng (kế thừa) lớp này có các lớp **istream, ostream** cung cấp thêm các toán tử nhập/xuất như >>, << và các hàm get, getline, read, ignore, put, write, flush … Một lớp rộng hơn có tên **iostream** là tổng hợp của 2 lớp trên. Bốn lớp nhập/xuất cơ bản này được khai báo trong các file tiêu đề có tên tương ứng (với đuôi *.h). Sơ đồ thừa kế của 4 lớp trên được thể hiện qua hình vẽ dưới đây.



Đối tượng của các lớp trên được gọi là các *dòng* dữ liệu. Một số đối tượng thuộc lớp iostream đã được khai báo sẵn (*chuẩn*) và được gắn với những thiết bị nhập/xuất cố định như các đối tượng **cin, cout, cerr, clog** gắn với bàn phím (cin) và màn hình (cout, cerr, clog). Điều này có nghĩa các toán tử >>, << và các hàm kể trên khi làm việc với các đối tượng này sẽ cho phép NSD nhập dữ liệu thông qua bàn phím hoặc xuất kết quả thông qua màn hình.

Để nhập/xuất thông qua các thiết bị khác (như máy in, file trên đĩa …), C++ cung cấp thêm các lớp **ifstream, ofstream, fstream** cho phép NSD khai báo các đối tượng mới gắn với thiết bị và từ đó nhập/xuất thông qua các thiết bị này.
Trong chương này, chúng ta sẽ xét các đối tượng chuẩn **cin, cout** và một số toán tử, hàm nhập xuất đặc trưng của lớp **iostream** cũng như cách tạo và sử dụng các đối tượng thuộc các lớp **ifstream, ofstream, fstream** để làm việc với các thiết bị như máy in và file trên đĩa.

2.1. NHẬP/XUẤT VỚI CIN/COUT

Như đã nhắc ở trên, **cin** là dòng dữ liệu nhập (đối tượng) thuộc lớp *istream.* Các thao tác trên đối tượng này gồm có các toán tử và hàm phục vụ nhập dữ liệu vào cho biến từ bàn phím.

2.1.1. Toán tử nhập >>

Toán tử này cho phép nhập dữ liệu từ một dòng Input_stream nào đó vào cho một danh sách các biến. Cú pháp chung như sau:

Input_stream >> biến1 >> biến2 >> …

trong đó Input_stream là đối tượng thuộc lớp istream. Trường hợp Input_stream là cin, câu lệnh nhập sẽ được viết:

cin >> biến1 >> biến2 >> …

câu lệnh này cho phép nhập dữ liệu từ bàn phím cho các biến. Các biến này có thể thuộc các kiểu chuẩn như : kiểu nguyên, thực, ký tự, xâu kí tự. Chú ý 2 đặc điểm quan trọng của câu lệnh trên.

- Lệnh sẽ bỏ qua không gán các dấu trắng (dấu cách <>, dấu Tab, dấu xuống dòng ↵) vào cho các biến (kể cả biến xâu kí tự).
- Khi NSD nhập vào dãy byte nhiều hơn cần thiết để gán cho các biến thì số byte còn lại và kể cả dấu xuống dòng ↵ sẽ nằm lại trong cin. Các byte này sẽ tự động gán cho các biến trong lần nhập sau mà không chờ NSD gõ thêm dữ liệu vào từ bàn phím. Do vậy câu lệnh

```
cin >> a >> b >> c;
```

cũng có thể được viết thành

```
cin >> a;
cin >> b;
cin >> c;
```

và chỉ cần nhập dữ liệu vào từ bàn phím một lần chung cho cả 3 lệnh (mỗi dữ liệu nhập cho mỗi biến phải cách nhau ít nhất một dấu trắng)

Ví dụ 1 : Nhập dữ liệu cho các biến

```
int a;
float b;
char c;
char *s;
cin >> a >> b >> c >> s;
```

giả sử NSD nhập vào dãy dữ liệu : <><>12<>34.517ABC<>12E<>D ↵
khi đó các biến sẽ được nhận những giá trị cụ thể sau:

a = 12
b = 34.517
c = 'A'
s = "BC"

trong cin sẽ còn lại dãy dữ liệu : <>12E<>D ↵.
Nếu trong đoạn chương trình tiếp theo có câu lệnh cin >> s; thì s sẽ được tự động gán giá trị "12E" mà không cần NSD nhập thêm dữ liệu vào cho cin.
Qua ví dụ trên một lần nữa ta nhắc lại đặc điểm của toán tử nhập >> là các biến chỉ lấy dữ liệu vừa đủ cho kiểu của biến (ví dụ biến c chỉ lấy một kí tự 'A', b lấy giá trị 34.517) hoặc cho đến khi gặp dấu trắng đầu tiên (ví dụ a lấy giá trị 12, s lấy giá trị "BC" dù trong cin vẫn còn dữ liệu). Từ đó ta thấy toán tử >> là không phù hợp khi nhập dữ liệu cho các xâu kí tự có chứa dấu cách. C++ giải quyết trường hợp này bằng một số hàm (phương thức) nhập khác thay cho toán tử >>.

2.1.2. Các hàm nhập kí tự và xâu kí tự

a. Nhập kí tự

- **cin.get()** : Hàm trả lại một kí tự (kể cả dấu cách, dấu ↵).. Ví dụ:

```
char ch;
ch = cin.get();
```

  - nếu nhập AB↵, ch nhận giá trị 'A', trong cin còn B↵.
  - nếu nhập A↵, ch nhận giá trị 'A', trong cin còn ↵.
  - nếu nhập ↵, ch nhận giá trị '↵', trong cin rỗng.

- **cin.get(ch)** : Hàm nhập kí tự cho ch và trả lại một tham chiếu tới cin. Do hàm trả lại tham chiếu tới cin nên có thể viết các phương thức nhập này liên tiếp trên một đối tượng cin. Ví dụ:

```
char c, d;
cin.get(c).get(d);
```

nếu nhập AB↵ thì c nhận giá trị 'A' và d nhận giá trị 'B'. Trong cin còn 'C↵'.

b. **Nhập xâu kí tự**

- **cin.get(s, n, fchar)** : Hàm nhập cho s dãy kí tự từ cin. Dãy được tính từ kí tự đầu tiên trong cin cho đến khi đã <u>đủ n – 1 kí tự</u> hoặc <u>gặp kí tự kết thúc</u> fchar. Kí tự kết thúc này được ngầm định là dấu xuống dòng nếu bị bỏ qua trong danh sách đối. Tức có thể viết câu lệnh trên dưới dạng **cin.get(s, n)** khi đó xâu s sẽ nhận dãy kí tự nhập cho đến khi đủ n-1 kí tự hoặc đến khi NSD kết thúc nhập (bằng dấu ↵).

<u>Chú ý</u> :

- Lệnh sẽ tự động gán dấu kết thúc xâu ('\0') vào cho xâu s sau khi nhập xong.
- Các lệnh có thể viết nối nhau, ví dụ: **cin.get(s1, n1).get(s2,n2);**
- Kí tự kết thúc fchar (hoặc ↵) vẫn nằm lại trong cin. Điều này có thể làm trôi các lệnh get() tiếp theo. Ví dụ:

```
struct Sinhvien {
    char *ht;                       // họ tên
    char *qq;                       // quê quán
};
void main()
{
    int i;
    for (i=1; i<=3; i++) {
            cout << "Nhap ho ten sv thu " << i; cin.get(sv[i].ht, 25);
            cout << "Nhap que quan sv thu "<< i; cin.get(sv[i].qq, 30);
    }
    …
}
```

Trong đoạn lệnh trên sau khi nhập họ tên của sinh viên thứ 1, do kí tự ↵ vẫn nằm trong bộ đệm nên khi nhập quê quán chương trình sẽ lấy kí tự ↵ này gán cho qq, do đó quê quán của sinh viên sẽ là xâu rỗng.

Để khắc phục tình trạng này chúng ta có thể sử dụng một trong các câu lệnh nhập kí tự để "nhấc" dấu enter còn "rơi vãi" ra khỏi bộ đệm. Có thể sử dụng các câu lệnh sau :

```
cin.get();                          // đọc một kí tự trong bộ đệm
cin.ignore(n);                      //đọc n kí tự trong bộ đệm (với n=1)
```

như vậy để đoạn chương trình trên hoạt động tốt ta có thể tổ chức lại như sau:

```
    void main()
    {
       int i;
       for (i=1; i<=3; i++) {
              cout << "Nhap ho ten sv thu " << i; cin.get(sv[i].ht, 25);
       cin.get();                          // nhấc 1 kí tự (enter)
          cout << "Nhap que quan sv thu "<< i; cin.get(sv[i].qq, 30);
          cin.get()                        // hoặc cin.ignore(1);
    }
…
}
```

- **cin.getline(s, n, fchar):** Phương thức này hoạt động hoàn toàn tương tự phương thức **cin.get(s, n, fchar),** tuy nhiên nó có thể khắc phục "lỗi enter" của câu lệnh trên. Cụ thể hàm sau khi gán nội dung nhập cho biến s sẽ xóa kí tự enter khỏi bộ đệm và do vậy NSD không cần phải sử dụng thêm các câu lệnh phụ trợ (cin.get(), cin.ignore(1)) để loại enter ra khỏi bộ đệm.
- **cin.ignore(n):** Phương thức này của đối tượng cin dùng để đọc và loại bỏ n kí tự còn trong bộ đệm (dòng nhập cin).

Chú ý: Toán tử nhập >> cũng giống các phương thức nhập kí tự và xâu kí tự ở chỗ cũng <u>để lại kí tự enter trong cin</u>. Do vậy, chúng ta nên sử dụng các phương thức cin.get(), cin.ignore(n) để loại bỏ kí tự enter trước khi thực hiện lệnh nhập kí tự và xâu kí tự khác.

Tương tự dòng nhập cin, **cout** là dòng dữ liệu xuất thuộc lớp *ostream.* Điều này có nghĩa dữ liệu làm việc với các thao tác xuất (in) sẽ đưa kết quả ra cout mà đã được mặc định là màn hình. Do đó ta có thể sử dụng toán tử xuất << và các phương thức xuất trong các lớp ios (lớp cơ sở) và ostream.

2.1.3. Toán tử xuất <<

Toán tử này cho phép xuất giá trị của dãy các biểu thức đến một dòng Output_stream nào đó với cú pháp chung như sau:

Output_stream << bt_1 << bt_2 << …

ở đây Output_stream là đối tượng thuộc lớp ostream. Trường hợp Output_stream là cout, câu lệnh xuất sẽ được viết:

cout << bt_1 << bt_2 << …

câu lệnh này cho phép in kết quả của các biểu thức ra màn hình. Kiểu dữ liệu của các biểu thức có thể là số nguyên, thực, kí tự hoặc xâu kí tự.

2.2. ĐỊNH DẠNG

Các giá trị in ra màn hình có thể được trình bày dưới nhiều dạng khác nhau thông qua các công cụ định dạng như các phương thức, các cờ và các bộ phận khác được khai báo sẵn trong các lớp ios và ostream.

2.2.1.Các phương thức định dạng

a. Chỉ định độ rộng cần in

cout.width(n) ;

Số cột trên màn hình để in một giá trị được ngầm định bằng với độ rộng thực (số chữ số, chữ cái và kí tự khác trong giá tị được in). Để đặt lại độ rộng màn hình dành cho giá trị cần in (thông thường lớn hơn độ rộng thực) ta có thể sử dụng phương thức trên.

Phương thức này cho phép các giá trị in ra màn hình với độ rộng n. Nếu n bé hơn độ rộng thực sự của giá trị thì máy sẽ in giá trị với số cột màn hình bằng với độ rộng thực. Nếu n lớn hơn độ rộng thực, máy sẽ in giá trị căn theo lề phải, và để trống các cột thừa phía trước giá trị được in. Phương thức này chỉ có tác dụng với giá trị cần in ngay sau nó. Ví dụ:

```
int a = 12; b = 345;          // độ rộng thực của a là 2, của b là 3
cout << a;                          // chiếm 2 cột màn hình
cout.width(7);                // đặt độ rộng giá trị in tiếp theo là 7
cout  << b;                   // b in trong 7 cột với 4 dấu cách đứng
trước
```

Kết quả in ra sẽ là: 12<><><><><>345

b. **Chỉ định kí tự chèn vào khoảng trống trước giá trị cần in**

cout.fill(ch) ;

Kí tự độn ngầm định là dấu cách, có nghĩa khi độ rộng của giá trị cần in bé hơn độ rộng chỉ định thì máy sẽ độn các dấu cách vào trước giá trị cần in cho đủ với độ rộng chỉ định. Có thể yêu cầu độn một kí tự ch bất kỳ thay cho dấu cách bằng phương thức trên. Ví dụ trong dãy lệnh trên, nếu ta thêm dòng lệnh cout.fill('*') trước khi in b chẳng hạn thì kết quả in ra sẽ là: 12****345.

Phương thức này có tác dụng với mọi câu lệnh in sau nó cho đến khi gặp một chỉ định mới.

c. **Chỉ định độ chính xác (số số lẻ thập phân) cần in**

cout.precision(n) ;

Phương thức này yêu cầu các số thực in ra sau đó sẽ có n chữ số lẻ. Các số thực trước khi in ra sẽ được làm tròn đến chữ số lẻ thứ n. Chỉ định này có tác dụng cho đến khi gặp một chỉ định mới. Ví dụ:

```
int a = 12.3; b = 345.678;          // độ rộng thực của a là 4, của b là 7
cout << a;                                  // chiếm 4 cột màn hình
cout.width(10);                             // đặt độ rộng giá trị in tiếp theo là
10
cout.precision(2);                  // đặt độ chính xác đến 2 số lẻ
cout  << b;                  // b in trong 10 cột với 4 dấu cách đứng trước
```

Kết quả in ra sẽ là: 12.3<><><><><>345.68

2.2.2. Các cờ định dạng

Một số các qui định về định dạng thường được gắn liền với các "cờ". Thông thường nếu định dạng này được sử dụng trong suốt quá trình chạy chương trình hoặc trong một khoảng thời gian dài trước khi gỡ bỏ thì ta "bật" các cờ tương ứng với nó. Các cờ được bật sẽ có tác dụng cho đến khi cờ với định dạng khác được bật. Các cờ được cho trong file tiêu đề iostream.h.

Để bật/tắt các cờ ta sử dụng các phương thức sau:

cout.setf(danh sách cờ); // Bật các cờ trong danh sách

cout.unsetf(danh sách cờ); // Tắt các cờ trong danh sách

Các cờ trong danh sách được viết cách nhau bởi phép toán hợp bit (|). Ví dụ lệnh cout.setf(ios::left | ios::scientific) sẽ bật các cờ ios::left và ios::scientific. Phương thức cout.unsetf(ios::right | ios::fixed) sẽ tắt các cờ ios::right | ios::fixed.

Dưới đây là danh sách các cờ cho trong iostream.h.

d. Nhóm căn lề

- **ios::left** : nếu bật thì giá trị in nằm bên trái vùng in ra (kí tự độn nằm sau).
- **ios::right** : giá trị in nằm bên phái vùng in ra (kí tự độn nằm trước), đây là trường hợp ngầm định nếu ta không sử dụng cờ cụ thể.
- **ios::internal** : giống cờ ios::right tuy nhiên dấu của giá trị in ra sẽ được in đầu tiên, sau đó mới đến kí tự độn và giá trị số.

Ví dụ:

```
int a = 12.3; b = –345.678;          // độ rộng thực của a là 4, của b là 8
cout << a;                                  // chiếm 4 cột màn hình
cout.width(10);                      // đặt độ rộng giá trị in tiếp theo là
10
cout.fill('*') ;                     // dấu * làm kí tự độn
cout.precision(2);                   // đặt độ chính xác đến 2 số lẻ
cout.setf(ios::left) ;               // bật cờ ios::left
cout << b;                                  // kết qủa: 12.3–345.68***
cout.setf(ios::right) ;              // bật cờ ios::right
cout << b;                                  // kết qủa: 12.3***–345.68
cout.setf(ios::internal) ;      // bật cờ ios::internal
cout << b;                                  // kết qủa: 12.3–***345.68
```

e. Nhóm định dạng số nguyên

- **ios::dec** : in số nguyên dưới dạng thập phân (ngầm định)
- **ios::oct** : in số nguyên dưới dạng cơ số 8
- **ios::hex** : in số nguyên dưới dạng cơ số 16

f. Nhóm định dạng số thực

- **ios::fixed** : in số thực dạng dấu phảy tĩnh (ngầm định)
- **ios::scientific** : in số thực dạng dấu phảy động
- **ios::showpoint** : in đủ n chữ số lẻ của phần thập phân, nếu tắt (ngầm định) thì không in các số 0 cuối của phần thập phân.

Ví dụ: giả sử độ chính xác được đặt với 3 số lẻ (bởi câu lệnh cout.precision(3))

- nếu fixed bật + showpoint bật :

123.2500	được in thành	123.250
123.2599	được in thành	123.260

123.2	được in thành	123.200

- nếu fixed bật + showpoint tắt :

123.2500	được in thành	123.25
123.2599	được in thành	123.26
123.2	được in thành	123.2

- nếu scientific bật + showpoint bật :

12.3	được in thành	1.230e+01
2.32599	được in thành	2.326e+00
324	được in thành	3.240e+02

- nếu scientific bật + showpoint tắt :

12.3	được in thành	1.23e+01
2.32599	được in thành	2.326e+00
324	được in thành	3.24e+02

g. **Nhóm định dạng hiển thị**

- **ios::showpos** : nếu tắt (ngầm định) thì không in dấu cộng (+) trước số dương. Nếu bật trước mỗi số dương sẽ in thêm dấu cộng.
- **ios::showbase** : nếu bật sẽ in số 0 trước các số nguyên hệ 8 và in 0x trước số hệ 16. Nếu tắt (ngầm định) sẽ không in 0 và 0x.
- **ios::uppercase** : nếu bật thì các kí tự biểu diễn số trong hệ 16 (A..F) sẽ viết hoa, nếu tắt (ngầm định) sẽ viết thường.

