C++ và lập trình hướng đối tượng

Bởi:

Phạm Văn Ất

Làm việc với TC++ 3.0

Các ví dụ trong cuốn sách này sẽ viết và thực hiện trên môi trường TC++ 3.0. Bộ cài đặt TC++ 3.0 gồm 5 đĩa. Sau khi cài đặt (giả sử vào thư mục C:\TC) thì trong thư mục TC sẽ gồm các thư mục con sau:

C:\TC\BGI chứa các tệp đuôi BGI và CHR

C:\TC\BIN chứa các tệp chương trình (đuôi EXE) như TC, TCC, TLIB, TLINK

C:\TC\INCLUDE chứa các tệp tiêu đề đuôi H

C:\TC\LIB chứa các tệp đuôi LIB, OBJ

Để vào môi trường của TC++ chỉ cần thực hiện tệp chương trình TC trong thư mục C:\TC\BIN . Kết quả nhận được hệ menu chính của TC++ với mầu nền xanh gần giống như hệ menu quen thuộc của TC (Turbo C). Hệ menu của TC++ gồm các menu: File, Edit, Search, Run, Compile, Debug, Project, Options, Window, Help.

Cách soạn thảo, biên dịch và chạy chương trình trong TC++ cũng giống như trong TC, ngoại trừ điểm sau: Tệp chương trình trong hệ soạn thảo của TC++ có đuôi mặc định là CPP còn trong TC thì tệp chương trình luôn có đuôi C.

Trong TC++ có thể thực hiện cả chương trình C và C++. Để thực hiện chương trình C cần dùng đuôi C để đặt tên cho tệp chương trình, để thực hiện chương trình C++ cần dùng đuôi CPP để đặt tên cho tệp chương trình.

C và C++

- Có thể nói C++ là sự mở rộng (đáng kể) của C. Điều đó có nghĩa là mọi khả năng, mọi khái niệm trong C đều dùng được trong C++.

- Vì trong C++ sử dụng gần như toàn bộ các khái niệm, định nghĩa, các kiểu dữ liệu, các cấu trúc lệnh, các hàm và các công cụ khác của C, nên yêu cầu bắt buộc đối với các đọc giả C++ là phải biết sử dụng tương đối thành thạo ngôn ngữ C.

- Vì C++ là sự mở rộng của C, nên bản thân một chương trình C đã là chương trình C++ (chỉ cần thay đuôi C bằng đuôi CPP). Tuy nhiên Trình biên dịch TC++ yêu cầu mọi hàm chuẩn dùng trong chương trình đều phải khai báo nguyên mẫu bằng một câu lệnh #include, trong khi điều này không bắt buộc đối với Trình biên dịch của TC.

Trong C có thể dùng một hàm chuẩn mà bỏ qua câu lệnh #include để khai báo nguyên mẫu của hàm được dùng. Điều này không báo lỗi khi biên dịch, nhưng có thể dẫn đến kết quả sai khi chạy chương trình.

Khi biên dịch chương trình sau trong môi trường C sẽ không gặp các dòng cảnh báo (Warning) và thông báo lỗi (error). Nhưng khi chạy sẽ nhận được kết quả sai.

```
#include <stdio.h> void main() { float a,b,c,p,s;
printf("\nNhap a, b, c "); scanf("%f%f%f",&a,&b,&c);
p=(a+b+c)/2; s= sqrt(p*(p-a)*(p-b)*(p-c)); printf("\nDien
tich = %0.2f",s); getch(); }
```

Nếu biên dịch chương trình này trong TC++ sẽ nhận được các thông báo lỗi sau:

```
Eror: Funtion ‘sqrt’ should have a prototype

Eror: Funtion ‘getch’ should have a prototype
```

Để biến chương trình trên thành một chương trình C++ cần:

+ Đặt tên chương chường với đuôi CPP

+ Thêm 2 câu lệnh #include để khai báo nguyên mẫu cho các hàm sqrt, getch:

```
#include <math.h>

#include <conio.h>
```

Lập trình cấu trúc và lập trình hướng đối tượng

Phương pháp lập trình cấu trúc

- Tư tưởng chính của lập trình cấu trúc là tổ chức chương trình thành các chương trình con. Trong PASCAL có 2 kiểu chương trình con là thủ tục và hàm. Trong C chỉ có một loại chương trình con là hàm.

Hàm là một đơn vị chương trình độc lập dùng để thực hiện một phần việc nào đó như: Nhập số liệu, in kết quả hay thực hiện một số tính toán. Hàm cần có đối và các biến, mảng cục bộ dùng riêng cho hàm.

Việc trao đổi dữ liệu giữa các hàm thực hiện thông qua các đối và các biến toàn bộ.

Các ngôn ngữ như C, PASCAL, FOXPRO là các ngôn ngữ cho phép triển khai phương pháp lập trình cấu trúc.

Một chương trình cấu trúc gồm các cấu trúc dữ liệu (như biến, mảng, bản ghi) và các hàm, thủ tục.

