CHƯƠNG 5: CÂU LỆNH LẶP

Mục tiêu của chương

Nắm vững:

- Cú pháp và ý nghĩa các câu lệnh lặp for.
- Cú pháp và ý nghĩa các câu lệnh lặp while và do while .
- Các quy trình thực hiện các lệnh lặp.
- Vận dụng các câu lệnh đã học để giải các bài tập trong chương 5.

Nội dung của chương

Nghiên cứu cơ chế hoạt động của các lệnh lặp, cách sử dụng các lệnh.

5.1. Câu lệnh lặp for

Câu lệnh lặp

Một trong những cấu trúc quan trọng của lập trình cấu trúc là các câu lệnh cho phép lặp nhiều lần một đoạn lệnh nào đó của chương trình.

Chẳng hạn trong ví dụ về bài toán nhân theo phương pháp Ấn độ, để lặp lại một đoạn lệnh chúng ta đã sử dụng câu lệnh goto. Tuy nhiên như đã lưu ý việc dùng nhiều câu lệnh này làm chương trình rất khó đọc.

Do vậy cần có những câu lệnh khác trực quan hơn và thực hiện các phép lặp một cách trực tiếp. C++ cung cấp cho chúng ta 3 lệnh lặp như vậy. Về thực chất 3 lệnh này là tương đương (cũng như có thể dùng goto thay cho cả 3 lệnh lặp này), tuy nhiên để chương trình viết được sáng sủa, rõ ràng, C++ đã cung cấp nhiều phương án cho người sử dụng lựa chọn câu lệnh khi viết chương trình phù hợp với tính chất lặp.

Mỗi bài toán lặp có một đặc trưng riêng, ví dụ lặp cho đến khi đã đủ số lần định trước thì dừng hoặc lặp cho đến khi một điều kiện nào đó không còn thoả mãn nữa thì dừng … việc sử dụng câu lệnh lặp phù hợp sẽ làm cho chương trình dễ đọc và dễ bảo trì hơn.

a. Cú pháp

for ([<phần khởi tạo>] ; [<điều kiện>] ; [<dãy biểu thức >])

<lệnh>;

Vòng lặp for được định nghĩa bởi từ khóa for và được chia làm 3 phần chính, mỗi phần được ngăn cách bởi dấu chấm phẩy.

Trong đó:

- <phần khởi tạo> là một hay nhiều biểu thức gán (được phân cách bởi dấu ‘,’) có nhiệm vụ khởi tạo giá trị ban đầu cho các biến đếm.
- < điều kiện > thường là biểu thức logic.
- < dãy biểu thức> là một hay nhiều biểu thức gán (được phân cách bởi dấu ‘,’) có nhiệm vụ thay đổi trị của các biến ở <phần khởi tạo>.
- <lệnh> có thể là câu lệnh đơn, khối lệnh, hoặc câu lệnh điều khiển.

b. Ý nghĩa

Vòng lặp for được sử dụng phần lớn trong các cấu trúc lặp. Vòng lặp for phù hợp cho cả trường hợp biết trước số lần lặp lẫn không biết trước số lần lặp.

Lệnh for thực hiện như sau:

B1: Thực hiện <phần khởi tạo> (nếu có).

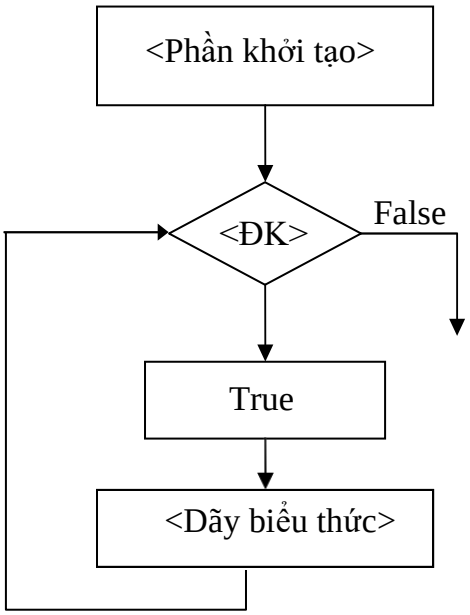
B2: Kiểm tra <điều kiện>.