2.2.3. Các bộ và hàm định dạng

iostream.h cũng cung cấp một số bộ và hàm định dạng cho phép sử dụng tiện lợi hơn so với các cờ và các phương thức vì nó có thể được viết liên tiếp trên dòng lệnh xuất.

h. **Các bộ định dạng**

```
dec                     // tương tự ios::dec
oct                     // tương tự ios::dec
hex                     // tương tự ios::hex
endl                    // xuất kí tự xuống dòng ('\n')
flush                   // đẩy toàn bộ dữ liệu ra dòng xuất
```

Ví dụ :

```
cout.setf(ios::showbase) ;          // cho phép in các kí tự biểu thị cơ
```

số

```
cout.setf(ios::uppercase) ;                    // dưới dạng chữ viết hoa
int a = 171; int b = 32 ;
cout << hex << a << endl << b ;                // in 0xAB và 0x20
```

i. **Các hàm định dạng (#include <iomanip.h>)**

```
setw(n)              // tương tự cout.width(n)
setprecision(n)      // tương tự cout.precision(n)
setfill(c)                  // tương tự cout.fill(c)
setiosflags(l)       // tương tự cout.setf(l)
resetiosflags(l)     // tương tự cout.unsetf(l)
```

2.3. IN RA MÁY IN

Như trong phần đầu chương đã trình bày, để làm việc với các thiết bị khác với màn hình và đĩa … chúng ta cần tạo ra các đối tượng (thuộc các lớp ifstream, ofstream và fstream) tức các dòng tin bằng các hàm tạo của lớp và gắn chúng với thiết bị bằng câu lệnh:

ofstream Tên_dòng(thiết bị) ;

Ví dụ để tạo một đối tượng mang tên Mayin và gắn với máy in, chúng ta dùng lệnh:

ofstream Mayin(4) ;

trong đó 4 là số hiệu của máy in.

Khi đó mọi câu lệnh dùng toán tử xuất << và cho ra Mayin sẽ đưa dữ liệu cần in vào một bộ đệm mặc định trong bộ nhớ. Nếu bộ đệm đầy, một số thông tin đưa vào trước sẽ tự động chuyển ra máy in. Để chủ động đưa tất cả dữ liệu còn lại trong bộ đệm ra máy in chúng ta cần sử dụng bộ định dạng flush (Mayin << flush << …) hoặc phương thức flush (Mayin.flush();). Ví dụ:

Sau khi đã khai báo một đối tượng mang tên Mayin bằng câu lệnh như trên Để in chu vi và diện tísch hình chữ nhật có cạnh cd và cr ta có thể viết:

```
Mayin << "Diện tích HCN = " << cd * cr << endl;
Mayin << "Chu vi HCN = " << 2*(cd + cr) << endl;
Mayin.flush();
```

hoặc :

```
Mayin << "Diện tích HCN = " << cd * cr << endl;
Mayin << "Chu vi HCN = " << 2*(cd + cr) << endl << flush;
```

khi chương trình kết thúc mọi dữ liệu còn lại trong các đối tượng sẽ được tự động chuyển ra thiết bị gắn với nó. Ví dụ máy in sẽ in tất cả mọi dữ liệu còn sót lại trong Mayin khi chương trình kết thúc.

2.4. LÀM VIỆC VỚI FILE

Làm việc với một file trên đĩa cũng được quan niệm như làm việc với các thiết bị khác của máy tính (ví dụ như làm việc với máy in với đối tượng Mayin trong phần trên hoặc làm việc với màn hình với đối tượng chuẩn cout). Các đối tượng này được khai báo thuộc lớp ifstream hay ofstream tùy thuộc ta muốn sử dụng file để đọc hay ghi.

Như vậy, để sử dụng một file dữ liệu đầu tiên chúng ta cần tạo đối tượng và gắn cho file này. Để tạo đối tượng có thể sử dụng các hàm tạo có sẵn trong hai lớp ifstream và ofstream. Đối tượng sẽ được gắn với tên file cụ thể trên đĩa ngay trong quá trình tạo đối tượng (tạo đối tượng với tham số là tên file) hoặc cũng có thể được gắn với tên file sau này bằng câu lệnh mở file. Sau khi đã gắn một đối tượng với file trên đĩa, có thể sử dụng đối tượng như đối với Mayin hoặc cin, cout. Điều này có nghĩa trong các câu lệnh in ra màn hình chỉ cần thay từ khóa cout bởi tên đối tượng mọi dữ liệu cần in trong câu lệnh sẽ được ghi lên file mà đối tượng đại diện. Cũng tương tự nếu thay cin bởi tên đối tượng, dữ liệu sẽ được đọc vào từ file thay cho từ bàn phím. Để tạo đối tượng dùng cho việc ghi ta khai báo chúng với lớp *ofstream* còn để dùng cho việc đọc ta khai báo chúng với lớp *ifstream*.

2.4.1. Tạo đối tượng gắn với file

Mỗi lớp ifstream và ofstream cung cấp 4 phương thức để tạo file. Ở đây chúng tôi chỉ trình bày 2 cách (2 phương thức) hay dùng.

+ Cách 1: **<Lớp> đối_tượng;**
  đối_tượng.open(tên_file, chế_độ);

Lớp là một trong hai lớp ifstream và ofstream. Đối tượng là tên do NSD tự đặt. Chế độ là cách thức làm việc với file (xem dưới). Cách này cho phép tạo trước một đối tượng chưa gắn với file cụ thể nào. Sau đó dùng tiếp phương thức open để đồng thời mở file và gắn với đối tượng vừa tạo.

Ví dụ:

```
ifstream f;              // tạo đối tượng có tên f để đọc hoặc
ofstream f;                    // tạo đối tượng có tên f để ghi
f.open("Baitap");        // mở file Baitap và gắn với f
```

+ Cách 2: **<Lớp> đối_tượng(tên_file, chế_độ)**

Cách này cho phép đồng thời mở file cụ thể và gắn file với tên đối tượng trong câu lệnh.
Ví dụ:

```
ifstream f("Baitap");       // mở file Baitap gắn với đối tượng f để
ofstream f("Baitap);                // đọc hoặc ghi.
```

Sau khi mở file và gắn với đối tượng f, mọi thao tác trên f cũng chính là làm việc với file Baitap.
Trong các câu lệnh trên có các chế độ để qui định cách thức làm việc của file. Các chế độ này gồm có:

- ios::binary : quan niệm file theo kiểu nhị phân. Ngầm định là kiểu văn bản.
- ios::in : file để đọc (ngầm định với đối tượng trong ifstream).
- ios::out : file để ghi (ngầm định với đối tượng trong ofstream), nếu file đã có trên đĩa thì nội dung của nó sẽ bị ghi đè (bị xóa).
- ios::app : bổ sung vào cuối file
- ios::trunc : xóa nội dung file đã có
- ios::ate : chuyển con trỏ đến cuối file
- ios::nocreate : không làm gì nếu file chưa có
- ios::replace : không làm gì nếu file đã có

có thể chỉ định cùng lúc nhiều chế độ bằng cách ghi chúng liên tiếp nhau với toán tử hợp bit |. Ví dụ để mở file bài tập như một file nhị phân và ghi tiếp theo vào cuối file ta dùng câu lệnh:

```
ofstream f("Baitap", ios::binary | ios::app);
```

2.4.2 Đóng file và giải phóng đối tượng

Để đóng file được đại diện bởi f, sử dụng phương thức close như sau:

đối_tượng.close();

Sau khi đóng file (và giải phóng mối liên kết giữa đối tượng và file) có thể dùng đối tượng để gắn và làm việc với file khác bằng phương thức open như trên.

Ví dụ 2 : Đọc một dãy số từ bàn phím và ghi lên file. File được xem như file văn bản (ngầm định), các số được ghi cách nhau 1 dấu cách.

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

```
void main()
{
  ofstream f;                                  // khai báo (tạo) đối tượng f
  int x;
  f.open("DAYSO");                    // mở file DAYSO và gắn với f
  for (int i = 1; i<=10; i++) {
          cin >> x;
          f << x << ' ';
  }
  f.close();
}
```

Ví dụ 3 : Chương trình sau nhập danh sách sinh viên, ghi vào file 1, đọc ra mảng, sắp xếp theo tuổi và in ra file 2. Dòng đầu tiên trong file ghi số sinh viên, các dòng tiếp theo ghi thông tin của sinh viên gồm họ tên với độ rộng 24 kí tự, tuổi với độ rộng 4 kí tự và điểm với độ rộng 8 kí tự.

```
#include <iostream.h>
#include <iomanip.h>
#include <fstream.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <ctype.h>
struct Sv {
  char *hoten;
  int tuoi;
  double diem;
};
class Sinhvien {
  int sosv ;
  Sv *sv;
public:
  Sinhvien() {
          sosv = 0;
          sv = NULL;
  }
```

```
  void nhap();
  void sapxep();
  void ghifile(char *fname);
};

void Sinhvien::nhap()
{
  cout << "\nSố sinh viên: "; cin >> sosv;
  int n = sosv;
  sv = new Sinhvien[n+1];                    // Bỏ phần tử thứ 0
  for (int i = 1; i <= n; i++) {
          cout << "\nNhập sinh viên thứ: " << i << endl;
          cout << "\nHọ tên: "; cin.ignore(); cin.getline(sv[i].hoten);
          cout << "\nTuổi: "; cin >> sv[i].tuoi;
          cout << "\nĐiểm: "; cin >> sv[i].diem;
  }
}

void Sinhvien::ghi(char fname)
{
  ofstream f(fname) ;
  f << sosv;
  f << setprecision(1) << setiosflags(ios::showpoint) ;
  for (int i=1; i<=sosv; i++) {
          f << endl << setw(24) << sv[i].hoten << setw(4) << tuoi;
          f << setw(8) << sv[i].diem;
  }
  f.close();
}

void Sinhvien::doc(char fname)
{
  ifstream f(fname) ;
  f >> sosv;
  for (int i=1; i<=sosv; i++) {
```

```
            f.getline(sv[i].hoten, 25);
               f >> sv[i].tuoi >> sv[i].diem;
  }
  f.close();
}

void Sinhvien::sapxep()
{
  int n = sosv;
  for (int i = 1; i < n; i++) {
  for (int j = j+1; j <= n; j++) {
  if (sv[i].tuoi > sv[j].tuoi) {
          Sinhvien t = sv[i]; sv[i] = sv[j]; sv[j] = t;
  }
}

void main() {
  clrscr();
  Sinhvien x ;
      x.nhap(); x.ghi("DSSV1");
      x.doc("DSSV1"); x.sapxep(); x.ghi("DSSV2");
      cout << "Đã xong";
      getch();
}
```

2.4.3. Kiểm tra sự tồn tại của file, kiểm tra hết file

Việc mở một file chưa có để đọc sẽ gây nên lỗi và làm dừng chương trình. Khi xảy ra lỗi mở file, giá trị trả lại của phương thức bad là một số khác 0. Do vậy có thể sử dụng phương thức này để kiểm tra một file đã có trên đĩa hay chưa. Ví dụ:

```
      ifstream f("Bai tap");
      if (f.bad()) {
            cout << "file Baitap chưa có";
            exit(1);
  }
```

Khi đọc hoặc ghi, con trỏ file sẽ chuyển dần về cuối file. Khi con trỏ ở cuối file, phương thức eof() sẽ trả lại giá trị khác không. Do đó có thể sử dụng phương thức này để kiểm tra đã hết file hay chưa.
Chương trình sau cho phép tính độ dài của file Baitap. File cần được mở theo kiểu nhị phân.

```
 #include <iostream.h>
#include <fstream.h>
#include <stdlib.h>
#include <conio.h>
void main()
{
  clrscr();
  long dodai = 0;
  char ch;
  ifstream f("Baitap", ios::in | ios::binary) ;
  if (f.bad()) {
          cout << "File Baitap không có";
          exit(1);
  }
  while (!f.eof()) {
             f.get(ch));
             dodai++;
  }
  cout << "Độ dài của file = " << dodai;
  getch();
}
```

2.4.4. Đọc ghi đồng thời trên file

Để đọc ghi đồng thời, file phải được gắn với đối tượng của lớp fstream là lớp thừa kế của 2 lớp ifstream và ofstream. Khi đó chế độ phải được bao gồm chỉ định ios::in | ios::out. Ví dụ:

```
  fstream f("Data", ios::in | ios::out) ;
```

hoặc

```
  fstream f ;
      f.open("Data", ios::in | ios::out) ;
```

2.4.5. Di chuyển con trỏ file

Các phương thức sau cho phép làm việc trên đối tượng của dòng xuất (ofstream).

- **đối_tượng.seekp(n)** ; Di chuyển con trỏ đến byte thứ n (các byte được tính từ 0)
- **đối_tượng.seekp(n, vị trí xuất phát) ;** Di chuyển đi n byte (có thể âm hoặc dương) từ vị trí xuất phát. Vị trí xuất phát gồm:
  - ios::beg : từ đầu file
  - ios::end : từ cuối file
  - ios::cur : từ vị trí hiện tại của con trỏ.
- **đối_tượng.tellp(n) ;** Cho biết vị trí hiện tại của con trỏ.

Để làm việc với dòng nhập tên các phương thức trên được thay tương ứng bởi các tên : **seekg** và **tellg**. Đối với các dòng nhập lẫn xuất có thể sử dụng được cả 6 phương thức trên.

Ví dụ sau tính độ dài tệp đơn giản hơn ví dụ ở trên.

```
fstream f("Baitap");
f.seekg(0, ios::end);
cout << "Độ dài bằng = " << f.tellg();
```

Ví dụ 4 : Chương trình nhập và in danh sách sinh viên trên ghi/đọc đồng thời.

```
#include <iostream.h>
#include <iomanip.h>
#include <fstream.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <ctype.h>
void main() {
  int stt ;
       char *hoten, *fname, traloi;
  int tuoi;
  float diem;
  fstream f;
  cout << "Nhập tên file: "; cin >> fname;
  f.open(fname, ios::in | ios::out | ios::noreplace) ;
  if (f.bad()) {
```

```
            cout << "Tệp đã có. Ghi đè (C/K)?" ;
            cin.get(traloi) ;
            if (toupper(traloi) == 'C') {
                  f.close() ;
                  f.open(fname, ios::in | ios::out | ios::trunc) ;
            } else exit(1);
}
stt = 0;
f << setprecision(1) << setiosflags(ios::showpoint) ;
// nhập danh sách
      while (1) {
            stt++;
            cout << "\nNhập sinh viên thứ " << stt ;
            cout << "\nHọ tên: "; cin.ignore() ; cin.getline(hoten, 25);
            if (hoten[0] = 0) break;
            cout << "\nTuổi: "; cin >> tuoi;
            cout << "\nĐiểm: "; cin >> diem;
            f << setw(24) << hoten << endl;
            f << setw(4) << tuoi << set(8) << diem ;
}
// in danh sách
      f.seekg(0) ;                              // quay về đầu danh sách
      stt = 0;
      clrscr();
      cout << "Danh sách sinh viên đã nhập\n" ;
      cout << setprecision(1) << setiosflags(ios::showpoint) ;
      while (1) {
            f.getline(hoten,25);
            if (f.eof()) break;
            stt++;
            f >> tuoi >> diem;
            f.ignore();
            cout << "\nSinh viên thứ " << stt ;
            cout << "\nHọ tên: " << hoten;
            cout << "\nTuổi: " << setw(4) << tuoi;
            cout << "\nĐiểm: " << setw(8) << diem;
```

```
  }
  f.close();
  getch();
}
```

2.5. NHẬP/XUẤT NHỊ PHÂN

2.5.1. Khái niệm về 2 loại file: văn bản và nhị phân

j. **File văn bản**

Trong file văn bản mỗi byte được xem là một kí tự. Tuy nhiên nếu 2 byte 10 (LF), 13 (CR) đi liền nhau thì được xem là một kí tự và nó là kí tự xuống dòng. Như vậy file văn bản là một tập hợp các dòng kí tự với kí tự xuống dòng có mã là 10. Kí tự có mã 26 được xem là kí tự kết thúc file.

k. **File nhị phân**

Thông tin lưu trong file được xem như dãy byte bình thường. Mã kết thúc file được chọn là -1, được định nghĩa là EOF trong stdio.h. Các thao tác trên file nhị phân thường đọc ghi từng byte một, không quan tâm ý nghĩa của byte.

Một số các thao tác nhập/xuất sẽ có hiệu quả khác nhau khi mở file dưới các dạng khác nhau.

Ví dụ 1 : giả sử ch = 10, khi đó f << ch sẽ ghi 2 byte 10,13 lên file văn bản f, trong khi đó lệnh này chỉ khi 1 byte 10 lên file nhị phân.

Ngược lại, nếu f la file văn bản thì f.getc(ch) sẽ trả về chỉ 1 byte 10 khi đọc được 2 byte 10, 13 liên tiếp nhau.

Một file luôn ngầm định dưới dạng văn bản, do vậy để chỉ định file là nhị phân ta cần sử dụng cờ ios::binary.