Nhiệm vụ chính của việc tổ chức thiết kế chương trình cấu trúc là tổ chức chương trình thành các hàm, thủ tục: Chương trình sẽ bao gồm các hàm, thủ tục nào.

Xét yêu cầu sau: Viết chương trình nhập toạ độ (x,y) của một dẫy điểm, sau đó tìm một cặp điểm cách xa nhau nhất.

Trên tư tưởng của lập trình cấu trúc có thể tổ chức chương trình như sau:

+ Sử dụng 2 mảng thực toàn bộ x và y để chứa toạ độ dẫy điểm

+ Xây dựng 2 hàm:

Hàm nhapsl dùng để nhập toạ độ n điểm, hàm này có một đối là biến nguyên n và được khai báo như sau:

```
void nhapsl(int n);
```

Hàm do_dai dùng để tính độ dài đoạn thẳng đi qua 2 điểm có chỉ số là i và j , nó được khai báo như sau:

```
float do_dai(int i, int j);
```

Chương trình C cho bài toán trên được viết như sau:

```
#include <stdio.h> #include <conio.h> #include <math.h>
float x[100],y[100]; float do_dai(int i, int j) { return
```

```
sqrt(pow(x[i]-x[j],2)+pow(y[i]-y[j],2)); } void nhapsl(int
n) { int i; for (i=1;i<=n;++i) { printf("\nNhap toa do x,
y cua diem thu %d : ",i); scanf("%f%f",&x[i],&y[i]); } }
void main() { int n,i,j,imax,jmax; float d,dmax;
printf("\nSo diem N= "); scanf("%d",&n); nhapsl(n);
dmax=do_dai(1,2); imax=1;jmax=2; for (i=1;i<=n-1;++i) for
(j=i+1;j<=n;++j) { d=do_dai(i,j); if (d>dmax) { dmax=d;
imax=i; jmax=j; } } printf("\nDoan thang lon nhat co do
dai bang: %0.2f",dmax); printf("\n Di qua 2 diem co chi so
la %d va %d",imax,jmax); getch(); }
```

Phương pháp lập trình hướng đối tượng

+ Khái niệm trung tâm của lập trình hướng đối tượng là lớp (class). Có thể xem lớp là sự kết hợp các thành phần dữ liệu và các hàm. Cũng có thể xem lớp là sự mở rộng của cấu trúc trong C (struct) bằng cách đưa thêm vào các phương thức (method) hay còn gọi là hàm thành viên (member function). Một lớp được định nghĩa như sau:

```
class Tên_Lớp { // Khai báo các thành phần dữ liệu // Khai
báo các phương thức };
```

+ Các phương thức có thể được viết (xây dựng) bên trong hoặc bên ngoài (phía dưới) phần định nghiã lớp. Cấu trúc (cách viết) phương thức tương tự như hàm ngoại trừ quy tắc sau: Khi xây dựng một phương thức bên ngoài định nghĩa lớp thì trong dòng đầu tiên cần dùng tên lớp và 2 dấu : đặt trước tên phương thức để chỉ rõ phương thức thuộc lớp nào (xem ví dụ bên dưới).

+ Sử dụng các thành phần dữ liệu trong phương thức: Vì phương thức và các thành phần dữ liệu thuộc cùng một lớp và vì phương thức được lập lên cốt để xử lý các thành phần dữ liệu, nên trong thân của phương thức có quyền truy nhập đến các thành phần dữ liệu (của cùng lớp).

+ Biến lớp: Sau khi định nghĩa một lớp, có thể dùng tên lớp để khai báo các biến kiểu lớp hay còn gọi là đối tượng. Mỗi đối tượng sẽ có các thành phần dữ liệu và các phương thức. Lời gọi một phương thức cần chứa tên đối tượng để xác định phương thức thực hiện từ đối tượng nào.

+ Một chương trình hướng đối tượng sẽ bao gồm các lớp có quan hệ với nhau.

+ Việc phân tích, thiết kế chương trình theo phương pháp hướng đối tượng nhằm thiết kế, xây dựng các lớp.

+ Từ khái niệm lớp nẩy sinh hàng loạt khái niệm khác như: Thành phần dữ liệu, phương thức, phạm vi, sự đóng gói, hàm tạo, hàm huỷ, sự thừa kế, lớp cơ sử, lớp dẫn xuất, tương ứng bội, phương thức ảo, ...

+ Ưu điểm của việc thiết kế hướng đối tượng là tập trung xác định các lớp để mô tả các thực thể của bài toán. Mỗi lớp đưa vào các thành phần dữ liệu của thực thể và xây dựng luôn các phương thức để xử lý dữ liệu. Như vậy việc thiết kế chương trình xuất phát từ các nội dụng, các vấn đề của bài toán.

+ Các ngôn ngữ thuần tuý hướng đối tượng (như Smalltalk) chỉ hỗ trợ các khái niệm về lớp, không có các khái niệm hàm.

+ C++ là ngôn ngữ lai , nó cho phép sử dụng cả các công cụ của lớp và hàm.