B3: Nếu <điều kiện> đúng thì thực hiện <lệnh>, sau đó thực hiện <phần khởi tạo> (nếu có) và quay về B2.

Còn ngược lại nếu <điều kiện > sai thì chuyển sang B4.

B4: Thoát khỏi vòng lặp, và chuyển quyền điều khiển sang câu lệnh kế tiếp sau lệnh for.

c. Lưu đồ



Hình 13: Sơ đồ hoạt động của lệnh for.

Ví dụ 5.1: In các số từ 1 đến 10 ra màn hình.

```
for (int x = 1; x <= 10; x = x+ 1)
{
   cout << x<< endl;
}
```

Quá trình thực hiện:

1. Khai báo biến x và gán giá trị x = 1.
2. Kiểm tra x <= 10 không? Nếu không thì dừng.
3. In giá trị x ra màn hình.
4. Tăng x lên 1.
5. Chuyển về bước 2.

Ví dụ 5.2: biểu diễn các thành phần của vòng lặp for

```
for (int count = 1; count <= 10; count++)
{
   cout << "count = " << count << endl;
}
```

Ví dụ 5.3: sử dụng vòng lặp for để in ra tất cả các số chẵn từ 0 đến 10.

```
for (int i = 0; i <= 10; i++)
{
   if (i % 2 == 0)
         cout << i << " ";
}
```

Vòng lặp trên có thể được rút gọn lại như sau:

```
for (int i = 0; i <= 10; i += 2)
{
   cout << i << " ";
}
```

Ví dụ 5.4: sử dụng vòng lặp for để in ra tất cả các số chẵn từ 10 đến 0.

```
for (int i = 10; i >= 0; i -= 2)
{
   cout << i << " ";
}
```

Ví dụ 5.5: Tính tổng của dãy các số từ 1 đến 500.

```
int main ()
{
   int i, kq = 0;
   for (i = 1 ; i <= 500 ; i ++) kq += i ;
   cout << "Tổng = " << kq;
}
```

Ví dụ 5.6: Tính giai thừa của một số nguyên dương

```
#include <iostream>
#include <math.h>
using namespace std;
int main ()
{
   // Khai bao bien
   int x, n, fact = 1;
   // Nhap gia tri dau vao
   cout<<"Nhap mot so :";
   cin>>n;
   //Vong lap for
   for (int x = 1; x <= n; x++)
   {
      fact = fact * x;
   }
   cout<<n<<" Gia tri cua giai thua la "<<fact;
   return 0;
 }
```

5.2. Câu lệnh lặp while

a. Cú pháp

while (<điều kiện>)

<lệnh>;

Trong đó:

- <điều kiện> thường là biểu thức logic.

- <lệnh> có thể là câu lệnh đơn, khối lệnh, hoặc câu lệnh điều khiển.

b. Ý nghĩa

Chừng nào mà điều kiện còn thỏa mãn thì thực hiện <lệnh>.

Lệnh while thực hiện như sau:

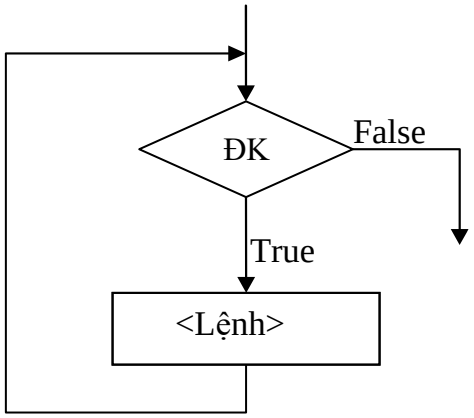
B1: Kiểm tra <điều kiện>.

B2: Nếu <điều kiện> đúng thì thực hiện <lệnh>, sau đó quay về B1.

Còn ngược lại <điều kiện > sai thì chuyển sang B3.