2.5.2.Đọc, ghi kí tự

- **put(c);** // ghi kí tự ra file
- **get(c);** // đọc kí tự từ file

Ví dụ 2 : Sao chép file 1 sang file 2. Cần sao chép và ghi từng byte một do vậy để chính xác ta sẽ mở các file dưới dạng nhị phân.

```
#include <iostream.h>
#include <fstream.h>
#include <stdlib.h>
#include <conio.h>
void main()
```

```
{
  clrscr();
  fstream fnguon("DATA1", ios::in | ios::binary);
  fstream fdich("DATA2", ios::out | ios::binary);
  char ch;
  while (!fnguon.eof()) {
        fnguon.get(ch);
        fdich.put(ch);
  }
  fnguon.close();
  fdich.close();
}
```

2.5.3. Đọc, ghi dãy kí tự

- **write(char *buf, int n);** // ghi n kí tự trong buf ra dòng xuất
- **read(char *buf, int n);** // nhập n kí tự từ buf vào dòng nhập
- **gcount();** // cho biết số kí tự read đọc được

Ví dụ 3 : Chương trình sao chép file ở trên có thể sử dụng các phương thức mới này như sau:

```
#include <iostream.h>
#include <fstream.h>
#include <stdlib.h>
#include <conio.h>
void main()
{
  clrscr();
  fstream fnguon("DATA1", ios::in | ios::binary);
  fstream fdich("DATA2", ios::out | ios::binary);
  char buf[2000] ;
  int n = 2000;
  while (n) {
        fnguon.read(buf, 2000);
        n = fnguon.gcount();
        fdich.write(buf, n);
  }
  fnguon.close();
```

```
  fdich.close();
}
```

2.5.4. Đọc ghi đồng thời

```
#include <iostream.h>
#include <iomanip.h>
#include <fstream.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <string.h>
#include <ctype.h>

struct Sv {
  char *hoten;
  int tuoi;
  double diem;
};
class Sinhvien {
  int sosv;
       Sv x;
  char fname[30];
  static int size;
public:
  Sinhvien(char *fn);
  void tao();
  void bosung();
  void xemsua();
};

int Sinhvien::size = sizeof(Sv);
Sinhvien::Sinhvien(char *fn)
{
  strcpy(fname, fn) ;
  fstream f;
  f.open(fname, ios::in | ios::ate | ios::binary);
```

```
  if (!f.good) sosv = 0;
  else {
          sosv = f.tellg() / size;
  }
}
void Sinhvien::tao()
{
  fstream f;
  f.open(fname, ios::out | ios::noreplace | ios::binary);
  if (!f.good()) {
          cout << "danh sach da co. Co tao lai (C/K) ?";
          char traloi = getch();
          if (toupper(traloi) == 'C') return;
          else {
                  f.close() ;
                  f.open(fname, ios::out | ios::trunc | ios::binary);
          }
  }
  sosv = 0
       while (1) {
          cout << "\nSinh viên thứ: " << sosv+1;
          cout << "\nHọ tên: "; cin.ignore(); cin.getline(x.hoten);
          if (x.hoten[0] == 0) break;
          cout << "\nTuổi: "; cin >> x.tuoi;
          cout << "\nĐiểm: "; cin >> x.diem;
          f.write((char*)(&x), size);
          sosv++;
  }
  f.close();
}

void Sinhvien::bosung()
{
  fstream f;
  f.open(fname, ios::out | ios::app | ios::binary);
  if (!f.good()) {
```

```
            cout << "danh sach chua co. Tao moi (C/K) ?";
            char traloi = getch();
            if (toupper(traloi) == 'C') return;
            else {
                  f.close() ;
                  f.open(fname, ios::out | ios::binary);
            }
  }
  int stt = 0
        while (1) {
            cout << "\nBổ sung sinh viên thứ: " << stt+1;
            cout << "\nHọ tên: "; cin.ignore(); cin.getline(x.hoten);
            if (x.hoten[0] == 0) break;
            cout << "\nTuổi: "; cin >> x.tuoi;
            cout << "\nĐiểm: "; cin >> x.diem;
            f.write((char*)(&x), size);
            stt++;
  }
  sosv += stt;
  f.close();
}

void Sinhvien::xemsua()
{
  fstream f;
  int ch;
  f.open(fname, ios::out | ios::app | ios::binary);
  if (!f.good()) {
            cout << "danh sach chua co";
            getch(); return;
  }
  cout << "\nDanh sách sinh viên" << endl;
  int stt ;
        while (1) {
            cout << "\nCần xem (sua) sinh viên thứ (0: dừng): " ;
            cin >> stt;
```

```
            if (stt < 1 || stt > sosv) break;
            f.seekg((stt-1) * size, ios::beg);
            f.read((char*)(&x), size);
            cout << "\nHọ tên: " << x.hoten;
            cout << "\nTuổi: " << x.tuoi;
            cout << "\nĐiểm: " << x.diem;
            cout << "Có sửa không (C/K) ?";
            cin >> traloi;
            if (toupper(traloi) == 'C') {
                  f.seekg(-size, ios::cur);
                  cout << "\nHọ tên: "; cin.ignore(); cin.getline(x.hoten);
                  cout << "\nTuổi: "; cin >> x.tuoi;
                  cout << "\nĐiểm: "; cin >> x.diem;
                  f.write((char*)(&x), size);
            }
  }
  f.close();
}
void main()
{
  int chon;
  Sinhvien SV("DSSV") ;
  while (1) {
            clrscr();
            cout << "\n1: Tạo danh sách sinh viên";
            cout << "\n2: Bổ sung danh sách";
            cout << "\n3: Xem – sửa danh sách";
            cout << "\n0: Kết thúc";
            chon = getch();
            chon = chon – 48;
            clrscr();
            if (chon == 1) SV.tao();
            else if (chon == 2) SV.bosung();
            else if (chon == 3) SV.xemsua();
            else break;
        }
}
```

BÀI TẬP

1. Viết chương trình đếm số dòng của một file văn bản.
2. Viết chương trình đọc in từng kí tự của file văn bản ra màn hình, mỗi màn hình 20 dòng.
3. Viết chương trình tìm xâu dài nhất trong một file văn bản.
4. Viết chương trình ghép một file văn bản thứ hai vào file văn bản thứ nhất, trong đó tất cả chữ cái của file văn bản thứ nhất phải đổi thành chữ in hoa.
5. Viết chương trình in nội dung file ra màn hình và cho biết tổng số chữ cái, tổng số chữ số đã xuất hiện trong file.
6. Cho 2 file số thực (đã được sắp tăng dần). In ra màn hình dãy số xếp tăng dần của cả 2 file. (Cần tạo cả 2 file dữ liệu này bằng Editor của C++).
7. Viết hàm nhập 10 số thực từ bàn phím vào file INPUT.DAT. Viết hàm đọc các số thực từ file trên và in tổng bình phương của chúng ra màn hình.
8. Viết hàm nhập 10 số nguyên từ bàn phím vào file văn bản tên INPUT.DAT. Viết hàm đọc các số nguyên từ file trên và ghi những số chẵn vào file EVEN.DAT còn các số lẻ vào file ODD.DAT.
9. Nhập bằng chương trình 2 ma trận số nguyên vào 2 file văn bản. Hãy tạo file văn bản thứ 3 chứa nội dung của ma trận tích của 2 ma trận trên.
10. Tổ chức quản lý file sinh viên (Họ tên, ngày sinh, giới tính, điểm) với các chức năng : Nhập, xem, xóa, sửa, tính điểm trung chung.
11. Thông tin về một nhân viên trong cơ quan bao gồm : họ và tên, nghề nghiệp, số điện thoại, địa chỉ nhà riêng. Viết hàm nhập từ bàn phím thông tin của 7 nhân viên và ghi vào file INPUT.DAT. Viết hàm tìm trong file INPUT.DAT và in ra thông tin của 1 nhân viên theo số điện thoại được nhập từ bàn phím.

CHƯƠNG 3
DỮ LIỆU KIỂU CẤU TRÚC VÀ HỢP

Nội dung chính của chương này nhằm làm rõ các phương pháp, kỹ thuật biểu diễn, phép toán và ứng dụng của các cấu trúc dữ liệu trừu tượng. Cần đặc biệt lưu ý, ứng dụng các cấu trúc dữ liệu này không chỉ riêng cho lập trình ứng dụng mà còn ứng dụng trong biểu diễn bộ nhớ để giải quyết những vấn đề bên trong của các hệ điều hành. Các kỹ thuật lập trình trên cấu trúc dữ liệu trừu tượng được đề cập ở đây bao gồm:

Kiểu cấu trúc
Cấu trúc tự trỏ và danh sách liên kết
Kiểu hợp
Kiểu liệt kê

Để lưu trữ các giá trị gồm nhiều thành phần dữ liệu giống nhau ta có kiểu biến mảng. Thực tế rất nhiều dữ liệu là tập các kiểu dữ liệu khác nhau tập hợp lại, để quản lý dữ liệu kiểu này C++ đưa ra kiểu dữ liệu cấu trúc. Một ví dụ của dữ liệu kiểu cấu trúc là một bảng lý lịch trong đó mỗi nhân sự được lưu trong một bảng gồm nhiều kiểu dữ liệu khác nhau như họ tên, tuổi, giới tính, mức lương …

3.1. KIỂU CẤU TRÚC

3.1.1. Khai báo, khởi tạo

Để tạo ra một kiểu cấu trúc NSD cần phải khai báo tên của kiểu (là một tên gọi do NSD tự đặt), tên cùng với các thành phần dữ liệu có trong kiểu cấu trúc này. Một kiểu cấu trúc được khai báo theo mẫu sau:

```
struct <tên kiểu>
    {
        các thành phần ;
    } <danh sách biến>;
```

- Mỗi thành phần giống như một biến riêng của kiểu, nó gồm kiểu và tên thành phần. Một thành phần cũng còn được gọi là trường.
- Phần tên của kiểu cấu trúc và phần danh sách biến có thể có hoặc không. Tuy nhiên trong khai báo kí tự kết thúc cuối cùng phải là dấu chấm phẩy

(;).

- Các kiểu cấu trúc được phép khai báo lồng nhau, nghĩa là một thành phần của kiểu cấu trúc có thể lại là một trường có kiểu cấu trúc.
- Một biến có kiểu cấu trúc sẽ được phân bố bộ nhớ sao cho các thực hiện của nó được sắp liên tục theo thứ tự xuất hiện trong khai báo.
- Khai báo biến kiểu cấu trúc cũng giống như khai báo các biến kiểu cơ sở dưới dạng:

struct <tên cấu trúc> <danh sách biến> ; // kiểu cũ trong C

hoặc

<tên cấu trúc> <danh sách biến> ; // trong C++

Các biến được khai báo cũng có thể đi kèm khởi tạo:

<tên cấu trúc> biến = { giá trị khởi tạo } ;

Ví dụ:

- Khai báo kiểu cấu trúc chứa phân số gồm 2 thành phần nguyên chứa tử số và mẫu số.

```
struct Phanso
{
   int tu ;
   int mau ;
} ;
```

hoặc:

```
struct Phanso { int tu, mau ; }
```

- Kiểu ngày tháng gồm 3 thành phần nguyên chứa ngày, tháng, năm.

```
struct Ngaythang {
      int ng ;
      int th ;
      int nam ;
} holiday = { 1,5,2000 } ;
```

một biến holiday cũng được khai báo kèm cùng kiểu này và được khởi tạo bởi bộ số 1. 5. 2000. Các giá trị khởi tạo này lần lượt gán cho các thành phần theo đúng thứ tự trong khai báo, tức ng = 1, th = 5 và nam = 2000.

- Kiểu Lop dùng chứa thông tin về một lớp học gồm tên lớp và sĩ số sinh viên. Các biến kiểu Lop được khai báo là daihoc và caodang, trong đó daihoc được khởi tạo bởi bộ giá trị {"K41T", 60} với ý nghĩa tên lớp đại

học là K41T và sĩ số là 60 sinh viên.

```
struct Lop {
        char tenlop[10],
        int soluong;
} ;
struct Lop daihoc = {"K41T", 60}, caodang ;
```

hoặc:

```
Lop daihoc = {"K41T", 60}, caodang ;
```

- Kiểu Sinhvien gồm có các trường hoten để lưu trữ họ và tên sinh viên, ns lưu trữ ngày sinh, gt lưu trữ giới tính dưới dạng số (qui ước 1: nam, 2: nữ) và cuối cùng trường diem lưu trữ điểm thi của sinh viên. Các trường trên đều có kiểu khác nhau.

```
struct Sinhvien {
        char hoten[25] ;
        Ngaythang ns;
        int gt;
        float diem ;
} x, *p, K41T[60];
Sinhvien y = {"NVA", {1,1,1980}, 1} ;
```

Khai báo cùng với cấu trúc Sinhvien có các biến x, con trỏ p và mảng K41T với 60 phần tử kiểu Sinhvien. Một biến y được khai báo thêm và kèm theo khởi tạo giá trị {"NVA", {1,1,1980}, 1}, tức họ tên của sinh viên y là "NVA", ngày sinh là 1/1/1980, giới tính nam và điểm thi để trống. Đây là kiểu khởi tạo thiếu giá trị, giống như khởi tạo mảng, các giá trị để trống phải nằm ở cuối bộ giá trị khởi tạo (tức các thành phần bỏ khởi tạo không được nằm xen kẽ giữa những thành phần được khởi tạo).Ví dụ này còn minh hoạ cho các cấu trúc lồng nhau, cụ thể trong kiểu cấu trúc Sinhvien có một thành phần cũng kiểu cấu trúc là thành phần ns.

3.1.2.Truy nhập các thành phần kiểu cấu trúc

Để truy nhập vào các thành phần kiểu cấu trúc ta sử dụng cú pháp: tên biến.tên thành phần hoặc tên biến → tên thành phần đối với biến con trỏ cấu trúc. Cụ thể:

- Đối với biến thường: **tên biến.tên thành phần**

Ví dụ:

```
struct Lop {
        char tenlop[10];
        int siso;
} ;
Lop daihoc = "K41T", caodang ;
caodang.tenlop = daihoc.tenlop ;      // gán tên lớp cđẳng bởi tên lớp đhọc
caodang.siso++;                       // tăng sĩ số lớp caodang lên 1
```

- Đối với biến con trỏ: **tên biến → tên thành phần**

Ví dụ:

```
struct Sinhvien {
        char hoten[25] ;
        Ngaythang ns;
        int gt;
        float diem ;
} x, *p, K41T[60];
Sinhvien y = {"NVA", {1,1,1980}, 1} ;
y.diem = 5.5 ;                                  // gán điểm thi cho sinh viên y
p = new Sinhvien ;                              // cấp bộ nhớ chứa 1 sinh viên
strcpy(p→hoten, y.hoten) ;                      // gán họ tên của y cho sv trỏ bởi p
cout << p→hoten << y.hoten;           // in hoten của y và con trỏ p
```

- Đối với biến mảng: truy nhập thành phần mảng rồi đến thành phần cấu trúc.

Ví dụ:

```
strcpy(K41T[1].hoten, p→hoten) ;    // gán họ tên cho sv đầu tiên của lớp
K41T[1].diem = 7.0 ;                            // gán điểm cho sv đầu tiên
```

- Đối với cấu trúc lồng nhau. Truy nhập thành phần ngoài rồi đến thành phần của cấu trúc bên trong, sử dụng các phép toán . hoặc → (các phép toán lấy thành phần) một cách thích hợp.

```
x.ngaysinh.ng = y.ngaysinh.ng ;                 // gán ngày,
```

```
x.ngaysinh.th = y.ngaysinh.th ;                    // tháng,
x.ngaysinh.nam = y.ngaysinh.nam ;   // năm sinh của y cho x.
```

3.1.3. Phép toán gán cấu trúc

Cũng giống các biến mảng, để làm việc với một biến cấu trúc chúng ta phải thực hiện thao tác trên từng thành phần của chúng. Ví dụ vào/ra một biến cấu trúc phải viết câu lệnh vào/ra từng cho từng thành phần. Nhận xét này được minh họa trong ví dụ sau:

```
struct Sinhvien {
        char hoten[25] ;
        Ngaythang ns;
        int gt;
        float diem ;
} x, y;
cout << " Nhập dữ liệu cho sinh viên x:" << endl ;
cin.getline(x.hoten, 25);
cin >> x.ns.ng >> x.ns.th >> x.ns.nam;
cin >> x.gt;
cin >> x.diem
cout << "Thông tin về sinh viên x là:" << endl ;
cout << "Họ và tên: " << x.hoten << endl;
cout << "Sinh ngày: " << x.ns.ng << "/" << x.ns.th << "/" << x.ns.nam ;
cout << "Giới tính: " << (x.gt == 1) ? "Nam": "Nữ ;
cout << x.diem
```

Tuy nhiên, khác với biến mảng, **đối với cấu trúc chúng ta có thể gán giá trị của 2 biến cho nhau**. Phép gán này cũng tương đương với việc gán từng thành phần của cấu trúc. Ví dụ:

```
struct Sinhvien {
        char hoten[25] ;
        Ngaythang ns;
        int gt;
        float diem ;
} x, y, *p ;
cout << " Nhập dữ liệu cho sinh viên x:" << endl ;
cin.getline(x.hoten, 25);
```

```
cin >> x.ns.ng >> x.ns.th >> x.ns.nam;
cin >> x.gt;
cin >> x.diem


y = x ;     // Đối với biến mảng, phép gán này là không thực hiện được
p = new Sinhvien[1] ; *p = x ;


cout << "Thông tin về sinh viên y là:" << endl ;
cout << "Họ và tên: " << y.hoten << endl;
cout << "Sinh ngày: " << y.ns.ng << "/" << y.ns.th << "/" << y.ns.nam ;
cout << "Giới tính: " << (y.gt = 1) ? "Nam": "Nữ ;
cout << y.diem
```

Chú ý: không gán bộ giá trị cụ thể cho biến cấu trúc. Cách gán này chỉ thực hiện được khi khởi tạo. Ví dụ:

```
Sinhvien x = { "NVA", {1,1,1980}, 1, 7.0}, y ;                    // được
y = { "NVA", {1,1,1980}, 1, 7.0};                                 //
không được
y = x;                                                            //
```

được

3.1.4. Các ví dụ minh hoạ

Dưới đây chúng ta đưa ra một vài ví dụ minh hoạ cho việc sử dụng kiểu cấu trúc.