Để minh hoạ các khái niệm vừa nêu về lập trình hướng đối tượng ta trở lại xét bài toán tìm độ dài lớn nhất đi qua 2 điểm. Trong bài toán này ta gặp một thực thể là dẫy điểm. Các thành phần dữ liệu của lớp dẫy điểm gồm:

- Biến nguyên n là số điểm của dẫy

- Con trỏ x kiểu thực trỏ đến vùng nhớ chứa dẫy hoành độ

- Con trỏ y kiểu thực trỏ đến vùng nhớ chứa dẫy tung độ

Các phương thức cần đưa vào theo yêu cầu bài toán gồm:

- Nhập toạ độ một điểm

- Tính độ dài đoạn thẳng đi qua 2 điểm

Dưới đây là chương trình viết theo thiết kế hướng đối tượng. Để thực hiện chương trình này nhớ đặt tên tệp có đuôi CPP. Xem chương trình ta thấy thêm một điều mới trong C++ là:

Các khai báo biến, mảng có thể viết bất kỳ chỗ nào trong chương trình (tất nhiên phải trước khi sử dụng biến, mảng).

```
#include <stdio.h> #include <conio.h> #include <math.h>
#include <alloc.h> class daydiem { public: int n; float
*x,*y; float do_dai(int i, int j) { return sqrt(pow(x[i]-
x[j],2)+pow(y[i]-y[j],2)); 1213} void nhapsl(void); };
void daydiem::nhapsl(void) { int i; printf("\nSo diem N=
"); scanf("%d",&n); x=(float*)malloc((n+1)*sizeof(float));
```

```
y=(float*)malloc((n+1)*sizeof(float)); for (i=1;i<=n;++i)
{ printf("\nNhap toa do x, y cua diem thu %d : ",i);
scanf("%f%f",&x[i],&y[i]); } } void main() { daydiem p;
p.nhapsl(); int n,i,j,imax,jmax; float d,dmax; n=p.n;
dmax=p.do_dai(1,2); imax=1;jmax=2; for (i=1;i<=n-1;++i)
for (j=i+1;j<=n;++j) { d=p.do_dai(i,j); if (d>dmax) {
dmax=d; imax=i; jmax=j; } } printf("\nDoan thang lon nhat
co do dai bang: %0.2f",dmax); printf("\n Di qua 2 diem co
chi so la %d va %d",imax,jmax); getch(); }
```

Một số mở rộng đơn giản của C++ so với C

Trong mục này trình bầy một số mở rộng của C++ , tuy đơn giản, ngắn gọn nhưng đem lại rất nhiều tiện lợi.

Viết các dòng ghi chú

Trong C++ vẫn có thể viết các dòng ghi chú trong các dấu /* và */ như trong C. Cách này cho phép viết các ghi chú trên nhiều dòng hoặc trên một dòng. Ngoài ra trong C++ còn cho phép viết ghi chú trên một dòng sau 2 dấu gạch chéo, ví dụ:

```
int x,y ; // Khai báo 2 biến thực
```

Khai báo linh hoạt

Trong C tất cả các câu lệnh khai báo biến, mảng cục bộ phải đặt tại đầu khối. Do vậy nhiều khi, vị trí khai báo và vị trí sử dụng của biến khá xa nhau, gây khó khăn trong việc kiểm soát chương trình. C++ đã khắc phục nhược điểm này bằng cách cho phép các lệnh khai báo biến, mảng có thể đặt bất kỳ chỗ nào trong chương trình trước khi các biến, mảng được sử dụng. Ví dụ chương trình nhập một dẫy số thực rồi sắp xếp theo thứ tự tăng dần có thể viết trong C++ như sau:

```
#include <stdio.h> #include <conio.h> #include <alloc.h>
void main() { int n; printf("\n So phan tu cua day N= ");
scanf("%d",&n); float *x=
(float*)malloc((n+1)*sizeof(float)); for (int
i=1;i<=n;++i) { printf("\nX[%d]= ",i); scanf("%f",x+i); }
for (i=1;i<=n-1;++i) for (int j=i+1;j<=n;++j) if
(x[i]>x[j]) { float tg=x[i]; x[i]=x[j]; x[j]=tg; }
printf("\nDay sau khi sap xep\n"); for (i=1;i<=n;++i)
printf("%0.2f ",x[i]); getch(); }
```

Toán tử ép kiểu

Toán tử này được viết trong C như sau:

```
(Kiểu) biểu thức
```

Trong C++ vẫn có thể dùng cách viết này. Ngoài ra C++ cho phép viết một cách khác tiện lợi hơn như sau:

```
Kiểu(biểu thức)
```

Ví dụ chương trình tính công thức

S = 2/1 + 3/2 + ... + (n+1)/n

với n là một số nguyên dương nhập từ bàn phím, có thể viết như sau:

```
#include <stdio.h> #include <conio.h> void main() { int n;
printf("\n So phan tu cua day N= "); scanf("%d",&n); float
s=0.0; for (int i=1;i<=n;++i) s += float(i+1)/float(i) ;
// Ep kieu theo C++ printf("S= %0.2f ",s); getch(); }
```

Hằng có kiểu

Để tạo ra một hằng có kiểu, ta sử dụng từ khoá const đặt trước một khai báo có khởi gán giá trị. Sau đây là một số ví dụ.