B3: Thoát khỏi vòng lặp, và chuyển quyền điều khiển sang câu lệnh kế tiếp sau lệnh while.

c. Lưu đồ



Hình 14: Sơ đồ hoạt động của lệnh while

Chú ý:

- Do <điều kiện> được kiểm tra trước, nên phần <lệnh> của vòng lặp while có thể không được thực hiện lần nào.

- Để bảo đảm cho vòng lặp while (…) không xác định, thì phải có ít nhất một câu lệnh trong phần thân vòng lặp có tác dụng làm thay đổi việc đánh giá lại <điều kiện > thoát sau mỗi lần lặp.

Ví dụ 5.7: Sử dụng vòng lặp while để in các số từ 1 đến 10.

```
#include <iostream>
#include <math.h>
using namespace std;
int main ()
{
int i=1;
        while (i<=10)
```

```
        {
        cout << i << endl;
        i++;
        }
       return 0;
}
```

Ví dụ 5.8: Chương trình nhập một câu, đếm số từ và ký tự trong câu đó, và in kết quả ra màn hình.

```
#include <iostream>
#include <math.h>
using namespace std;
int main ()
{
    int demkytu=0;
    int demtu=1;
    cout << "Ban nhap mot cau gom cac chu thuong: " << endl;
    char ch='a';
    while(ch!='\r')
    {
        ch=getch();
        if(ch==' ')
        demtu++;
        else
        demkytu++;
    }
    cout << "\n So tu trong cau la = " << demtu << endl;
    cout << "\nSo ky tu trong cau la = " << demkytu-1 << endl;
    return 0;
}
```

Ví dụ 5.9: Chương trình tìm ước chung lớn nhất (UCLN) của 2 số nguyên m và n.

Áp dụng thuật toán Euclide bằng cách liên tiếp lấy số lớn trừ đi số nhỏ khi nào 2 số bằng nhau thì đó là UCLN. Trong chương trình ta qui ước m là số lớn và n là số nhỏ.

Thêm biến phụ r để tính hiệu của 2 số. Sau đó đặt lại m hoặc n bằng r sao cho m > n và lặp lại. Vòng lặp dừng khi m = n.

```
#include <iostream>
#include <math.h>
using namespace std;
int main ()
{
    int m, n, r;
    cout << "Nhập m, n: " ; cin >> m >> n ;
    if (m < n) { int t = m; m = n; n = t; } // nếu m < n thì đổi vai trò hai số
    while (m != n)
    {
        r = m - n ;
        if (r > n) m = r; else { m = n ; n = r ; }
    }
    cout << "UCLN = " << m ;
}
```

Ví dụ 5.10: Chương trình nhân 2 số nguyên theo phương pháp Ấn độ

```
int main ()
{
long m, n, kq; // Các số cần nhân và kết quả kq
cout << “Nhập m và n: “ ; cin >> m >> n ;
kq = 0 ;
while (m)
        {
        if (m%2) kq += n ;
        m >>= 1;
        n <<= 1;
        }
cout << “m nhân n =” << kq ;
}
```

5.3. Câu lệnh lặp do while

a. Cú pháp

do

{

<lệnh>;

} while (<điều kiện>);

Trong đó:

- <điều kiện> thường là biểu thức logic.
- <lệnh> có thể là câu lệnh đơn, khối lệnh, hoặc câu lệnh điều khiển.

b. Ý nghĩa

Thực hiện <lệnh> cho đến khi <điều kiện> không còn được thỏa mãn.

Lệnh do while thực hiện như sau:

B1: Thực hiện <lệnh>.

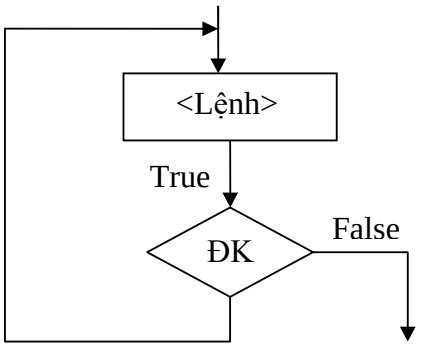
B2: Kiểm tra <điều kiện>.

B3: Nếu <điều kiện> đúng thì quay về B1.