Ví dụ 4 : Cộng, trừ, nhân chia hai phân số được cho dưới dạng cấu trúc.

```
#include <iostream.h>
#include <conio.h>
struct Phanso {
    int tu ;
    int mau ;
} a, b, c ;


void main()
{
    clrscr();
    cout << "Nhập phân số a:" << endl ;                    // nhập a
```

```
    cout << "Tử:"; cin >> a.tu;
    cout << "Mẫu:"; cin >> a.mau;
    cout << "Nhập phân số b:" << endl ;                    // nhập b
    cout << "Tử:"; cin >> b.tu;
    cout << "Mẫu:"; cin >> b.mau;
    c.tu = a.tu*b.mau + a.mau*b.tu;                        // tính và in
a+b
    c.mau = a.mau*b.mau;
    cout << "a + b = " << c.tu << "/" << c.mau;
    c.tu = a.tu*b.mau - a.mau*b.tu;                    // tính và in a-b
    c.mau = a.mau*b.mau;
    cout << "a - b = " << c.tu << "/" << c.mau;
    c.tu = a.tu*b.tu;                                  // tính và in axb
    c.mau = a.mau*b.mau;
    cout << "a + b = " << c.tu << "/" << c.mau;
    c.tu = a.tu*b.mau;                                     // tính và in a/b
    c.mau = a.mau*b.tu;
    cout << "a + b = " << c.tu << "/" << c.mau;
    getch();
}
```

Ví dụ 5 : Nhập mảng K41T. Tính tuổi trung bình của sinh viên nam, nữ. Hiện danh sách của sinh viên có điểm thi cao nhất.

```
#include <iostream.h>
#include <conio.h>
void main()
{
      struct Sinhvien {
            char hoten[25] ;
            Ngaythang ns;
            int gt;
            float diem ;
      } x, K41T[60];
      int i, n;
```

```
        // nhập dữ liệu
    cout << "Cho biết số sinh viên: "; cin >> n;
        for (i=1, i<=n, i++)
        {
            cout << "Nhap sinh vien thu " << i);
            cout << "Ho ten: " ; cin.getline(x.hoten);
            cout << "Ngày sinh: " ; cin >> x.ns.ng >> x.ns.th >>x.ns.nam ;
            cout << "Giới tính: " ; cin >> x.gt ;
            cout << "Điểm: " ; cin >> x.diem ;
            cin.ignore();
            K41T[i] = x ;
        }
  }

// Tính điểm trung bình
float tbnam = 0, tbnu = 0;
int sonam = 0, sonu = 0 ;
for (i=1; i<=n; i++)
  if (K41T[i].gt == 1) { sonam++ ; tbnam += K41T[1].diem ; }
else { sonu++ ; tbnu += K41T[1].diem ; }
cout << "Điểm trung bình của sinh viên nam là " << tbnam/sonam ;
cout << "Điểm trung bình của sinh viên nữ là " << tbnu/sonu ;


// In danh sách sinh viên có điểm cao nhất
float diemmax = 0;
for (i=1; i<=n; i++)                                    // Tìm điểm cao nhất
if (diemmax < K41T[i].diem) diemmax = K41T[i].diem ;


for (i=1; i<=n; i++)                                    // In danh sách
{
      if (K41T[i].diem < diemmax) continue ;
      x = K41T[1] ;
      cout << x.hoten << '\t' ;
      cout << x.ns.ng << "/" << x.ns.th << "/" << x.ns.nam << '\t' ;
```

```
            cout << (x.gt == 1) ? "Nam": "Nữ" << '\t' ;
            cout << x.diem << endl;
        }
}
```

3.1.5.Hàm với cấu trúc

I. Con trỏ và địa chỉ cấu trúc

Một con trỏ cấu trúc cũng giống như con trỏ trỏ đến các kiểu dữ liệu khác, có nghĩa nó chứa địa chỉ của một biến cấu trúc hoặc một vùng nhớ có kiểu cấu trúc nào đó. Một con trỏ cấu trúc được khởi tạo bởi:

- Gán địa chỉ của một biến cấu trúc, một thành phần của mảng, tương tự nếu địa chỉ của mảng (cũng là địa chỉ của phần tử đầu tiên của mảng) gán cho con trỏ thì ta cũng gọi là con trỏ mảng cấu trúc. Ví dụ:

```
struct Sinhvien {
  char hoten[25] ;
  Ngaythang ns;
  int gt;
  float diem ;
} x, y, *p, lop[60];

p = &x ;                    // cho con trỏ p trỏ tới biến cấu trúc x
p→diem = 5.0;               // gán giá trị 5.0 cho điểm của biến x
p = &lop[10] ;              // cho p trỏ tới sinh viên thứ 10 của lớp
cout << p→hoten;            // hiện họ tên của sinh viên này
*p = y ;                          // gán lại sinh viên thứ 10 là y
(*p).gt = 2;                // sửa lại giới tính của sinh viên thứ 10 là
nữ
```

Chú ý: thông qua ví dụ này ta còn thấy một cách khác nữa để truy nhập các thành phần của x được trỏ bởi con trỏ p. Khi đó *p là tương đương với x, do vậy ta dùng lại cú pháp sử dụng toán tử . sau *p để lấy thành phần như (*p).hoten, (*p).diem, …

- Con trỏ được khởi tạo do xin cấp phát bộ nhớ. Ví dụ:

```
Sinhvien *p, *q ;
p = new Sinhvien[1];
q = new Sinhvien[60];
```

trong ví dụ này *p có thể thay cho một biến kiểu sinh viên (tương đương biến x ở trên) còn q có thể được dùng để quản lý một danh sách có tối đa là 60 sinh viên (tương đương biến lop[60], ví dụ khi đó *(p+10) là sinh viên thứ 10 trong danh sách).

- Đối với con trỏ p trỏ đến mảng a, chúng ta có thể sử dụng một số cách sau để truy nhập đến các trường của các thành phần trong mảng, ví dụ để truy cập hoten của thành phần thứ i của mảng a ta có thể viết:
  - p[i].hoten
  - (p+i)→hoten
  - *(p+i).hoten

Nói chung các cách viết trên đều dễ nhớ do suy từ kiểu mảng và con trỏ mảng. Cụ thể trong đó p[i] là thành phần thứ i của mảng a, tức a[i]. (p+i) là con trỏ trỏ đến thành phần thứ i và *(p+i) chính là a[i]. Ví dụ sau gán giá trị cho thành phần thứ 10 của mảng sinh viên lop, sau đó in ra màn hình các thông tin này. Ví dụ dùng để minh hoạ các cách truy nhập trường dữ liệu của thành phần trong mảng lop.

Ví dụ 6 :

```
struct Sinhvien {
        char hoten[25] ;
        Ngaythang ns;
        int gt;
        float diem ;
} lop[60] ;


strcpy(lop[10].hoten, "NVA");
lop[10].gt = 1; lop[10].diem = 9.0 ;
Sinhvien *p ;                       // khai báo thêm biến con trỏ Sinh viên
p = &lop ;                                  // cho con trỏ p trỏ tới mảng lop
cout << p[10].hoten ;               // in họ tên sinh viên thứ 10
cout << (p+10)  gt ;          // in giới tính của sinh viên thứ 10
```

```
cout << (*(p+10)).diem ;            // in điểm của sinh viên thứ 10
```

Chú ý: Độ ưu tiên của toán tử lấy thành phần (dấu chấm) là cao hơn các toán tử lấy địa chỉ (&) và lấy giá trị (*) nên cần phải viết các dấu ngoặc đúng cách.

m. Địa chỉ của các thành phần của cấu trúc

Các thành phần của một cấu trúc cũng giống như các biến, do vậy cách lấy địa chỉ của các thành phần này cũng tương tự như đối với biến bình thường. Chẳng hạn địa chỉ của thành phần giới tính của biến cấu trúc x là &x.gt (lưu ý độ ưu tiên của . cao hơn &, nên &x.gt là cũng tương đương với &(x.gt)), địa chỉ của trường hoten của thành phần thứ 10 trong mảng lớp là lop[10].hoten (hoten là xâu kí tự), tương tự địa chỉ của thành phần diem của biến được trỏ bởi p là &(p→diem).

Ví dụ:

```
struct Sinhvien {
        char hoten[25] ;
        Ngaythang ns;
        int gt;
        float diem ;
} lop[60], *p, x = { "NVA", {1,1,1980}, 1, 9.0) };
  lop[10] = x; p = &lop[10] ;       // p trỏ đến sinh viên thứ 10 trong lop
  char *ht; int *gt; float *d;      // các con trỏ kiểu thành phần
  ht = x.ht ;                              // cho ht trỏ đến thành phần hoten
của x
        gt = &(lop[10].gt) ;               // gt trỏ đến gt của sinh viên thứ 10
  d = &(p→diem) ;                  // p trỏ đến diem của sv p đang trỏ
  cout << ht ;                     // in họ tên sinh viên x
  cout << *gt ;                    // in giới tính của sinh viên thứ 10
  cout << *d ;                     // in điểm của sinh viên p đang trỏ.
```

n. Đối của hàm là cấu trúc

Một cấu trúc có thể được sử dụng để làm đối của hàm dưới các dạng sau đây:

- Là một biến cấu trúc, khi đó tham đối thực sự là một cấu trúc.
- Là một con trỏ cấu trúc, tham đối thực sự là địa chỉ của một cấu trúc.
- Là một tham chiếu cấu trúc, tham đối thực sự là một cấu trúc.
- Là một mảng cấu trúc hình thức hoặc con trỏ mảng, tham đối thực sự là tên mảng cấu trúc.

Ví dụ 7 : Ví dụ sau đây cho phép tính chính xác khoảng cách của 2 ngày tháng bất kỳ, từ đó có thể suy ra thứ của một ngày tháng bất kỳ. Đối của các hàm là một biến cấu trúc.

- Khai báo

```
struct DATE {                                          // Kiểu ngày tháng
  int ngay ;
  int thang;
  int nam ;
};
// Số ngày của mỗi tháng
int n[13] = {0,31,28,31,30,31,30,31,31,30,31,30,31};
```

- Hàm tính năm nhuận hay không nhuận, trả lại 1 nếu năm nhuận, ngược lại trả 0.

```
int Nhuan(int nm)
{
    return (nam%4==0 && nam%100!=0 || nam%400==0)? 1: 0;
}
```

- Hàm trả lại số ngày của một tháng bất kỳ. Nếu năm nhuận và là tháng hai số ngày của tháng hai (28) được cộng thêm 1.

```
int Ngayct(int thang, int nam)
{
    return n[thang] + ((thang==2) ? Nhuan(nam): 0);
}
```

- Hàm trả lại số ngày tính từ ngày 1 tháng 1 năm 1 bằng cách cộng dồn số ngày của từng năm từ năm 1 đến năm hiện tại, tiếp theo cộng dồn số ngày từng tháng của năm hiện tại cho đến tháng hiện tại và cuối cùng cộng thêm

số ngày hiện tại.

```
long Tongngay(DATE d)
{
    long i, kq = 0;
    for (i=1; i<d.nam; i++) kq += 365 + Nhuan(i);
    for (i=1; i<d.thang; i++) kq += Ngayct(i,d.nam);
    kq += d.ngay;
    return kq;
}
```

- Hàm trả lại khoảng cách giữa 2 ngày bất kỳ.

```
long Khoangcach(DATE d1, DATE d2)
{
    return Tongngay(d1)-Tongngay(d2);
}
```

- Hàm trả lại thứ của một ngày bất kỳ. Qui ước 1 là chủ nhật, 2 là thứ hai, … Để tính thứ hàm dựa trên một ngày chuẩn nào đó (ở đây là ngày 1/1/2000, được biết là thứ bảy). Từ ngày chuẩn này nếu cộng hoặc trừ 7 sẽ cho ra ngày mới cũng là thứ bảy. Từ đó, tính khoảng cách giữa ngày cần tính thứ và ngày chuẩn. Tìm phần dư của phép chia khoảng cách này với 7, nếu phần dư là 0 thì thứ là bảy, phần dư là 1 thì thứ là chủ nhật …

```
int Thu(DATE d)
{
    DATE curdate = {1,1,2000};          // ngày 1/1/2000 là thứ bảy
    long kc = Khoangcach(d, curdate);
    int du = kc % 7; if (du < 0) du += 7;
    return du;
}
```

- Hàm dịch một số dư sang thứ

```
char* Dich(int t)
```

```
{
    char* kq = new char[10];
    switch (t) {
    case 0: strcpy(kq, "thứ bảy"); break;
        case 1: strcpy(kq, "chủ nhật"); break;
    case 2: strcpy(kq, "thứ hai"); break;
        case 3: strcpy(kq, "thứ ba"); break;
    case 4: strcpy(kq, "thứ tư"); break;
    case 5: strcpy(kq, "thứ năm"); break;
        case 6: strcpy(kq, "thứ sáu"); break;
  }
  return kq;
}
```

- Hàm main()

```
void main()
{
    DATE d;
  cout << "Nhap ngay thang nam: " ;
    cin >> d.ngay >> d.thang >> d.nam ;
    cout << "Ngày " << d.ngay << "/" << d.thang << "/" << d.nam ;
    cout << " là " << Dich(Thu(d));
}
```

Ví dụ 8 : Chương trình đơn giản về quản lý sinh viên.

- Khai báo.

```
struct Sinhvien {                          // cấu trúc sinh viên
  char hoten[25] ;
  Ngaythang ns;
  int gt;
  float diem ;
} ;
Sinhvien lop[3];                           // lớp chứa tối đa 3 sinh viên
```

- Hàm in thông tin về sinh viên sử dụng biến cấu trúc làm đối. Trong lời gọi sử dụng biến cấu trúc để truyền cho hàm.

```
void in(Sinhvien x)
{
    cout << x.hoten << "\t" ;
    cout << x.ns.ng << "/" << x.ns.th << "/" << x.ns.nam << "\t" ;
    cout << x.gt << "\t";
    cout << x.diem << endl;
}
```

- Hàm nhập thông tin về sinh viên sử dụng con trỏ sinh viên làm đối. Trong lời gọi sử dụng địa chỉ của một cấu trúc để truyền cho hàm.

```
void nhap(Sinhvien *p)
{
    cin.ignore();
    cout << "Họ tên: "; cin.getline(p→hoten, 25) ;
    cout << "Ngày sinh: ";
       cin >> (p→ns).ng >> (p→ns).th >> (p→ns).nam ;
    cout << "Giới tính: "; cin >> (p→gt) ;
    cout << "Điểm: "; cin >> (p→diem) ;
}
```

- Hàm sửa thông tin về sinh viên sử dụng tham chiếu cấu trúc làm đối. Trong lời gọi sử dụng biến cấu trúc để truyền cho hàm.

```
void sua(Sinhvien &r)
{
    int chon;
    do {
            cout << "1: Sửa họ tên" << endl ;
            cout << "2: Sửa ngày sinh" << endl ;
            cout << "3: Sửa giới tính" << endl ;
            cout << "4: Sửa điểm" << endl ;
            cout << "0: Thôi" << endl ;
            cout << "Sửa (0/1/2/3/4) ? ; cin >> chon ; cin.ignore();
            switch (chon) {
                    case 1: cin.getline(r.hoten, 25) ; break;
                    case 2: cin >> r.ns.ng >> r.ns.th >> r.ns.nam ; break;
                    case 3: cin >> r.gt ; break;
```

```
                case 4: cin >> r.diem ; break;
            }
        } while (chon) ;
    }
```

- Hàm nhapds nhập thông tin của mọi sinh viên trong mảng, sử dụng con trỏ mảng Sinhvien làm tham đối hình thức. Trong lời gọi sử dụng tên mảng để truyền cho hàm.

```
void nhapds(Sinhvien *a)
{
    int sosv = sizeof(lop) / sizeof(Sinhvien) -1;          // bỏ phần tử 0
    for (int i=1; i<=sosv; i++) nhap(&a[i]) ;
}
```

- Hàm suads cho phép sửa thông tin của sinh viên trong mảng, sử dụng con trỏ mảng Sinhvien làm tham đối hình thức. Trong lời gọi sử dụng tên mảng để truyền cho hàm.

```
void suads(Sinhvien *a)
{
    int chon;
        cout << "Chọn sinh viên cần sửa: " ; cin >> chon ; cin.ignore();
        sua(a[chon]) ;
}
```

- Hàm inds hiện thông tin của mọi sinh viên trong mảng, sử dụng hằng con trỏ mảng Sinhvien làm tham đối hình thức. Trong lời gọi sử dụng tên mảng để truyền cho hàm.

```
void hien(const Sinhvien *a)
{
        int sosv = sizeof(lop) / sizeof(Sinhvien) -1;          // bỏ phần tử 0
    for (int i=1; i<=sosv; i++) in(a[i]) ;
}
```

- Hàm main() gọi chạy các hàm trên để nhập, in, sửa danh sách sinh viên.

```
void main()
{
    nhapds(lop) ;
```

```
    inds(lop);
    suads(lop);
    inds(lop);
}
```

o. Giá trị hàm là cấu trúc

Cũng tương tự như các kiểu dữ liệu cơ bản, giá trị trả lại của một hàm cũng có thể là các cấu trúc dưới các dạng sau:

- là một biến cấu trúc.
- là một con trỏ cấu trúc.
- là một tham chiếu cấu trúc.

Sau đây là các ví dụ minh hoạ giá trị cấu trúc của hàm.

Ví dụ 9 : Đối và giá trị của hàm là cấu trúc: Cộng, trừ hai số phức.

– Khai báo kiểu số phức

```
struct Sophuc                    // Khai báo kiểu số phức dùng chung
{
        float thuc;
        float ao;
};
```

- Hàm cộng 2 số phức, trả lại một số phức

```
Sophuc Cong(Sophuc x, Sophuc y)
  {
    Sophuc kq;
    kq.thuc = x.thuc + y.thuc ;
    kq.ao = x.ao + y.ao ;
    return kq;
}
```

- Hàm trừ 2 số phức, trả lại một số phức

```
Sophuc Tru(Sophuc x, Sophuc y)
{
  Sophuc kq;
  kq.thuc = x.thuc + y.thuc ;
  kq.ao = x.ao + y.ao ;
```

```
  return kq;
}
```

- Hàm in một số phức dạng (r + im)

```
void In(Sophuc x)
{
    cout << "(" << x.thuc << "," << x.ao << ")" << endl ;
}
```

- Hàm chính

```
void main()
{
    Sophuc x, y, z ;
    cout << "x = " ; cin >> x.thuc >> x.ao ;
    cout << "y = " ; cin >> y.thuc >> y.ao ;
    cout << "x + y = " ; In(Cong(x,y)) ;
    cout << "x - y = " ; In(Tru(x,y)) ;
}
```

Ví dụ 10 : Chương trình nhập và in thông tin về một lớp cùng sinh viên có điểm cao nhất lớp.

- Khai báo kiểu dữ liệu Sinh viên và biến mảng lop.

```
struct Sinhvien {
    char *hoten ;
    float diem ;
} lop[4] ;
```

- Hàm nhập sinh viên, giá trị trả lại là một con trỏ trỏ đến dữ liệu vừa nhập.

```
Sinhvien* nhap()
{
        Sinhvien* kq = new Sinhvien[1];          // nhớ cấp phát vùng
nhớ
```

```
        kq->hoten = new char[15];                    // cho cả con trỏ
    hoten
        cout << "Họ tên: "; cin.getline(kq->hoten,30);
        cout << "Điểm: "; cin >> kq->diem; cin.ignore();
        return kq;                                   // trả lại con
    trỏ kq
    }
```

- Hàm tìm sinh viên có điểm cao nhất, giá trị trả lại là một tham chiếu đến sinh viên tìm được.

```
Sinhvien& svmax()
{
    int sosv = sizeof(lop)/sizeof(Sinhvien)-1;      // bỏ thành phần thứ 0
    float maxdiem = 0;
    int kmax;                                       // chỉ số sv có
điểm max
    for (int i=1; i<sosv; i++)
    if (maxdiem < lop[i].diem)
    {
        maxdiem = lop[i].diem ;
        kmax = i;
    }
    return lop[kmax];                               // trả lại sv có điểm
max
}
```

- Hàm in thông tin của một sinh viên x

```
void in(Sinhvien x)
{
    cout << x.hoten << "\t";
    cout << x.diem << endl;
}
```

- Hàm chính

```
void main()
{
    clrscr();
    int i;
    int sosv = sizeof(lop)/sizeof(Sinhvien)-1;      // bỏ thành phần thứ 0
    for (i=1; i<=sosv; i++) lop[i] = *nhap();       // nhập danh sách lớp
    for (i=1; i<=sosv; i++) in(lop[i]);             // in danh sách
lớp
    Sinhvien &b = svmax();          // khai báo tham chiếu b và cho
                                    // tham chiếu đến sv có điểm max
    in(b);                          // in sinh viên có
điểm max
    getch();
}
```

3.1.6.Cấu trúc với thành phần kiểu bit

p. **Trường bit**

Thông thường các trường trong một cấu trúc thường sử dụng ít nhất là 2 byte tức 16 bit. Trong nhiều trường hợp một số trường có thể chỉ cần đến số bit ít hơn, ví dụ trường gioitinh thông thường chỉ cần đến 1 bit để lưu trữ. Những trường hợp như vậy ta có thể khai báo kiểu bit cho các trường này để tiết kiệm bộ nhớ. Tuy nhiên, cách khai báo này ít được sử dụng trừ khi cần thiết phải truy nhập đến mức bit của dữ liệu trong các chương trình liên quan đến hệ thống.