+ Hằng nguyên:

```
const int maxsize = 1000; int a[maxsize] ;
```

+ Cấu trúc hằng:

```
typedef struct { int x, y ; // Toạ độ của điểm int mau ;
// Mã mầu của điểm } DIEM ; const DIEM d = {320, 240, 15};
```

Chương trình dưới đây minh hoạ cách dùng hằng có kiểu. Chương trình tạo một cấu trúc hằng (kiểu DIEM) mô tả điểm giữa màn hình đồ hoạ với mầu trắng. Điểm này được hiển thị trên màn hình đồ hoạ.

```
#include <stdio.h> #include <conio.h> #include
<graphics.h> #include <stdlib.h> typedef struct { int x,y;
int mau; } DIEM; void main() { int mh=0,mode=0;
```

```
initgraph(&mh,&mode,""); int loi=graphresult(); if (loi) {
printf("\nLoi do hoa: %s",grapherrormsg(loi)); getch();
exit(0); } const DIEM gmh =
{getmaxx()/2,getmaxy()/2,WHITE};
putpixel(gmh.x,gmh.y,gmh.mau); getch(); closegraph(); }
```

a. Có thể dùng các hàm để gán giá trị cho các hằng có kiểu (trong chương trình trên dùng các hàm getmax và getmaxy).

b. Mọi câu lệnh nhằm thay đổi giá trị hằng có kiểu đều bị báo lỗi khi biên dịch chương trình. Ví dụ nếu trong chương trình đưa vào câu lệnh:

```
gmh.x=200;
```

thì khi dịch chương trình sẽ nhận được thông báo lỗi như sau:

```
Cannot modify a const object
```

Các kiểu char và int

Trong C một hằng ký tự được xem là nguyên do đó nó có kích thước 2 byte, ví dụ trong C:

```
sizeof('A') = sizeof(int) = 2
```

Còn trong C++ một hằng ký tự được xem là giá trị kiểu char và có kích thước một byte. Như vậy trong C++ thì:

```
sizeof('A') = sizeof(char) = 1
```

Lấy địa chỉ các phần tử mảng thực 2 chiều

Trong Turbo C 2.0 không cho phép dùng phép & để lấy địa chỉ các phần tử mảng thực 2 chiều. Vì vậy khi nhập một ma trân thực (dùng scanf) ta phải nhập qua một biến trung gian sau đó mới gán cho các phần tử mảng.

Trong TC ++ 3.0 cho phép lấy địa chỉ các phần tử mảng thực 2 chiều, do đó có thể dùng scanf để nhập trực tiếp vào các phần tử mảng.

Chương trình C++ dưới đây sẽ minh hoạ điều này. Chương trình nhập một ma trận thực cấp mxn và xác định phần tử có giá trị lớn nhất.

```
#include <conio.h> #include <stdio.h> void main() { float
a[20][20], smax; int m,n,i,j, imax, jmax; clrscr(); puts(
"Cho biet so hang va so cot cua ma tran: ") ;
scanf("%d%d",&m,&n) ; for (i=1;i<=m;++i) for
(j=1;j<=n;++j) { printf("\na[%d][%d]= ",i,j);
scanf("%f",&a[i][j]); // Lấy địa chỉ phần tử mảng thực //
2 chiều } smax = a[1][1]; imax=1; jmax=1; for
(i=1;i<=m;++i) for (j=1;j<=n;++j) if (smax<a[i][j]) { smax
= a[i][j]; imax=i ; jmax = j; } puts( "\n\n Ma tran") ;
for (i=1;i<=m;++i) for (j=1;j<=n;++j) { if (j==1)
puts(""); printf("%6.1f", a[i][j]); } puts( "\n\nPhan tu
max:" ); printf("\nco gia tri = %6.1f", smax);
printf("\nTai hang %d cot %d " ,imax, jmax) ; getch(); }
```

Vào ra trong C++

Các toán tử và phương thức xuất nhập

Để in dữ liệu ra màn hình và nhập dữ liệu từ bàn phím , trong C++ vẫn có thể dùng các hàm printf và scanf (như chỉ ra trong các chương trình C++ ở các mục trên).

Ngoài ra trong C++ còn dùng toán tử xuất:

```
cout << biểu thức << ... << biểu thức ;
```

để đưa giá trị các biểu thức ra màn hình, dùng toán tử nhập:

```
cin >> biến >> ... >> biến
```

để nhập các giá trị số (nguyên thực) từ bàn phím và gán cho các biến.

Để nhập một dẫy không quá n ký tự và chứa vào mảng h (kiểu char) có thể dùng phương thức cin.get như sau:

```
cin.get(h,n);
```

Toán tử nhập cin >> sẽ để lại ký tự chuyển dòng ‘\n’ trong bộ đệm, ký tự này có thể làm trôi phương thức cin.get. Để khắc phục tình trạng trên cần dùng phương thức cin.ignore để bỏ qua một ký tự chuyển dòng như sau: `cin.ignore(1);`

Để sử dụng các toán tử và phương thức nói trên cần khai báo tệp tiêu đề:

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

Chương trình sau minh hoạ việc sử dụng các công cụ vào ra mới của C++ để nhập một danh sách n thí sinh. Dữ liệu mỗi thí sinh gồm họ tên, các điểm toán, lý, hoá. Sau đó in danh sách thí sinh theo thứ tự giảm của tổng điểm.

```
#include <iostream.h> #include <conio.h> void main() {
struct { char ht[25]; float t,l,h,td; } ts[50],tg; int
n,i,j; clrscr(); cout << " So thi sinh: " ; cin >> n ; for
(i=1;i<=n;++i) { cout << "\n Thi sinh " << i ; cout << "\n
Ho ten: " ; cin.ignore(1); cin.get(ts[i].ht,25) ; cout <<
"Cac diem toan, ly, hoa: "; cin >> ts[i].t >> ts[i].l >>
ts[i].h ; ts[i].td = ts[i].t + ts[i].l + ts[i].h ; } for
(i=1;i<=n-1;++i) for (j=i+1;j<=n;++j) if (ts[i].td <
ts[j].td ) { tg=ts[i]; ts[i]=ts[j]; ts[j]=tg; } cout <<
"\nDanh sach thi sinh sau khi sap xep " ; for
(i=1;i<=n;++i) { cout << "\n Ho ten: " << ts[i].ht; cout
<< " Tong diem: " << ts[i].td; } getch(); }
```

Định dạng khi in ra màn hình

+ Để quy định số thực (float, double) được in ra có đúng p chữ số sau dấu chấm thập phân, ta sử dụng đồng thời các hàm sau:

```
setiosflags(ios::showpoint); // Bật cờ hiệu showpoint
setprecision(p);
```

Các hàm này cần đặt trong toán tử xuất như sau:

```
cout << setiosflags(ios::showpoint) << setprecision(p) ;
```

Câu lệnh trên sẽ có hiệu lực đối với tất cả các toán tử xuất tiếp theo cho đến khi gặp một câu lệnh định dạng mới.

+ Để quy định độ rộng tối thiểu là w vị trí cho giá trị (nguyên, thực, chuỗi) được in trong các toán tử xuất, ta dùng hàm

```
setw(w)
```

Hàm này cần đặt trong toán tử xuất và nó chỉ có hiệu lực cho một giá trị được in gần nhất. Các giá trị in ra tiếp theo sẽ có độ rộng tối thiểu mặc định là 0. Như vậy câu lệnh:

```
cout << setw(3) << “AB” << “CD”
```

Sẽ in ra 5 ký tự là: một dấu cách và 4 chữ cái A, B, C và D.

Muốn sử dụng các hàm trên cần đưa vào câu lệnh #include sau: `#include <iomanip.h>`

Trở lại chương trình trên ta thấy danh sách thí sinh in ra sẽ không thẳng cột. Để khắc phục điều này cần viết lại đoạn chương trình in như sau:

```
cout << "\nDanh sach thi sinh sau khi sap xep " ; cout <<
setiosflags(ios::showpoint) << setprecision(1) ;
for(i=1;i<=n;++i) { cout << "\n Ho ten: " << setw(25) <<
ts[i].ht; cout << " Tong diem: " << setw(5)<< ts[i].td; }
getch();
```

Chương trình dưới đây là một minh hoạ khác về việc sử dụng các toán tử nhập xuất và cách định dạng trong C++ . Chương trình nhập một ma trận thực cấp mxn. Sau đó in ma trận dưới dạng bảng và tìm một phần tử lớn nhất.

```
#include <iostream.h> #include <iomanip.h> #include
<conio.h> void main() { float a[20][20], smax; int
m,n,i,j, imax, jmax; clrscr(); cout << " Cho biet so hang
va so cot cua ma tran: " ; cin >> m >> n ; for
(i=1;i<=m;++i) for (j=1;j<=n;++j) { cout << "a[" << i <<
"," << j << "]= " ; cin >> a[i][j] ; } smax = a[1][1];
imax=1; jmax=1; for (i=1;i<=m;++i) for (j=1;j<=n;++j) if
(smax<a[i][j]) { smax = a[i][j]; imax=i ; jmax = j; } cout
<< "\n\n Ma tran" ; cout << setiosflags(ios::showpoint) <<
setprecision(1) ; for (i=1;i<=m;++i) for (j=1;j<=n;++j) {
if (j==1) cout << '\n' ; cout << setw(6) << a[i][j]; }
cout << "\n\n" << "Phan tu max:" << '\n' ; cout << "co gia
tri = " << setw(6) << smax; cout << "\nTai hang " << imax
<< " cot " << jmax ; getch(); }
```