Một trường bit là một khai báo trường int và thêm dấu: cùng số bit n theo sau, trong đó $0 \leq n < 15$. Ví dụ do độ lớn của ngày không vượt quá 31, tháng không vuợt quá 12 nên 2 trường này trong cấu trúc ngày tháng có thể khai báo tiết kiệm hơn bằng 5 và 4 bit như sau:

```
struct Date {
    int ng: 5;
    int th: 4;
    int nam;
} ;
```

***q.* Đặc điểm**

Cần chú ý các đặc điểm sau của một cấu trúc có chứa trường bit:

- Các bit được bố trí liên tục trên dãy các byte.
- Kiểu trường bit phải là int (signed hoặc unsigned).
- Độ dài mỗi trường bit không quá 16 bit.
- Có thể bỏ qua một số bit nếu bỏ trống tên trường, ví dụ:

```
struct tu {
        int: 8;
        int x:8;
}
```

mỗi một biến cấu trúc theo khai báo trên gồm 2 byte, bỏ qua không sử dụng byte thấp và trường x chiếm byte (8 bit) cao.

- Không cho phép lấy địa chỉ của thành phần kiểu bit.
- Không thể xây dựng được mảng kiểu bit.
- Không được trả về từ hàm một thành phần kiểu bit. Ví dụ nếu b là một thành phần của biến cấu trúc x có kiểu bit thì câu lệnh sau là sai:

```
return x.b ;                    // sai
```

tuy nhiên có thể thông qua biến phụ như sau:

```
int tam = x.b ; return tam ;
```

- Tiết kiệm bộ nhớ
- Dùng trong kiểu union để lấy các bit của một từ (xem ví dụ trong phần kiểu hợp).

3.1.7.Câu lệnh typedef

Để thuận tiện trong sử dụng, thông thường các kiểu được NSD tạo mới sẽ được gán cho một tên kiểu bằng câu lệnh typedef như sau:

typedef <kiểu> <tên_kiểu> ;

Ví dụ: Để tạo kiểu mới có tên Bool và chỉ chứa giá trị nguyên (thực chất chỉ cần 2 giá trị 0, 1), ta có thể khai báo:

```
typedef int Bool;
```

khai báo này cho phép xem Bool như kiểu số nguyên.

hoặc có thể đặt tên cho kiểu ngày tháng là Date với khai báo sau:

```
typedef struct Date {
        int ng;
```

```
        int th;
        int nam;
    };
```

khi đó ta có thể sử dụng các tên kiểu này trong các khai báo (ví dụ tên kiểu của đối, của giá trị hàm trả lại …).

3.1.8.Hàm sizeof()

Hàm trả lại kích thước của một biến hoặc kiểu. Ví dụ:

```
    Bool a, b;
    Date x, y, z[50];
    cout << sizeof(a) << sizeof(b) << sizeof(Bool) ;                    //
in 2 2 2
    cout << "Số phần tử của z = " << sizeof(z) / sizeof(Date)           // in 50
```

3.2. CẤU TRÚC TỰ TRỎ VÀ DANH SÁCH LIÊN KẾT

Thông thường khi thiết kế chương trình chúng ta chưa biết được số lượng dữ liệu cần dùng là bao nhiêu để khai báo số biến cho phù hợp. Đặc biệt là biến dữ liệu kiểu mảng. Số lượng các thành phần của biến mảng cần phải khai báo trước và chương trình dịch sẽ bố trí vùng nhớ cố định cho các biến này. Do buộc phải khai báo trước số lượng thành phần nên kiểu mảng thường dẫn đến hoặc là lãng phí bộ nhớ (khi chương trình không dùng hết) hoặc là không đủ để chứa dữ liệu (khi chương trình cần chứa dữ liệu với số lượng thành phần lớn hơn).

Để khắc phục tình trạng này C++ cho phép cấp phát bộ nhớ động, nghĩa là số lượng các thành phần không cần phải khai báo trước. Bằng toán tử new chúng ta có thể xin cấp phát vùng nhớ theo nhu cầu mỗi khi chạy chương trình. Tuy nhiên, cách làm này dẫn đến việc dữ liệu của một danh sách sẽ không còn nằm liên tục trong bộ nhớ như đối với biến mảng. Mỗi lần xin cấp phát bởi toán tử new, chương trình sẽ tìm một vùng nhớ đang rỗi bất kỳ để cấp phát cho biến và như vậy dữ liệu sẽ nằm rải rác trong bộ nhớ. Từ đó, để dễ dàng quản lý trật tự của một danh sách các dữ liệu, mỗi thành phần của danh sách cần phải chứa địa chỉ của thành phần tiếp theo hoặc trước nó (điều này là không cần thiết đối với biến mảng vì các thành phần của chúng sắp xếp liên tục, kề nhau). Từ đó, mỗi thành phần của danh sách sẽ là một cấu trúc, ngoài các thành phần chứa thông tin của bản thân, chúng còn phải có thêm một hoặc nhiều con trỏ để trỏ đến các thành phần tiếp theo hay đứng trước. Các cấu trúc như vậy được gọi là cấu trúc tự trỏ vì các thành phần con trỏ trong cấu trúc này sẽ trỏ đến các vùng dữ liệu có kiểu chính là kiểu của chúng.

3.2.1.Cấu trúc tự trỏ

Một cấu trúc có chứa ít nhất một thành phần con trỏ có kiểu của chính cấu trúc đang định nghĩa được gọi là cấu trúc tự trỏ. Có thể khai báo cấu trúc tự trỏ bởi một trong những cách sau:

Cách 1:

```
typedef struct <tên cấu trúc> <tên kiểu> ;          // định nghĩa tên cấu trúc
struct <tên cấu trúc>
{
    các thành phần chứa thông tin … ;
    <tên kiểu> *con trỏ ;
} ;
```

Ví dụ:

```
typedef struct Sv Sinhvien ;                // lưu ý phải có từ khoá struct
struct Sv
{
   char hoten[30] ;            // thành phần chứa thông tin
   float diem ;          // thành phần chứa thông tin
   Sinhvien *tiep ;            // thành phần con trỏ chứa địa chỉ tiếp theo
} ;
```

Cách 2:

```
struct <tên cấu trúc>
{
    các thành phần chứa thông tin … ;
    <tên cấu trúc> *con trỏ ;
} ;
typedef <tên cấu trúc> <tên kiểu> ;          // định nghĩa tên cấu trúc tự trỏ
```

Ví dụ:

```
struct Sv
{
   char hoten[30] ;                  // thành phần chứa thông tin
```

```
    float diem ;                    // thành phần chứa thông tin
    Sv *tiep ;                      // thành phần con trỏ chứa địa chỉ
tiếp theo
} ;
typedef Sv Sinhvien ;
```

Cách 3:

```
typedef struct <tên kiểu>           // định nghĩa tên cấu trúc tự trỏ
{
    các thành phần chứa thông tin … ;
    <tên kiểu> *con trỏ ;
} ;
```

Ví dụ:

```
typedef struct Sinhvien
{
    char hoten[30] ;                // thành phần chứa thông tin
    float diem ;                    // thành phần chứa thông tin
    Sinhvien *tiep ;                // con trỏ chứa địa chỉ thành phần
tiếp theo
} ;
```

Cách 4:

```
struct <tên kiểu>
{
    các thành phần chứa thông tin … ;
    <tên kiểu> *con trỏ ;
} ;
```

Ví dụ:

```
struct Sinhvien
{
    char hoten[30] ;                // thành phần chứa thông tin
    float diem ;                    // thành phần chứa thông tin
    Sinhvien *tiep ;                // con trỏ chứa địa chỉ của phần tử
tiếp.
```

```
} ;
```

Trong các cách trên ta thấy 2 cách khai báo cuối cùng là đơn giản nhất. C++ quan niệm các tên gọi đứng sau các từ khoá struct, union, enum là các tên kiểu (dù không có từ khoá typedef), do vậy có thể sử dụng các tên này để khai báo.

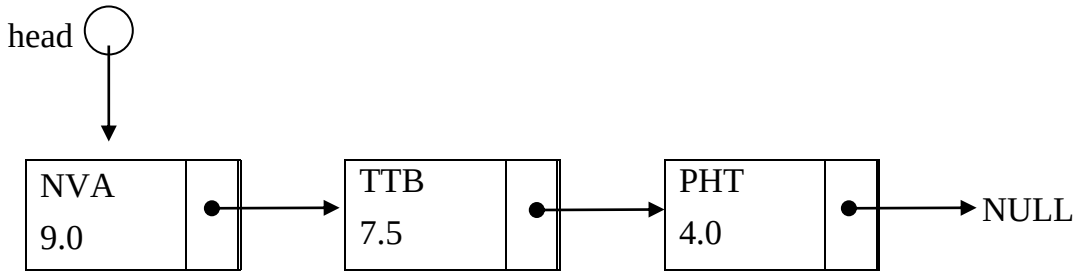
3.2.2. Khái niệm danh sách liên kết

Danh sách liên kết là một cấu trúc dữ liệu cho phép thể hiện và quản lý danh sách bằng các cấu trúc liên kết với nhau thông qua các con trỏ trong cấu trúc. Có nhiều dạng danh sách liên kết phụ thuộc vào các kết nối, ví dụ:

- Danh sách liên kết đơn, mỗi cấu trúc chứa một con trỏ trỏ đến cấu trúc tiếp theo hoặc trước đó. Đối với danh sách con trỏ trỏ về trước, cấu trúc đầu tiên của danh sách sẽ trỏ về hằng con trỏ NULL, cấu trúc cuối cùng được đánh dấu bởi con trỏ last là con trỏ trỏ vào cấu trúc này. Đối với danh sách con trỏ trỏ về cấu trúc tiếp theo, cấu trúc đầu sẽ được đánh dấu bằng con trỏ head và cấu trúc cuối cùng chưa con trỏ NULL.
- Danh sách liên kết kép gồm 2 con trỏ, một trỏ đến cấu trúc trước và một trỏ đến cấu trúc sau, 2 đầu của danh sách được đánh dấu bởi các con trỏ head, last.
- Danh sách liên kết vòng gồm 1 con trỏ trỏ về sau (hoặc trước), hai đầu của danh sách được nối với nhau tạo thành vòng tròn. Chỉ cần một con trỏ head để đánh dấu đầu danh sách.

Do trong cấu trúc có chứa các con trỏ trỏ đến cấu trúc tiếp theo và/hoặc cấu trúc đứng trước nên từ một cấu trúc này chúng ta có thể truy cập đến một cấu trúc khác (trước và/hoặc sau nó). Kết hợp với các con trỏ đánh dấu 2 đầu danh sách (head, last) chúng ta sẽ dễ dàng làm việc với bất kỳ phần tử nào của danh sách. Có thể kể một số công việc thường thực hiện trên một danh sách như: bổ sung phần tử vào cuối danh sách, chèn thêm một phần tử mới, xoá một phần tử của danh sách, tìm kiếm, sắp xếp danh sách, in danh sách …

Hình vẽ bên dưới minh hoạ một danh sách liên kết đơn quản lý sinh viên, thông tin chứa trong mỗi phần tử của danh sách gồm có họ tên sinh viên, điểm. Ngoài ra mỗi phần tử còn chứa con trỏ tiep để nối với phần tử tiếp theo của nó. Phần tử cuối cùng nối với cấu trúc rỗng (NULL).



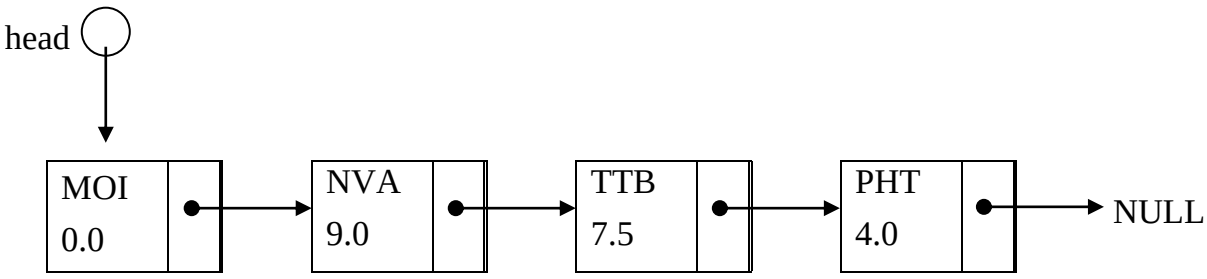
3.2.3. Các phép toán trên danh sách liên kết

Dưới đây chúng ta mô tả tóm tắt cách thức thực hiện một số thao tác trên danh sách liên kết đơn.

a. Tạo phần tử mới

Để tạo phần tử mới thông thường chúng ta thực hiện theo các bước sau đây:

- dùng toán tử new xin cấp phát một vùng nhớ đủ chứa một phần tử của danh sách.
- nhập thông tin cần lưu trữ vào phần tử mới. Con trỏ tiep được đặt bằng NULL.
- gắn phần tử vừa tạo được vào danh sách. Có hai cách:
  - hoặc gắn vào đầu danh sách, khi đó vị trí của con trỏ head (chỉ vào đầu danh sách) được điều chỉnh lại để chỉ vào phần tử mới.
  - hoặc gắn vào cuối danh sách bằng cách cho con trỏ tiep của phần tử cuối danh sách (đang trỏ vào NULL) trỏ vào phần tử mới. Nếu danh sách có con trỏ last để chỉ vào cuối danh sách thì last được điều chỉnh để trỏ vào phần tử mới. Nếu danh sách không có con trỏ last thì để tìm được phần tử cuối chương trình phải duyệt từ đầu, bắt đầu từ con trỏ head cho đến khi gặp phần tử trỏ vào NULL, đó là phần tử cuối của danh sách.

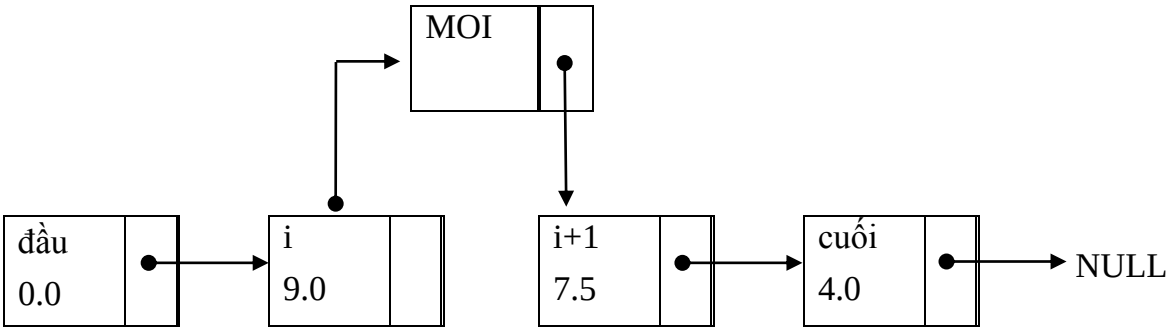


Gắn phần tử mới vào đầu danh sách

b. **Chèn phần tử mới vào giữa**

Giả sử phần tử mới được chèn vào giữa phần tử thứ i và i+1. Để chèn ta nối phần tử thứ i vào phần tử mới và phần tử mới nối vào phần tử thứ i+1. Thuật toán sẽ như sau:

- Cho con trỏ p chạy đến phần tử thứ i.
- Cho con trỏ tiep của phần tử mới trỏ vào phần tử thứ i+1 (tuc p->tiep).
- Cho con trỏ tiep của phần tử thứ i (hiện được trỏ bởi p) thay vì trỏ vào phần tử thứ i+1 bây giờ sẽ trỏ vào phần tử mới.

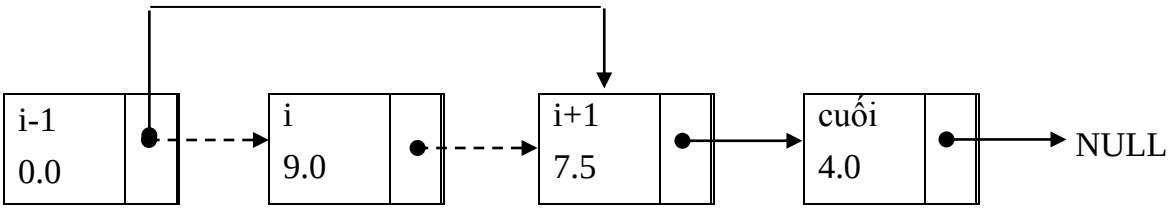


Chèn phần tử mới vào giữa phần tử i và i+1

a. Xoá phần tử thứ i khỏi danh sách

Việc xoá một phần tử ra khỏi danh sách rất đơn giản bởi chỉ việc thay đổi các con trỏ. Cụ thể giả sử cần xoá phần tử thứ i ta chỉ cần cho con trỏ tiep của phần tử thứ i-1 trỏ ("vòng qua" phần tử thứ i) vào phần tử thứ i+1. Như vậy bây giờ khi chạy trên danh sách đến phần tử thứ i-1, phần tử tiếp theo là phần tử thứ i+1 chứ không còn là phần tử thư i. Nói cách khác phần tử thứ i không được nối bởi bất kỳ phần tử nào nên nó sẽ không thuộc danh sách. Có thể thực hiện các bước như sau:

- Cho con trỏ p chạy đến phần tử thứ i-1.
- Đặt phần tử thứ i vào biến x.
- Cho con trỏ tiep của phần tử thứ i-1 trỏ vào phần tử thứ i+1 bằng cách đặt tiep = x.tiep.
- Giải phóng bộ nhớ được trỏ bởi x bằng câu lệnh delete x.



Xóa phần tử thứ i

c. Duyệt danh sách

Duyệt là thao tác đi qua từng phần tử của danh sách, tại mỗi phần tử chương trình thực hiện một công việc gì đó trên phần tử mà ta gọi là thăm phần tử đó. Một phép thăm có thể đơn giản là hiện nội dung thông tin của phần tử đó ra màn hình chẳng hạn. Để duyệt danh sách ta chỉ cần cho một con trỏ p chạy từ đầu đến cuối danh sách đến khi phần tử cuối có con trỏ tiep = NULL thì dừng. Câu lệnh cho con trỏ p chuyển đến phần tử tiếp theo của nó là:

p = p → tiep ;

d. Tìm kiếm

Cho một danh sách trong đó mỗi phần tử của danh sách đều chứa một trường gọi là trường khoá, thường là các trường có kiểu cơ sở hoặc kết hợp của một số trường như vậy. Bài toán đặt ra là tìm trên danh sách phần tử có giá trị của trường khoá bằng với một giá trị cho trước. Tiến trình thực hiện nhiệm vụ thựcchất cũng là bài toán duyệt, trong đó thao tác "thăm" chính là so sánh trường khoá của phần tử với giá trị cho trước, nếu trùng nhau ta in kết quả và dừng, Nếu đã duyệt hết mà không có phần tử nào có trường khoá trùng với giá trị cho trước thì xem danh sách không chứa giá trị này.