Cấu trúc, hợp và kiểu liệt kê

Tên sau từ khoá struct được xem như tên kiểu cấu trúc

Trong C++ một kiểu cấu trúc cũng được định nghĩa như C theo mẫu:

```
struct Tên_kiểu_ct { // Khai báo các thành phần của cấu
trúc } ;
```

Sau đó để khai báo các biến, mảng cấu trúc, trong C dùng mẫu sau:

```
struct Tên_kiểu_ct danh sách biến, mảng cấu trúc ;
```

Như vậy trong C, tên viết sau từ khoá struct chưa phải là tên kiểu và chưa có thể dùng để khai báo.

Trong C++ xem tên viết sau từ khoá struct là tên kiểu cấu trúc và có thể dùng nó để khai báo. Như vậy để khai báo các biến, mảng cấu trúc trong C++ , ta có thể dùng mẫu sau:

```
Tên_kiểu_ct danh sách biến, mảng cấu trúc ;
```

Định nghĩa kiểu cấu trúc TS (thí sinh) gồm các thành phần : ht (họ tên), sobd (số báo danh), dt (điểm toán), dl (điểm lý), dh (điểm hoá) và td (tổng điểm), sau đó khai báo biến cấu trúc h và mảng cấu trúc ts.

```
struct TS { char ht [25]; long sobd; float dt, dl, dh, td;
} ; TS h, ts[1000] ;
```

Tên sau từ khoá union được xem như tên kiểu hợp

Trong C++ một kiểu hợp (union) cũng được định nghĩa như C theo mẫu:

```
union Tên_kiểu_hợp { // Khai báo các thành phần của hợp }
;
```

Sau đó để khai báo các biến, mảng kiểu hợp , trong C dùng mẫu sau:

```
union Tên_kiểu_hợp danh sách biến, mảng kiểu hợp ;
```

Như vậy trong C, tên viết sau từ khoá union chưa phải là tên kiểu và chưa có thể dùng để khai báo.

Trong C++ xem tên viết sau từ khoá union là tên kiểu hợp và có thể dùng nó để khai báo. Như vậy để khai báo các biến, mảng kiểu hợp, trong C++ có thể dùng mẫu sau:

```
Tên_kiểu_hợp danh sách biến, mảng kiểu hợp ;
```

Các union không tên

Trong C++ cho phép dùng các union không tên dạng:

```
union { // Khai báo các thành phần } ;
```

Khi đó các thành phần (khai báo trong union) sẽ dùng chung một vùng nhớ. Điều này cho phép tiết kiệm bộ nhớ và cho phép dễ dàng tách các byte của một vùng nhớ.

Nếu các biến nguyên i , biến ký tự ch và biến thực x không đồng thời sử dụng thì có thể khai báo chúng trong một union không tên như sau:

```
union { int i ; char ch ; float x ; } ;
```

Khi đó các biến i , ch và f sử dụng chung một vùng nhớ 4 byte.

Xét ví dụ khác, để tách các byte của một biến unsigned long ta dùng union không tên sau:

```
union { unsigned long u ; unsigned char b[4] ; };
```

Khí đó nếu gán

```
u = 0xDDCCBBAA; // Số hệ 16 thì : b[0] = 0xAA b[1] = 0xBB
b[2] = 0xCC b[3] = 0xDD
```

Kiểu liệt kê (enum)

+ Cũng giống như cấu trúc và hợp, tên viết sau từ khoá enum được xem là kiểu liệt kê và có thể dùng để khai báo, ví dụ:

```
enum MAU { xanh, do, tim, vang } ; // Định nghĩa kiểu MAU
MAU m, dsm[10] ; // Khai báo các biến, mảng kiểu MAU
```

+ Các giá trị kiểu liệt kê (enum) là các số nguyên. Do đó có thể thực hiện các phép tính trên các giá trị enum, có thể in các giá trị enum, có thể gán giá trị enum cho biến nguyên, ví dụ:

```
MAU m1 , m2 ; int n1, n2 ; m1 = tim ; m2 = vàng ; n1 = m1
; // n1 = 2 n2 = m1 + m2 ; // n2 = 5 printf ("\n %d " , m2
); // in ra số 3
```

+ Không thể gán trực tiếp một giá trị nguyên cho một biến enum mà phải dùng phép ép kiểu, ví dụ:

```
m1 = 2 ; // lỗi m1 = MAU(2) ; // đúng
```

Cấp phát bộ nhớ

Trong C++ có thể sử dụng các hàm cấp phát bộ nhớ động của C như: hàm malloc để cấp phát bộ nhớ, hàm free để giải phóng bộ nhớ được cấp phát.