Ngoài các thao tác trên, nói chung còn nhiều các thao tác quen thuộc khác tuy nhiên chúng ta không trình bày ở đây vì nó không thuộc phạm vi của giáo trình này.

Dưới đây là một ví dụ minh hoạ cho cấc cấu trúc tự trỏ, danh sách liên kết và một vài thao tác trên danh sách liên kết thông qua bài toán quản lý sinh viên.

- Khai báo

```
struct DATE
{
        int day, month, year;                          // ngày, tháng,
năm
};
struct Sinhvien {                                      // cấu trúc tự
```

```
trỏ
    char hoten[31];
    DATE ns;
    float diem;
    Sinhvien *tiep ;
};
Sinhvien *dau = NULL, *cuoi = NULL;      // Các con trỏ tới đầu và cuối
ds
Sinhvien *cur = NULL;                    // Con trỏ tới sv hiện
tại
int sosv = 0;                            // Số sv của danh sách
```

- Tạo sinh viên mới và nhập thông tin, trả lại con trỏ trỏ đến sinh viên mới.

```
Sinhvien* Nhap1sv()                      // Tạo 1 khối dữ liệu
cho sv mới
{
    Sinhvien *kq = new Sinhvien[1] ; // Cấp phát bộ nhớ cho kq
       cout << "\nSinh vien thu ", sosv+1 ;
    cout << "Ho ten = " ; cin.getline(kq->hoten);
    cout << "Ns = " ; cin >> kq->ns.day >> kq->ns.month >> kq-
>ns.year;
    cout << "Diem = " ; cin >> kq->diem ; cin.ignore() ;
    kq->tiep = NULL;
    return kq ;
}
```

- Bổ sung sinh viên mới vào cuối danh sách.

```
void Bosung()                            // Bổ sung sv mới vào
cuối ds
{
    cur = Nhap1sv();
    if (sosv == 0) {dau = cuoi = cur;}
    else { cuoi->tiep = cur; cuoi = cur; }
    sosv++;
}
```

- Chèn sv mới vào trước sinh viên thứ n.

```
void Chentruoc(int n)                    // Chèn sv mới vào trước sv thứ n
{
    cur = Nhap1sv();
    if (sosv==0) { dau = cuoi = cur; sosv++; return; }
    if (sosv==1 || n==1) {cur->tiep = dau; dau = cur; sosv++; return;}
    Sinhvien *truoc, *sau;
    truoc = dau;
    sau = dau -> tiep;
    for (int i=1; i<n-1; i++) truoc = truoc->tiep;
    sau = truoc->tiep;
    truoc->tiep = cur;
    cur -> tiep = sau;
    sosv ++;
}
```

- Chèn sv mới vào sau sinh viên thứ n.

```
void Chensau(int n)                      // Chèn sv mới vào sau sv thứ n
{
    cur = Nhap1sv();
    if (sosv==0 || sosv<n) { dau = cuoi = cur; sosv++; return; }
    Sinhvien *truoc, *sau;
    truoc = dau; sau = dau -> tiep;
    for (int i=1; i<n; i++) truoc = truoc->tiep;
    sau = truoc->tiep;
    truoc->tiep = cur;
    cur -> tiep = sau;
    sosv ++;
}
```

- Xoá sinh viên thứ n.

```
void Xoa(int n)                          // Xoá sinh viên thứ n
{
```

```
    if (sosv==1&&n==1) { delete dau ; dau = cuoi = NULL; sosv--; return; }
    if (n==1) { cur = dau; dau = cur->tiep; delete cur; sosv--; return; }
    Sinhvien *truoc, *sau;
    truoc = dau;
    sau = dau -> tiep;
    for (int i=1; i<n-1; i++) truoc = truoc->tiep;
    cur = truoc->tiep; sau = cur->tiep; truoc->tiep = sau;
    delete cur ;
    sosv --;
}
```

- Tạo danh sách sinh viên.

```
void Taods()                                    // Tạo danh sách
{
    int tiep = 1;
    while (tiep) {
            Bosung();
            cout << "Tiep (0/1) ? " ; cin >> tiep ;
    }
}
```

- In danh sách sinh viên.

```
void Inds()                                     // In danh sách
{
    cur = dau;      int i=1;
    while (cur != NULL) {
            cout << "\nSinh vien thu " << i << " ----------------------------\n") ;
            cout << "Hoten:" << cur->hoten ;
            cout << "Ngay sinh: "
             cout << cur -> ns.day << "/" ;
             cout << cur -> ns.month << "/" ;
             cout << cur -> ns.year ;
            cout << "Diem: " << cur->diem ;
```

```
            cur = cur->tiep; i++;
        }
    }
```

- Hàm chính.

```
void main()
{
    clrscr();
    Taods();
    Inds();
    getch();
}
```

3.3. KIỂU HỢP

3.3.1. Khai báo

Giống như cấu trúc, kiểu hợp cũng có nhiều thành phần nhưng các thành phần của chúng sử dụng chung nhau một vùng nhớ. Do vậy kích thước của một kiểu hợp là độ dài của trường lớn nhất và việc thay đổi một thành phần sẽ ảnh hưởng đến tất cả các thành phần còn lại.

```
union <tên kiểu> {
        Danh sách các thành phần;
};
```

3.3.2. Truy cập

Cú pháp truy cập đến các thành phần của hợp cũng tương tự như kiểu cấu trúc, tức cũng sử dụng toán tử lấy thành phần (dấu chấm . hoặc □ cho biến con trỏ kiểu hợp). Dưới đây là một ví dụ minh hoạ việc sử dụng khai báo kiểu hợp để tách byte thấp, byte cao của một số nguyên.

Ví dụ 11 :

```
void main()
{
    union songuyen {
        int n;
        unsigned char c[2];
    } x;
    cout << "Nhập số nguyên: " ; cin >> x.n ;
    cout << "Byte thấp của x = " << x.c[0] << endl ;
```

```
        cout << "Byte cao của x = " << x.c[1] << endl;
}
```

Ví dụ 12 : Kết hợp cùng kiểu nhóm bit trong cấu trúc, chúng ta có thể tìm được các bit của một số như chương trình sau. Trong chương trình ta sử dụng một biến u có kiểu hợp. Trong kiểu hợp này có 2 thành phần là 2 cấu trúc lần lượt có tên s và f.

```
union {
        struct { unsigned a, b ; } s;
        struct {
                unsigned n1: 1;
                unsigned: 15;
                unsigned n2: 1;
                unsigned: 7;
                unsigned n3: 8;
        } t ;
} u;
```

với khai báo trên đây khi nhập u.s thì nó cũng ảnh hưởng đến u.t, cụ thể

- u.t.n1 là bit đầu tiên (0) của thành phần u.s.a
- u.t.n2 là bit 0 của thành phần u.s.b
- u.t.n3 là byte cao của u.s.b

3.4. KIỂU LIỆT KÊ

Có thể gán các giá trị nguyên liên tiếp (tính từ 0) cho các tên gọi cụ thể bằng kiểu liệt kê theo khai báo sau đây:

enum tên_kiểu { d/s tên các giá trị };

Ví dụ:

```
enum Bool {false, true};
```

khai báo kiểu mới đặt tên Bool chỉ nhận 1 trong 2 giá trị đặt tên false và true, trong đó false ứng với giá trị 0 và true ứng với giá trị 1. Cách khai báo kiểu enum trên cũng tương đương với dãy các macro sau:

```
#define false 0
#define true 1
```

Với kiểu Bool ta có thể khai báo một số biến như sau:

```
Bool Ok, found;
```

hai biến Ok và found sẽ chỉ nhận 1 trong 2 giá trị false (thay cho 0) hoặc true (thay cho 1). Có nghĩa có thể gán:

Ok = true;

hoặc: found = false;

Tuy nhiên không thể gán các giá trị nguyên trực tiếp cho các biến enum mà phải thông qua ép kiểu. Ví dụ:

```
Ok = 0;                 // sai
Ok = Bool(0) ;          // đúng
hoặc Ok = false ;       // đúng
```

BÀI TẬP

12. Có thể truy nhập thành phần của cấu trúc thông qua con trỏ như sau (với p là con trỏ cấu trúc và a là thành phần của cấu trúc):

A: (*p).a B: *p→a C: a và b sai D: a và b đúng

13. Cho khai báo struct T {int x; float y;} t, *p, a[10]; Câu lệnh nào trong các câu sau là không hợp lệ:

(1) p = &t; (2) p = &t.x; (3) p = a;

(4) p = &a (5) p = &a[5]; (6) p = &a[5].y;

A: 1, 2 và 3 B: 4, 5 và 6 C: 1, 3 và 5 D: 2, 4 và 6

14. Cho các khai báo sau:

```
struct ngay {int ng, th, nam;} vaotruong, ratruong;
typedef struct {char hoten[25]; ngay ngaysinh;} sinhvien;
```

Hãy chọn câu đúng nhất

A: Không được phép gán: ratruong = vaotruong;

B: sinhvien là tên cấu trúc, vaotruong, ratruong là biến cấu trúc

C: Có thể viết: vaotruong.ng, ratruong.th, sinhvien.vaotruong.nam để truy nhập đến các thành phần tương ứng.

D: a, b, c đúng

15. Trong các khởi tạo giá trị cho các cấu trúc sau, khởi tạo nào đúng:

```
struct S1 {
        int ngay, thang, nam;
} s1 = {2,3};
struct S2 {
        char hoten[10];
        struct S1 ngaysinh;
} s2 = {"Ly Ly",1,2,3};
```

```
struct S3 {
    struct S2 sinhvien;
    float diem;
} s3 = {{{"Cốc cốc", {4,5,6}}, 7};
```

A: S1 và S2 đúng B: S2 và S3 đúng C: S3 và S1 đúng D: Cả 3 cùng đúng

16. Đối với kiểu cấu trúc, cách gán nào dưới đây là không được phép:

A: Gán hai biến cho nhau.

B: Gán hai phần tử mảng (kiểu cấu trúc) cho nhau

C: Gán một phần tử mảng (kiểu cấu trúc) cho một biến và ngược lại

D: Gán hai mảng cấu trúc cùng số phần tử cho nhau

17. Cho đoạn chương trình sau:

```
struct {
    int to ;
    float soluong;
} x[10];
for (int i = 0; i < 10; i++) cin >> x[i].to >> x[i].soluong ;
```

Chọn câu đúng nhất trong các câu sau:

A: Đoạn chương trình trên có lỗi cú pháp

B: Không được phép sử dụng toán tử lấy địa chỉ đối với các thành phần to và soluong

C: Lấy địa chỉ thành phần soluong dẫn đến chương trình hoạt động không đúng đắn

D: Cả a, b, c đều sai

18. Chọn câu đúng nhất trong các câu sau:

A: Các thành phần của kiểu hợp (union) được cấp phát một vùng nhớ chung

B: Kích thước của kiểu hợp bằng kích thước của thành phần lớn nhất

C: Một biến kiểu hợp có thể được tổ chức để cho phép thay đổi được kiểu dữ liệu của biến trong qua trình chạy chương trình

D: a, b, c đúng

19. Cho khai báo:

```
union {
    unsigned x;
    unsigned char y[2];
} z = {0xabcd};
```

Chọn câu đúng nhất trong các câu sau:

A: Khai báo trên là sai vì thiếu tên kiểu

B: Khởi tạo biến z là sai vì chỉ có một giá trị (0xabcd)

C: z.y[0] = 0xab

D: z.y[1] = 0xab

20. Cho kiểu hợp:

```
union U {
    char x[1];
    int y[2]; float z[3];
} u;
```

Chọn câu đúng nhất trong các câu sau:

A: sizeof(U) = 1+2+3 = 6

B: sizeof(U) = max(sizeof(char), sizeof(int), sizeof(float))

C: sizeof(u) = max(sizeof(u.x), sizeof(u.y), sizeof(u.z))

D: b và c đúng

21. Cho khai báo:

```
union {
    unsigned x;
    struct {
        unsigned char a, b;
    } y;
} z = {0xabcd};
```

Giá trị của z.y.a và z.y.b tương ứng:

A: 0xab, 0xcd B: 0xcd, 0xab C: 0xabcd, 0 D: 0, 0xabcd

22. Cho khai báo:

```
union {
    struct {
        unsigned char a, b;
    } y;
    unsigned x;
} z = {{1,2}};
```

Giá trị của z.x bằng:

A: 513 B: 258

C: Không xác định vì khởi tạo sai D: Khởi tạo đúng nhưng z.x chưa có giá trị

23. Xét đoạn lệnh:

```
union U {
    int x; char y;
} u;
u.x = 0; u.y = 200;
```

Tìm giá trị của u.x + u.y ?

A: 122 B: 144 C: 200 D: 400

24. Cho số phức dưới dạng cấu trúc gồm 2 thành phần là thực và ảo. Viết chương trình nhập 2 số phức và in ra tổng, tích, hiệu, thương của chúng.

25. Cho phân số dưới dạng cấu trúc gồm 2 thành phần là tử và mẫu. Viết chương trình nhập 2 phân số, in ra tổng, tích, hiệu, thương của chúng dưới dạng tối giản.

26. Tính số ngày đã qua kể từ đầu năm cho đến ngày hiện tại. Qui ước ngày được khai báo dưới dạng cấu trúc và để đơn giản một năm bất kỳ được tính 365 ngày và tháng bất kỳ có 30 ngày.

27. Nhập một ngày tháng năm dưới dạng cấu trúc. Tính chính xác (kể cả năm nhuận) số ngày đã qua kể từ ngày 1/1/1 cho đến ngày đó.

28. Tính khoảng cách giữa 2 ngày tháng bất kỳ.

29. Hiện thứ của một ngày bất kỳ nào đó, biết rằng ngày 1/1/1 là thứ hai.

30. Hiện thứ của một ngày bất kỳ nào đó, lấy ngày thứ hiện tại để làm chuẩn.

31. Viết chương trình nhập một mảng sinh viên, thông tin về mỗi sinh viên gồm họ tên và ngày sinh (kiểu cấu trúc). Sắp xếp mảng theo tuổi và in ra màn hình

32. Để biểu diễn số phức có thể sử dụng định nghĩa sau:

```
typedef struct {    float re, im; } sophuc;
```

Cần bổ sung thêm trường nào vào cấu trúc để có thể lập được một danh sách liên kết các số phức.

33. Để tạo danh sách liên kết, theo bạn sinh viên nào dưới đây khai báo đúng cấu trúc tự trỏ sẽ được dùng:

Sinh viên 1: struct SV {char ht[25]; int tuoi; struct SV *tiep;};

Sinh viên 2: typedef struct SV node; struct SV {char ht[25]; int tuoi; node *tiep;};

Sinh viên 3: typedef struct SV {char ht[25]; int tuoi; struct SV *tiep;} node;

A: Sinh viên 1 B: Sinh viên 2 C: Sinh viên 2 và 3 D: Sinh viên 1, 2 và 3

34. Lập danh sách liên kết chứa bảng chữ cái A, B, C … Hãy đảo phần đầu từ A .. M xuống cuối thành N, O, … Z, A, …M.

35. Viết chương trình tìm người cuối cùng trong trò chơi: 30 người xếp vòng tròn.

Đếm vòng tròn (bắt đầu từ người số 1) cứ đến người thứ 7 thì người này bị loại ra khỏi vòng. Hỏi người còn lại cuối cùng ?

36. Giả sử có danh sách liên kết mà mỗi nốt của nó lưu một giá trị nguyên. Viết chương trình sắp xếp danh sách theo thứ tự giảm dần.

37. Giả sử có danh sách liên kết mà mỗi nốt của nó lưu một giá trị nguyên được sắp giảm dần. Viết chương trình cho phép chèn thêm một phần tử vào danh sách sao cho danh sách vẫn được sắp giảm dần.

38. Tạo danh sách liên kết các số thực x_1, x_2, ..., xn. Gọi m là trung bình cộng: $m=\frac{x_1+x_2+...+x_n}{n}$.

Hãy in lần lượt ra màn hình các giá trị: m, x_1 - m, x_2 - m, ..., xn - m.

39. Sử dụng kiểu union để in ra byte thấp, byte cao của một số nguyên.

CHƯƠNG 4: XỬ LÝ NGOẠI LỆ

4.1. Xử lý ngoại lệ

Một Exception (ngoại lệ) là một vấn đề xuất hiện trong khi thực thi một chương trình. Một Exception trong C++ là một phản hồi về một tình huống ngoại lệ mà xuất hiện trong khi một chương trình đang chạy, ví dụ như chia cho số 0.

Exception cung cấp một cách để truyền điều khiển từ một phần của một chương trình tới phần khác. Exception Handling (Xử lý ngoại lệ) trong C++ được xây dựng dựa trên 3 từ khóa là: try, catch, và throw.

Những loại tình huống bất thường này được nằm trong cái được gọi là exceptions và C++ đã vừa tích hợp ba toán tử mới để xử lý những tình huống này: **try**, **throw** và **catch**.

- throw: Một chương trình ném một Exception khi một vấn đề xuất hiện. Việc này được thực hiện bởi sử dụng từ khóa throw trong C++
- catch: Một chương trình bắt một Exception với một Exception Handler tại vị trí trong một chương trình nơi bạn muốn xử lý vấn đề đó. Từ khóa catch trong C++ chỉ dẫn việc bắt một exception
- try: Một khối try có thể được bắt bởi một số lượng cụ thể exception. Nó được theo sau bởi một hoặc nhiều khối catch

Giả sử một khối sẽ tạo một Exeption, một phương thức bắt một exception bởi sử dụng kết hợp các từ khóa try và catch. Một khối try/catch được đặt xung quanh code mà có thể tạo một exception. Code bên trong một khối try/catch được xem như là code được bảo vệ, và cú pháp để sử dụng try/catch trong C++ như sau:

Dạng thức sử dụng như sau:

```
try

{

// protected code
}catch( ExceptionName e1 )
{
   // catch block
}catch( ExceptionName e2 )
{
   // catch block
}catch( ExceptionName eN )
```

```
{

}

// catch block
```

Bạn có thể liệt kê nhiều lệnh catch để bắt các kiểu exception khác nhau trong trường hợp khối try của bạn xuất hiện nhiều hơn một exception trong các tình huống khác nhau.

4.2. Ném Exception trong C++

Exception có thể bị ném ở bất cứ đâu bên trong một khối code bởi sử dụng các lệnh throwtrong C++. Toán hạng của lệnh throw quyết định kiểu cho exception và có thể là bất kỳ biểu thức nào và kiểu kết quả của biểu thức quyết định kiểu của exception bị ném.