Ngoài ra trong C++ còn đưa thêm toán tử new để cấp phát bộ nhớ và toán tử delete để giải phóng bộ nhớ được cấp phát bởi new

Cách dùng toán tử new để cấp phát bộ nhớ như sau:

+ Trước hết cần khai báo một con trỏ để chứa địa chỉ vùng nhớ sẽ được cấp phát:

```
Kiểu *p;
```

ở đây Kiểu có thể là:

- các kiểu dữ liệu chuẩn của C++ như int , long, float , double, char , ...

- các kiểu do lập trình viên định nghĩa như: mảng, hợp, cấu trúc, lớp, ...

+ Sau đó dùng toán tử new theo mẫu:

```
p = new Kiểu ; // Cấp phát bộ nhớ cho một biến (một phần tử)

p = new Kiểu[n] ; //Cấp phát bộ nhớ cho n phần tử
```

Để cấp phát bộ nhớ cho một biến thực ta dùng câu lệnh sau:

```
float *px = new float ;
```

Để cấp phát bộ nhớ cho 100 phần tử nguyên ta dùng các câu lệnh:

```
int *pn = new int[100] ; for (int i=0 ; i < 100 ; ++i )
pn[i] = 20*i ; // Gán cho phần tử thứ i
```

Hai cách kiểm tra sự thành công của new

Khi dùng câu lệnh:

```
Kiểu *p = new Kiểu[n] ;
```

hoặc câu lệnh:

```
Kiểu *p = new Kiểu ;
```

để cấp phát bộ nhớ sẽ xuất hiện một trong 2 trường hợp: thành công hoặc không thành công.

Nếu thành công thì p sẽ chứa địa chỉ đầu vùng nhớ được cấp phát.

Nếu không thành công thì p = NULL.

Đoạn chương trình sau minh hoạ cách kiểm tra lỗi cấp phát bộ nhớ:

```
double *pd ; int n ; cout << "\n Số phần tử : " ; cin >> n
; pd = new double[n] ; if (pd==NULL) { cout << " Lỗi cấp
phát bộ nhớ " exit (0) ; }
```

Cách thứ 2 để kiểm tra sự thành công của toán tử new là dùng con trỏ hàm:

```
_new_handler
```

được định nghĩa trong tệp "new.h". Khi gặp lỗi trong toán tử new (cấp phát không thành công) thì chương trình sữ thực hiện một hàm nào đó do con trỏ _new_handler trỏ tới. Cách dùng con trỏ này như sau:

+ Xây dựng một hàm dùng để kiểm tra sự thành công của new

+ Gán tên hàm này cho con trỏ _new_handler

Như vậy hàm kiểm tra sẽ được gọi mỗi khi có lỗi xẩy ra trong toán tử new.

Đoạn chương trình kiểm tra theo cách thứ nhất có thể viết theo cách thứ hai như sau:

```
void kiem_tra_new(void) // Lập hàm kiểm tra { cout << "
Lỗi cấp phát bộ nhớ " exit (0) ; } _new_handler =
kiem_tra_new // Gán tên hàm cho con trỏ double *pd ; int n
; cout << "\n Số phần tử : " ; cin >> n ; pd = new
double[n] ; // Khi xẩy ra lỗi sẽ gọi hàm kiểm_tra_new
```

Có thể dùng lệnh gán để gán tên hàm xử lý lỗi cho con trỏ _new_handler như trong đoạn chương trình trên, hoặc dùng hàm:

```
set_new_handler(Tên hàm) ;
```

(xem các chương trình minh hoạ bên dưới)

Toán tử delete dùng để giải phóng vùng nhớ được cấp phát bởi new

Cách dùng như sau:

```
delete p ; // p là con trỏ dùng trong new

float *px ;

px = new float[2000] ; // Cấp phát bộ nhớ cho 2000 phần tử thực

// Sử dụng bộ nhớ được cấp phát

delete px ; // giải phóng bộ nhớ
```

Hai chương trình minh hoạ

Chương trình thứ nhất minh hoạ cách dùng new để cấp phát bộ nhớ chứa n thí sinh. Mỗi thí sinh là một cấu trúc gồm các trường ht (họ tên), sobd (số báo danh) và td (tổng điểm). Chương trình sẽ nhập n, cấp phát bộ nhớ chứa n thí sinh, kiểm tra lỗi cấp phát bộ nhớ (dùng cách 1), nhập n thí sinh, sắp xếp thí sinh theo thứ tự giảm của tổng điểm, in danh sách thí sinh sau khi sắp xếp, và cuối cùng là giải phóng bộ nhớ đã cấp phát.

```
#include <iomanip.h> #include <iostream.h> #include
<stdlib.h> #include <conio.h> struct TS { char ht[20];
long sobd; float td; } ; void main(void) { TS*ts ; int n;
cout << "\n So thi sinh n = " ; cin >> n; ts = new
TS[n+1]; if(ts==NULL) { cout << "\nLoi cap phat bo nho " ;
getch(); exit(0); } for (int i=1;i<=n;++i) { cout <<"\nThi
sinh thu " << i; cout << "\nHo ten: " ; cin.ignore(1) ;
cin.get(ts[i].ht,20); cout << "So bao danh: " ; cin >>
ts[i].sobd ; cout << "Tong diem: " ; cin >> ts[i].td ; }
for (i=1;i<=n-1;++i) for (int j=i+1;j<=n;++j) if (ts[i].td
< ts[j].td) { TS tg=ts[i]; ts[i]=ts[j]; ts[j]=tg; } cout
<< setiosflags(ios::showpoint) << setprecision(1) ; for
(i=1;i<=n;++i) cout << "\n" << setw(20) << ts[i].ht <<
setw(6)<< ts[i].sobd <<setw(6)<< ts[i].td; delete ts;
getch(); }
```