Ví dụ sau minh họa việc ném một exception khi chia cho số 0 trong C++:

```
double division(int a, int b)
{
if( b == 0 ) {

    throw "Division by zero condition!";
  }
  return (a/b);
}
```

4.3. Bắt Exception trong C++

Khối catch theo sau khối try trong C++ sẽ bắt bất kỳ exception nào. Bạn có thể xác định kiểu của exception bạn muốn bắt và điều này được xác định bởi khai báo exception mà xuất hiện trong các dấu ngoặc đơn theo sau từ khóa catch trong C++.

```
try

{

}catch( ExceptionName e )
{
  // code to handle ExceptionName exception
}
```

Code trên sẽ bắt một exception có kiểu là ExceptionName. Nếu bạn muốn xác định rằng một khối catch nến xử lý bất kỳ kiểu exception nào bị ném trong một khối try, bạn phải đặt một dấu ba chấm (...) trong các dấu ngoặc đơn theo sau từ khóa catch, như sau:

```
try

{

}catch(...)
{
```

```
// protected code
// code to handle any exception
}
```

Ví dụ sau ném một exception khi chia cho số 0 và chúng ta bắt nó trong khối catch

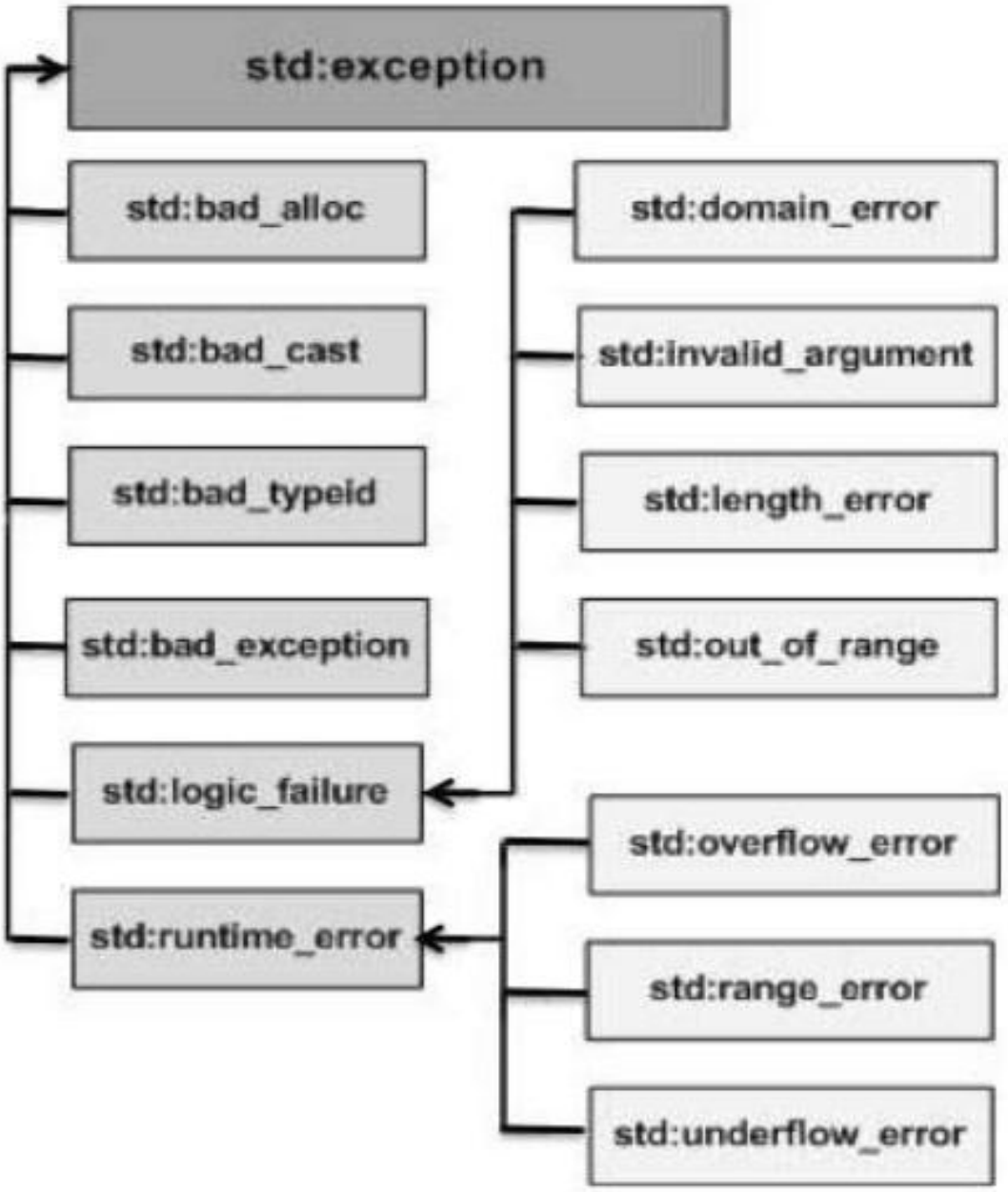
```
#include <iostream>
using namespace std;
double division(int a, int b)
{
if( b == 0 ) {

     throw "Division by zero condition!";
   }
   return (a/b);
}
int main ()

{  int x = 50;

   int y = 0;
   double z = 0;
   try {
     z = division(x, y);
cout << z << endl;
   }catch (const char* msg) {
     cerr << msg << endl;
   }
return 0; }
```

Bởi vì chúng ta đang tạo một exception có kiểu const char*, vì thế trong khi bắt exception này, chúng ta phải sử dụng const char* trong khối catch. Nếu chúng ta biên dịch và chạy code trên, nó sẽ cho kết quả sau:

Division by zero condition!

4.4.Chuẩn Exception trong C++

C++ cung cấp một danh sách các Standard Exception được định nghĩa trong <exception>mà chúng ta có thể sử dụng trong các chương trình. Những exception này được sắp xếp theo cấu trúc thứ tự cha-con như sau:



Bảng dưới là miêu tả ngắn gọn về mỗi exception được đề cập trong sơ đồ trên

std::bad_typeid	Có thể được ném bởi typeid
std::logic_error	Một exception mà theo lý thuyết có thể được phát hiện bởi việc đọc code
std::domain_error	Đây là một exception được ném khi một miền toán học không hợp lệ được sử dụng
std::invalid_argument	Được ném do các tham số không hợp lệ
std::length_error	Được ném khi một std::string quá lớn được tạo ra

std::out_of_range	Có thể được ném bởi một phương thức, ví dụ std::vector và std::bitset<>::operator[]().
std::runtime_error	Một exception mà theo lý thuyết không thể được phát hiện bởi việc đọc code
std::overflow_error	Được ném nếu một sự tràn luồng toán học (mathematical overflow) xuất hiện
std::range_error	Xuất hiện khi bạn cố gắng lưu giữ một giá trị bên ngoài dãy giá trị
std::underflow_error	Được ném nếu một mathematical underflow (sự tràn dưới) xuất hiện

4.5. Định nghĩa Exception mới trong C++

Bạn có thể định nghĩa các exception cho riêng bạn bằng việc kế thừa và ghi đè tính năng lớp exception trong C++. Ví dụ sau minh họa cách bạn có thể sử dụng lớp std::exception để triển khai exception của riêng bạn theo một cách chuẩn trong C++:

```
#include <iostream>
#include <exception>
using namespace std;
struct MyException : public exception
{
  const char * what () const throw ()
  {
    return "C++ Exception";
  }
};

int main() {

try

  {
    throw MyException();
  }
  catch(MyException& e)
  {
std::cout << "MyException caught" << std::endl;
    std::cout << e.what() << std::endl;
  }
  catch(std::exception& e)
```

```
  {
    //Other errors
} }
```

Nó sẽ cho kết quả sau:

```
MyException caught
C++ Exception
```

Ở đây, what() là một phương thức public được cung cấp bởi lớp exception trong C++ và nó đã được ghi đè bởi tất cả các lớp exception con. Ví dụ này trả về nguyên nhân của một exception trong C++

CHƯƠNG 5: MỘT SỐ VẤN ĐỀ

5.1. Một số quy tắc trong lập trình C++

5.1.1. Tổ chức chương trình

- Môđun hóa chương trình của bạn Chương trình của bạn nên được tách thành nhiều môđun, mỗi môđun thực hiện một công việc và càng độc lập với nhau càng tốt. Điều này sẽ giúp bạn dễ bảo dưỡng chương trình hơn và khi đọc chương trình, bạn không phải đọc nhiều, nhớ nhiều các đoạn lệnh nằm rải rác để hiểu được điều gì đang được thực hiện. Khi muốn chuyển thông tin cho các chương trình con, bạn nên sử dụng các tham số. Tránh sử dụng các biến toàn cục vì làm như vậy bạn sẽ triệt tiêu tính độc lập giữa các chương trình con và rất khó khăn khi kiểm soát giá trị của chúng khi chương trình thi hành. (Chú ý, bạn nên phân biệt giữa biến toàn cục và hằng số toàn cục)
- Định nghĩa và cài đặt của các lớp phải được chia thành nhiều file để ta có thể dễ dàng tái sử dụng. **Định nghĩa các lớp được lưu trong các file header với mở rộng *.h. Cài đặt của các thành viên của lớp lưu trong file nguồn với mở rộng *.cpp.** Thông thường mỗi lớp có một cặp file *.H và *.CPP, nhưng có thể gộp các lớp có liên quan vào một cặp file.Cuối mỗi file *.H là chỉ thị #include đến file *.CPP.Cuối mỗi file *.CPP là các "chương trình chính" dùng để test file CPP đó kèm theo các #define thích hợp cho việc test. Chương trình chính được lưu trong một file nguồn riêng và include các file header của các lớp được dùng đến.
- Mỗi file header của lớp nên sử dụng các định hướng #ifndef, #define, và #endif để đảm bảo mỗi file header chỉ được include 1 lần. Ký hiệu được #define nên là tên của file header viết toàn bằng chữ hoa với một dấu gạch dưới (_) thay cho dấu chấm.

  Ví dụ:

  ```
  //counter.h
  #ifndef COUNTER_H #define COUNTER_H
  class Counter {
  //...  }; // end Counter
  #include "counter.cpp" #endif // COUNTER_H
  ```

5.1.2. Chuẩn tài liệu

- Sử dụng // cho các chú thích. Chỉ dùng /* */ để tạm thời vô hiệu hóa các đoạn chương trình để test và debug.
- Mỗi file nguồn, cả .CPP và .H, đều phải bắt đầu bằng một khối chú thích đủ để người đọc có thể kết nối các file nếu chúng bị tách ra. Mẫu như sau:

```
/------------------------------------------------------------------
// Name: Họ tên
// Class: Lớp
// Project: mô tả/tên dự án (một dòng, giống nhau tại mọi file)
// Purpose: Mục đích sử dụng của mã chương trình hoặc các khai báo trong file này
//------------------------------------------------------------------
```

Mỗi lớp, hàm, phương thức phải có một khối chú thích mô tả ngắn gọn lớp, hàm, phương thức đó làm gì; đối với hàm/phương thức: liệt kê tất cả các tham số, nêu rõ ý nghĩa của tham số; và mô tả điều kiện trước và sau của hàm/phương thức đó.Chọn các tên có nghĩa sẽ đơn giản hóa các chú thích.
Lưu ý, tài liệu về phương thức đặt tại định nghĩa lớp (*.H) ta có thể sao chép tài liệu đó vào file *.CPP nhưng không bắt buộc.

- Có thể chú thích các đoạn code bên trong hàm, tuy nhiên chỉ nên chú thích đủ hiểu. Quá nhiều chú thích và chú thích thừa làm code trông rối. Tất cả các chú thích phải được lùi đầu dòng cùng đoạn code quanh nó.

5.1.3. Tên

- Sử dụng các tên có nghĩa.**Tên giàu tính mô tả cho các biến toàn cục và tên ngắn gọn cho các biến cục bộ.**Tên có nghĩa sẽ giúp chương trình dễ viết và dễ debug hơn. Nếu bạn phải dùng tên không có nghĩa cho một cái gì đó thì có thể bạn chưa hoàn toàn hiểu bài toán mình đang giải. Hãy cố hiểu rõ trước khi tiếp tục lập trì Theo thông lệ, các tên i và j được dành cho các chỉ số, p và q dành cho các con trỏ, s và t dành cho các xâu.Người ta dùng các tên bắt đầu hoặc kết thúc bởi chữ “p” cho các biến con trỏ (chẳng hạn nodep, intp, intpp, doublep), các tên bắt đầu bằng chữ hoa cho biến toàn cục (chẳng hạn Globals) và tất cả chữ cái hoa cho các hằng số (chẳng hạn CONSTANTS). Khuyến cáo sử dụng tên tiếng Anh kiểu camel (xem bên dưới)
  Các namespace trong C++ góp phần làm rõ nghĩa của các tên mà không cần sử dụng các tên dài.

- Đặt tên một cách nhất quán

Các biến có liên quan phải được đặt các tên có liên quan, đồng thời phải làm nổi bật được sự khác nhau của chúng.

Các tên trong lớp sau đây vừa quá dài vừa không hề nhất quán:

```
class
UserQueue
{
public int noOfUsersInQueue() {...} };
```

Thứ nhất, cùng một nội dung là queue nhưng được biểu diễn bởi ba dấu hiệu: Q, Queue và queue. Thứ hai, các biến và các hàm thành phần của lớp UserQueue chỉ có thể được sử dụng bởi các đối tượng của lớp này, do vậy viết
queue.queueCapacity hay
queue.noOfUsersInQueue() rõ ràng là thừa. Chúng ta có thể viết lại lớp này với các tên mới như sau:

```
class UserQueue {
int nitems, front, capacity;
public int nusers() {...} }
```

Không chỉ bản thân đoạn mã định nghĩa lớp đó dễ hiểu hơn, mà những đoạn mã sử dụng lớp UserQueue cũng dễ hiểu hơn:
queue.capacity++; n = queue.nusers();

Lớp UserQueue vẫn có thể cải tiến thêm, bởi nitems và nusers thực chất là cùng biểu diễn một khái niệm và do đó chỉ cần sử dụng một trong hai tên đó mà thôi.

- Tên của các project, form, và component sinh bởi môi trường lập trình: các project và form phải có tên hợp lý, không để nguyên là Form1. Các component phải được đặt tên có nghĩa, ngoại trừ các component như Label, Group Box, etc., nếu chúng không có mặt trong code. Các component nên được đặt hậu tố là kiểu đối tượng:

Ex: widthScale, nameText, leftScrollbar, mainForm, myLabel, printerDialog

- Tên biến và tên hàm:
    - Thường phải là các từ hoặc cụm từ. Ngoại lệ duy nhất: con đếm vòng for() đôi khi có thể chỉ cần dùng tên là 1 chữ cái chẳng hạn i
    - không viết tắt trừ các từ viết tắt thông dụng chẳng hạn HTML, khi đó coi từ viết tắt như từ thông thường (tên sẽ có dạng convertToHtml thay vì convertToHTML)
    - Đặt tên cho các namespace nên bằng chữ in thường toàn bộ: Ex:

mynamespace, com.company.application.ui

- Tên biến là một danh từ bắt đầu bằng một ký tự in thường, các từ tiếp theo được bắt đầu bằng một ký tự in hoa:

  Ex: line, audioSystem...

- Đặt các tên "động" cho hàm:

  Tên hàm nên là một động từ theo sau bởi một danh từ. Ví dụ: now = date.getTime();

  Các hàm trả về giá trị boolean nên được đặt tên thể hiện giá trị mà nó trả về. Ví dụ: isOctal(c) thì tốt hơn là: checkOctal(c);

  vì cách đặt tên thứ nhất cho biết ngay rằng hàm trả về giá trị true nếu c là một số octal và trả về false trong trường hợp ngược lại.

- Tên hàm thể hiện chức năng Các tiền tố thường được sử dụng: get/set, add/remove, create/destroy, start/stop, insert/delete, increment/decrement, old/new, begin/end, first/last, up/down, min/max, next/previous, old/new, open/close, show/hide, suspend/resume ...

- Tên class (và struct):

  Dùng chữ hoa tất cả các chữ cái đầu mỗi từ, còn lại là các chữ cái thường. Ví dụ: GameBoard, Game.

5.1.4. Định dạng

- Lùi đầu dòng các đoạn code, mỗi mức dùng 3 hoặc 4 ký tự, tốt nhất là dùng tab.
  - Phải thống nhất, luôn dùng 3 hoặc luôn dùng 4 ký tự
  - Chú ý không được dùng lẫn lộn giữa ký tự tab và space để lùi đầu dòng,
- Mỗi dòng chỉ chứa nhiều nhất 1 lệnh và không dài quá 79 ký tự. Một lệnh có thể được chia thành nhiều dòng, khi đó các phần sau phải được lùi đầu dòng hợp lý.

  Ví dụ :

```
cout << "The cost for 1 loaf of bread" << endl
        << "is $1.89" << endl;
```

- Có thể căn thẳng hàng để nâng cao hightlight.

VD:

```
int songuyen  = 100;
double sothuc         = 3.14
```

```
char kytu          = 'a' ;
char ten[]         = { "tam", "lan", "hiep", "bao", "yen", "tuan", "hoa" };
bool gt[]          = { 0 , 0 , 0 , 1 , 0 , 1 , 0 };
```

- Các khối với cặp ngoặc {} phải được trình bày 1 trong 2 cách sau:

```
if (!done)
{
doSomething(); moreToDo();
} else {
//... }
if(!done) { doSomething();
moreToDo(); } else{
//... }
```

Nếu trong khối chỉ có 1 lệnh thì có thể bỏ ngoặc (tốt hơn là không nên) nhưng vẫn lùi đầu dòng:

```
for( n++; n < 100; n++ )
        field[ n ] = 0;
*i = 0;
return ‘n’;
```

- Nên có khoảng trắng giữa từ khóa và dấu ‘(’, nhưng không nên có khoảng trắng giữa tên hàm và dấu ‘(‘. Ví dụ:

```
// no space between 'strcmp' and '(',
// but space between 'if' and '('
if ( strcmp( input_value, "done" ) == 0 )
return 0;
```

Ngoài ra nên có khoảng giữa dấu ngoặc của hàm và đối số như trên.

- Nên có 1 space trước và sau mỗi toán tử đôi số học hoặc lôgic, chẳng hạn +, <<, và ||. Nên dùng 1 space sau dấu phẩy, nhưng trước dấu phẩy hoặc chấm phẩy không nên có dấu cách.Ngoại lệ: không chèn khoảng trắng vào giữa toán hạng và toán tử ++ và –-
- Chèn dòng trắng giữa các đoạn khác nhau trong chương trình.

5.1.5. Thiết kế

Một số vấn đề thường gặp mà sinh viên cần chú ý:

- Không để dữ liệu của lớp dạng public.

- Hạn chế tối đa việc dùng biến toàn cục.
- Nguyên tắc quyền ưu tiên tối thiểu: chỉ cho hàm đủ quyền truy nhập để thực hiện nhiệm vụ của mình, không cho nhiều quyền hơn. const được sử dụng cho một biến khi hàm không cần thay đổi biến được tham chiếu đến đó.

  Nếu hàm không được sửa giá trị của một tham số, chỉ truyền tham số vào là giá trị đối với các kiểu đơn giản, mảng phải được truyền dưới dạng "const []", và các kiểu struct/class nên truyền dạng "const &" hoặc "const *". • Không để dữ liệu trong lớp mà nó không thực sự thuộc về lớp đó. Chẳng hạn, nếu một hàm cần một biến để lưu một kết quả tạm thời, không khai báo một biến thuộc lớp mà hãy khai báo một biến địa phương của hàm đó. • Các hàm hoặc phương thức của lớp không nên tạo output trừ khi đó là nhiệm vụ của phương thức đó. Ví dụ, một phương thức print có thể sẽ ghi thông tin ra cout, nhưng một thao tác để thêm hoặc bớt cái gì đó từ một danh sách thì không.
- Mỗi phương thức/hàm (kể cả hàm main()) chỉ chứa tối đa 30 dòng kể cả tính từ ngoặc mở hàm "{" tới ngoặc kết thúc hàm "}".

5.1.6. Code

- **Viết code theo chuẩn ISO dù compiler có bắt buộc hay không.**
  - Hạn chế #include ngoài chuẩn, VD: conio.h
  - Nên để int main()và return 0;thay vì void main()
  - Khai báo using std:: ngay cả khi IDE không bắt buộc. (dùng using namespace std; không tốt lắm)

- **Các hằng số không nên viết trực tiếp vào chương trình.**

Thay vì thế, người ta thường sử dụng lệnh #define hay const để đặt cho những hằng số này những tên có ý nghĩa. Điều này sẽ giúp lập trình viên dễ kiểm soát những chương trình lớn vì giá trị của hằng số khi cần thay đổi thì chỉ phải thay đổi một lần duy nhất ở giá trị định nghĩa (ở #define hay const).

Ví dụ:

```
popChange = (0.1758 - 0.1257) * population; nên được viết là:
const double BIRTH_RATE = 0.1758, DEATH_RATE = 0.1257; hoặc:
        #define BIRTH_RATE 0.1758
        #define DEATH_RATE 0.1257
        //...
popChange = (BIRTHRATE - DEATH_RATE) * population;
```

Ghi chú: bạn không nên dùng #define thường xuyên để định nghĩa các hằng số, bởi vì trong quá trình debug, bạn sẽ không thể xem được giá trị của một hằng số định nghĩa bằng #define.

Khi làm việc với các kí tự, hãy sử dụng các hằng kí tự thay vì các số nguyên. Ví dụ để kiểm tra xem c có phải một chữ cái hoa hay không, có thể dùng đoạn mã sau:

```
if( c >= 65 && c <= 90 ) ...
```

Nhưng đoạn mã này hoàn toàn phụ thuộc vào bộ mã biểu diễn kí tự đang được sử dụng. Cách tốt hơn là viết như sau:

```
if( c >= 'A' && c <= 'Z' ) ...
```

- **Luôn viết new và delete thành từng cặp.**
- **Khi khai báo con trỏ, dấu con trỏ nên được đặt liền với tên, nhằm tránh trường hợp sau:**

```
char* p, q, r; // q, r không là con trỏ
```

Trong trường hợp này nên viết là:

```
har *p, *q, *r;
```

(luật này cũng được dùng khi khai báo tham chiếu với dấu &)

- **Nên sử dụng các dấu ( ) khi muốn tránh các lỗi về độ ưu tiên toán tử.**

Ví dụ:

```
// No!
int i = a >= b && c < d && e <= g + h;
// Better
int j = (a >= b) && (c < d) && (e <= (g + h));
```

- **Tách các biểu thức phức tạp thành các biểu thức đơn giản hơn**

Biểu thức sau đây rất ngắn gọn nhưng lại chứa quá nhiều phép toán:

```
*x += ( *xp = ( 2*k < ( n – m ) ? c[ k + 1 ] : d[ k — ] ) );
```

Chúng ta nên viết lại như sau:

```
if( 2*k < n – m )
        *xp = c[ k + 1 ];
else
        *xp = d[ k– ];
        *x += *xp;
```

- **Viết các lệnh dễ hiểu, không viết các lệnh "khôn ngoan"**

Các lập trình viên thường thích viết các lệnh càng ngắn gọn càng tốt. Tuy nhiên điều này thường gây phiền toái cho người khác.

Hãy xem biểu thức sau đây làm gì:

```
subkey = subkey >> ( bitoff – ( ( bitoff >> 3 ) << 3 ) );
```

Biểu thức trong cùng (bitoff >> 3) dịch phải bitoff 3 bit. Kết quả thu được lại được dịch trái 3 bit. Bởi vậy 3 bit cuối cùng của bitoff được thay thế bởi các số 0. Kết quả này lại được trừ đi bởi giá trị ban đầu của bitoff, kết quả của phép trừ chính là 3 bit cuối cùng trong giá trị ban đầu của bitoff. Ba bit này được dùng để dịch subkey sang phải.

Bởi vậy, biểu thức nói trên tương đương với biểu thức sau đây:

```
subkeu = subkey >> ( bitoff & 0×7 );
```

Rõ ràng cách viết thứ hai dễ hiểu hơn nhiều. Một ví dụ khác về cách viết biểu thức ngắn gọn nhưng làm phức tạp hóa vấn đề:

```
child = ( ! LC && ! RC ) ? 0 : ( ! LC ? RC : LC );
```

Cách viết dưới đây dài hơn, nhưng dễ hiểu hơn nhiều:

```
if( LC == 0 && RC == 0 )
        child = 0;
else if( LC == 0 )
        child = RC;
else
        child = LC;
```

toán tử ? : chỉ thích hợp cho những biểu thức ngắn kiểu như sau đây:

```
max = ( a > b ) ? a : b;
```

hoặc: `printf( “The list has %d item%s\n”, n, n == 1 ? “” : “s” );`

- **Cẩn thận với dấu =**

= và == là 2 toán tử gây nhần lẫn nhất trên C, nhưng bạn có thể tránh gặp nó bằng thói quen viết r-value (biểu thức bên phải phép gán) sang bên trái phép so sánh:

```
if ( a == 42 ) { ... }// Cách viết thông thường.
if ( 42 == a ) { ... }// Nên viết thế này.
```

Và đây là sự khác biệt, khi bạn nhầm ...

```
if ( a = 42 ) { ... }// Chạy bình thường, khó tìm ra lỗi
if ( 42 = a ) { ... }// Báo lỗi ngay chỗ này
```

- **Các idiom**

Cũng giống như ngôn ngữ tự nhiên, ngôn ngữ lập trình cũng có các idiom (thành ngữ !?), là các cách viết code chính tắc cho các trường hợp thông dụng, tạm hiểu idiom là các chuẩn không bắt buộc nhưng được đa số người dùng tuân theo. Sử dụng các idiom giúp giảm bớt khả năng mắc lỗi đồng thời làm chương trình dễ đọc hơn và nhất là có vẻ “chuyên nghiệp” hơn Sau đây là một số idiom phổ biến:

- **Các idiom cho mảng**

Để duyệt qua n phần tử của một mảng và khởi tạo chúng, có các cách viết sau đây:

```
i = 0;  while ( i <= n – 1 )
        array[ i++ ] = 1.0;
```

hoặc

```
for( i = 0; i < n; )
        array[ i++ ] = 1.0;
```

hoặc

```
 for( i = n; --i >= 0; )
        array[ i ] = 1.0;
```

Tất cả những cách viết trên đều đúng, tuy nhiên idiom cho trường hợp này là:

```
for( i = 0; i < n; ++i )
        array[ i ] = 1.0;
```

Một lưu ý nhỏ là sự khác biệt giữa i++ và ++i: • i++ lấy giá trị của i trước rồi tăng nó lên. • ++i tăng giá trị của i rồi lấy giá trị mới.

Do đó đối với các con đếm vòng lặp (for(),while()) nên dùng ++i để tăng tốc độ.

Idiom của vòng lặp duyệt qua các phần tử của một danh sách (list) là for(p = list; p != NULL; p = p->next)

Đối với container:

```
vector<string>::iterator it;
for(it = v.begin(); it != v.end(); ++it)
std::cout << *it;
```

Đối với các vòng lặp vô hạn, idiom là: for (; ;) hoặc while(1)

Khởi tạo danh sách:

```
struct info {
char *name;
char *job;
char *address;
};
info *array[] = {
        { "name1", "job1", "add1" },
        { "name1", "job1", "add1" },
        { "name1", "job1", "add1" }, //...
};
```

Hàm tìm kiếm tuyến tính:

```
template <class T>
int find (T obj,T* array,int size,int from = 0) {
        for(int i = from; i<size; ++i)
                if(array[i] == T) return i;
        return size;
}
```

Sao chép mảng:

Giả sử 2 mảng double *a,*b; thay vì:

```
for(int i=0; i<n; ++i)
        b[i]=a[i];
```

ta có thể dùng:

```
//#include<string.h>
memcpy(b,a,n*sizeof(double));
```

Cấp phát động cho mảng 2 chiều:

```
nt **pp = new type*[n];
int *p = new type[n*m];
        for (int i = 0; i < n; ++i)
                pp[i] = p + i * m;
                //...
                //use array here
        delete[] p;
        delete[] pp;
```

- **Idiom cho lệnh if**

Tiếp theo là một idiom dành cho câu lệnh if. Hãy xem đoạn mã loằng ngoằng sau đây làm gì

```
if ( argc==3 )
        if ( ( fin = fopen(argv[l] , “r” ) ) != NULL )
                if ( ( fout = fopen( argv[2], “w” ) ) != NULL ) {
                while ( ( c = getc( fin ) ) != EOF )
                        putc( c, fout );
                        fclose( fin );
                        fclose( fout );
                        } else
                        printf ( “Can’t open output file %s\n”, argv[2] ) ;
                        else
                        printf( “Can’t open input file %s\n”, argv[l] ) ;
```

```
        else
        printf ( “Usage: cp input file outputfile\n” ) ;
```

Viết lại đoạn mã này theo đúng idiom như sau:

```
if ( argc != 3 )
    printf ( “Usage: cp input file outputfile\n” ) ;
else if ( ( fin = fopen( argv[l] , “r” ) ) == NULL )
    printf( “Can’t open input file %s\n”, argv[l] );
else if ( ( fout = fopen( argv[2], “w” ) ) == NULL )
{
    printf ( “Can’t open output file %s\n”, argv[2] ) ;
    fclose( fin ) ;
    } else
{
while ( ( c = getc( fin ) ) != EOF)
    putc( c, fout );
fclose( fin ) ;
fclose( fout ) ;
}
```

Nguyên tắc khi viết các lệnh if() là đặt các phép toán kiểm tra điều kiện càng gần các hành động tương ứng càng tốt.

- **Idiom cho switch() case:**

Xét ví dụ:

```
switch (c) {
case '-': sign = -1;
case '+': c = getchar();
case '.': break;
case '0': case 'o': default: if (!isdigit(c)) return 0;
}
```

cách viết sau tuy dài nhưng dễ đọc hơn:

```
switch (c) {
case '-':
    sign = -1;
case '+':
    c = getchar();
    break;
```

```
case '.':
        break;
default: case '0': case 'o':
        if (!isdigit(c))
        return 0;
        break;
}
```

- **Số 0 trong chương trình**

Số 0 thường xuyên xuất hiện trong các chương trình với nhiều ý nghĩa khác nhau. Trình dịch sẽ tự động chuyển số 0 thành kiểu thích hợp. Tuy nhiên nên viết ra một cách tường minh bản chất của số 0 mà chúng ta đang nói đến. Cụ thể, hãy sử dụng (void*)0 hoặc NULL để biểu diễn con trỏ null trong C, sử dụng '\0′ cho kí tự null ở cuối mỗi xâu và sử dụng 0.0 cho các số float hoặc double có giá trị không. Đừng viết đoạn mã như sau

```
p = 0;
name[ i ] = 0;
x = 0;
```

Hãy viết:

```
p = NULL;
name[ i ] = '\ 0';
x = 0.0;
```

Số 0 nên để dành cho các số nguyên có giá trị bằng không. Tuy nhiên trong C++, 0 (thay vì NULL) lại được sử dụng rộng rãi cho các con trỏ null, điều này không được khuyến khích.

5.2. Namespaces

Namespaces cho phép chúng ta gộp một nhóm các lớp, các đối tượng toàn cục và các hàm dưới một cái tên. Nói một cách cụ thể hơn, chúng dùng để chia phạm vi toàn cụ thành những phạm vi nhỏ hơn với tên gọi namespaces.

Khuông mẫu để sử dụng namespaces là:

```
namespace identifier
{
  namespace-body
}
```

Trong đó identifier là bất kì một tên hợp lệ nào và namespace-body là một tập hợp những lớp, đối tượng và hàm được gộp trong namespace. Ví dụ:

```
namespace general
{
int a, b;
}
```

Trong trường hợp này, a và b là những biến bình thường được tích hợp bên trong namespace general. Để có thể truy xuất vào các biến này từ bên ngoài namespace chúng ta phải sử dụng toán tử ::. Ví dụ, để truy xuất vào các biến đó chúng ta viết:

general::a

general::b

Namespace đặc biệt hữu dụng trong trường hợp có thể có một đối tượng toàn cục hoặc một hàm có cùng tên với một cái khác, gây ra lỗi định nghĩa lại. Ví dụ:

```
// namespaces
#include <iostream.h>
namespace first
{
int var = 5;
}
namespace second
{
double var = 3.1416;
}
int main () {
cout << first::var << endl;
cout << second::var << endl;
return 0;
}
```

kết quả:

```
5
3.1416
```

Trong ví dụ này có hai biến toàn cục cùng có tên var, một được định nghĩa trong namespace first và cái còn lại nằm trong second. Chương trình vẫn chạy tốt,

5.1.1. Using namespace

Chỉ thị using theo sau là namespace dùng để kết hợp mức truy xuất hiện thời với một namespace cụ thể để các đối tượng và hàm thuộc namespace có thể được truy xuất trực tiếp như thể chúng được khai báo toàn cục. Cách sử dụng như sau:

using namespace identifier;

Ví dụ:

```
// using namespace example
#include <iostream.h>
namespace first
{
int var = 5;
}
namespace second
{
double var = 3.1416;
}
int main () {
using namespace second;
cout << var << endl;
cout << (var*2) << endl;
return 0;
}
```

Kết quả:

```
3.1416
6.2832
```

Trong trường hợp này chúng ta có thể sử dụng var mà không phải đặt trước nó bất kì toán tử phạm vi nào.

Bạn phải để ý một điều rằng câu lệnh using namespace chỉ có tác dụng trong khối lệnh mà nó được khai báo hoặc trong toàn bộ chương trình nếu nó được dùng trong phạm vi toàn cục. Ví dụ, nếu chúng ta định đầu tiên sử dụng một đối tượng thuộc một namespace và sau đó sử dụng một đối tượng thuộc một namespace khác chúng ta có thể làm như sau:

```
// using namespace example
#include <iostream.h>
namespace first
{
int var = 5;
}
namespace second
{
```

```
double var = 3.1416;
}
int main () {
{
using namespace first;
cout << var << endl;
}
{
using namespace second;
cout << var << endl;
}
return 0;
}
```

Kết quả:

```
5
3.1416
```

5.1.2. Định nghĩa bí danh

Chúng ta cũng có thể định nghĩa những tên thay thế cho các namespaces đã được khai báo. Cách thức để làm việc này như sau:

```
namespace new_name = current_name ;
```

5.1.3. Namespace std

Một trong những ví dụ tốt nhất mà chúng ta có thể tìm thấy về namespaces chính là bản thân thư viện chuẩn của C++. Theo chuẩn ANSI C++, tất cả định nghĩa của các lớp, đối tượng và hàm của thư viện chuẩn đều được định nghĩa trong namespace std.

Bạn có thể thấy rằng chúng ta đã bỏ qua luật này trong suốt tutorial này. Tôi đã quyết định làm vậy vì luật này cũng mới như chuẩn ANSI (1997) và nhiều trình biên dịch cũ không tương thích với nó.

Hầu hết các trình biên dịch, thậm chí cả những cái tuân theo chuẩn ANSI, cho phép sử dụng các file header truyền thống (như là iostream.h, stdlib.h), những cái mà chúng ta trong suốt tutorial này. Tuy nhiên, chuẩn ANSI đã hoàn toàn thiết kế lại những thư viện này để tận dụng lợi thế của tính năng templates và để tuân theo luật phải khai báo tất cả các hàm và biến trong namespace std.

Chuẩn ANSI đã chỉ định những tên mới cho những file "header" này, cơ bản là dùng cùng tên với các file của chuẩn C++ nhưng không có phần mở rộng .h. Ví

dụ, iostream.h trở thành iostream.

Nếu chúng ta sử dụng các file include của chuẩn ANSI-C++ chúng ta phải luôn nhớ rằng tất cả các hàm, lớp và đối tượng sẽ được khai báo trong std. Ví dụ:

```
// ANSI-C++ compliant hello world
#include <iostream>
int main () {
std::cout << "Hello world in ANSI-C++\n";
return 0;
}
```

KQ:

```
Hello world in ANSI-C++
```

Mặc dù vậy chúng ta nên sử dụng using namespace để khỏi phải viết toán tử :: khi tam chiếu đến các đối tượng chuẩn:

```
// ANSI-C++ compliant hello world (II)
#include <iostream>
using namespace std;
int main () {
cout << "Hello world in ANSI-C++\n";
return 0;
}
```

Kết quả:

```
Hello world in ANSI-C+
```

Tên của các file C cũng có một số thay đổi. Bạn có thể tìm thêm thông tin về tên mới của các file header chuẩn trong tài liệu Các file header chuẩn.

TÀI LIỆU THAM KHẢO

[1] Phạm Văn Ất, *Kỹ thuật lập trình C*, NXB Thống kê, 2003

[2] Lê Hoài Bắc, Nguyễn Thanh Nghị, *Kỹ năng lập trình, nhà xuất bản Khoa học và kỹ thuật – Hà Nội, 2005*

[3] Đinh Mạnh Tường, *Cấu trúc dữ liệu và giải thuật*, NXB Giáo dục, 2002