Chương trình thứ hai minh hoạ cách dùng con trỏ _new_handler để kiểm tra sự thành công của toán tử new. Chương trình sẽ cấp phát bộ nhớ cho một mảng con trỏ và sẽ theo rõi khi nào thì không đủ bộ nhớ để cấp phát.

```
#include <new.h> #include <iostream.h> #include <stdlib.h>
#include <conio.h> int k; void loi_bo_nho(void) { cout <<
"\nLoi bo nho khi cap phat bo nho cho q[" << k << "]";
getch(); exit(0); } void main() { double *q[100] ; long n;
clrscr(); set_new_handler(loi_bo_nho) ; //
_new_handler=loi_bo_nho; n=10000; for ( k=0;k<100;++k)
q[k] = new double[n]; cout << "Khong loi"; getch(); }
```

Các hàm trong C++

Trong C++ có rất nhiều mở rộng, cải tiến về hàm làm cho việc xây dựng và sử dụng hàm rất tiện lợi. Điều này sẽ trình bầy kỹ trong chương sau. Trong mục này chỉ thống kê một số điểm mới về hàm mà C++ đưa vào.

Đối kiểu tham chiếu

Trong C, để nhận kết quả của hàm cần dùng đối con trỏ, làm cho việc xây dựng cũng như sử dụng hàm khá phiền phức. Trong C++ đưa vào đối kiểu tham chiếu (giống như PASCAL) dùng để chứa kết quả của hàm, khiến cho việc tạo lập cũng như sử dụng hàm đơn giản hơn.

Đối tham chiếu const

Đối tham chiếu có đặc điểm là các câu lệnh trong thân hàm có thể truy nhập tới và dễ dàng làm cho giá trị của nó thay đổi. Nhiều khi ta muốn dùng đối kiểu tham chiếu chỉ để tăng tốc độ trao đổi dữ liệu giữa các hàm , không muốn dùng nó để chứa kết quả của hàm. Khi đó có thể dùng đối tham chiếu const để bảo toàn giá trị của đối trong thân hàm.

Đối có giá trị mặc định

Trong nhiều trương hợp người dùng viết một lời gọi hàm nhưng còn chưa biết nên chọn giá trị nào cho các đối . Để khắc phục khó khăn này, C++ đưa ra giải pháp đối có giá trị mặc định. Khi xây dựng hàm, ta gán giá trị mặc định cho một số đối. Người dùng nếu không cung cấp giá trị cho các đối này, thì hàm sẽ dùng giá trị mặc định.

Hàm on line

Đối với một đoạn chương trình nhỏ (số lệnh không lớn) thì việc thay các đoạn chương trình này bằng các lời gọi hàm sẽ làm cho chương trình gọn nhẹ đôi chút nhưng làm tăng thời gian máy. Trong các trường hợp này có thể dùng hàm trực tuyến (on line) vừa giảm kích thước chương trình nguồn, vừa không làm tăng thời gian chạy máy.

Các hàm trùng tên (định nghĩa chồng các hàm)

Để lấy giá trị tuyệt đối của một số, trong C cần lập ra nhiều hàm với tên khác nhau, ví dụ abs cho số nguyên, fabs cho số thực, labs cho số nguyên dài, cabs cho số phức. Điều này rõ ràng gây phiền toái cho người sử dụng. Trong C++ cho phép xây dựng các hàm trùng tên nhưng khác nhau về kiểu đối. Như vậy chỉ cần lập một hàm để lấy giá trị tuyệt đối cho nhiều kiểu dữ liệu khác nhau.

Định nghĩa chồng toán tử

Việc dùng các phép toán thay cho một lời gọi hàm rõ ràng làm cho chương trình ngắn gọn, sáng sủa hơn nhiều. Ví dụ để thực hiện phép cộng 2 ma trận nếu dùng phép cộng và viết:

C = A + B ;

thì rất gần với toán học. Trong C++ cho phép dùng các phép toán chuẩn để đặt tên cho các hàm (gọi là định nghĩa chồng toán tử). Sau đó có thể thay lời gọi hàm bằng các phép toán như nói ở trên. Như vậy một phép toán mang nhiều ý nghĩa, ví dụ phép + có thể hiểu là cộng 2 số nguyên, 2 số thực hoặc 2 ma trận. C++ sẽ căn cứ vào kiểu của các số hạng mà quyết định chọn phép cộng cụ thể.