Còn ngược lại nếu <điều kiện> sai thì chuyển sang B4.

B4: Thoát khỏi vòng lặp, và chuyển quyền điều khiển sang câu lệnh kế tiếp sau lệnh do while.

c. Lưu đồ



Hình 15: Sơ đồ hoạt động của lệnh do while.

Chú ý: Khác với vòng lặp while(…), phần <lệnh> trong do.. while (…) luôn được thực hiện ít nhất là 1 lần, do <điều kiện> được kiểm tra sau.

Trong trường hợp vòng lặp do .. while (…) không xác định, ta nên xem xét và hiệu chỉnh lại các câu lệnh trong phần thân vòng lặp do … while (…) có liên quan đến <điều kiện > thoát của vòng lặp.

Ví dụ 5.11: Chương trình nhập một số nguyên và in kết quả ra màn hình dưới dạng số đảo ngược về thứ tự của số nguyên đó.

```
#include <iostream>
#include <math.h>
using namespace std;
int main ()
{
    long int so1, so2, sodaonguoc = 0;
    cout << "Nhap mot so nguyen : " << endl;
    cin>>so1;
    so2=so1;
    do
    {
       sodaonguoc=sodaonguoc*10;
       int digit=so1%10;
       sodaonguoc+=digit;
       so1/=10;
    }
    while(so1);
    cout << "So nguyen da nhap la " << so2 << "." << endl;
    cout << "So nguyen dao nguoc la " << sodaonguoc << "." << endl;
    return 0;
}
```

Ví dụ 5.12: Kiểm tra một số n có là số nguyên tố.

Để kiểm tra một số n > 3 có phải là số nguyên tố ta lần lượt chia n cho các số i đi từ 2 đến một nửa của n. Nếu có i sao cho n chia hết cho i thì n là hợp số ngược lại n là số nguyên tố.

```
#include <iostream>
#include <math.h>
using namespace std;
int main ()
{
```

```
        int i, n ; // n: số cần kiểm tra
        cout << "Cho biết số cần kiểm tra: " ; cin >> n ;
        i = 2 ;
        do
        {
        if (n%i == 0)
                {
                   cout << n << "là hợp số" ;
                   return ; // dừng chương trình
                }
                i++;
        } while (i <= n/2);
        cout << n << "là số nguyên tố" ;
}
```

Ví dụ 5.13: Chương trình kiểm tra dữ liệu nhập vào có thể là một tháng trong năm hay không:

```
#include <iostream>
#include <math.h>
using namespace std;
int main ()
{
        int month;
        do
        {
           cin >> month;
        }  while (month < 1 || month > 12);
        return 0;
}
```

5.4. Câu lệnh nhảy goto

Một dạng khác của rẽ nhánh là câu lệnh nhảy goto cho phép chương trình chuyển đến thực hiện một đoạn lệnh khác bắt đầu từ một điểm được đánh dấu bởi một nhãn trong chương trình.

Lệnh goto thường được sử dụng để tạo vòng lặp. Tuy nhiên việc xuất hiện nhiều lệnh goto dẫn đến việc khó theo dõi trình tự thực hiện chương trình, vì vậy lệnh này thường được sử dụng rất hạn chế.

a. Cú pháp

goto <nhãn> ;

Trong đó, "nhãn" là một tên hợp lệ được sử dụng để đánh dấu vị trí dòng lệnh mà người sử dụng muốn "nhảy" đến.

Vị trí chương trình chuyển đến thực hiện là đoạn lệnh đứng sau nhãn và dấu hai chấm (:).

b. Một số ví dụ

Ví dụ 5.14a: Chương trình nhập vào số nguyên dương. Nếu nhập một số âm, chương trình sẽ sử dụng lệnh goto để nhảy đến nhãn "nhaplai". Chương trình sẽ lặp lại thao tác nhập và chỉ kết thúc khi người dùng nhập vào một số nguyên dương.

```
#include <iostream>
#include <math.h>
using namespace std;
int main ()
{
	int n;
	nhaplai: // nhãn
	cout << "Nhap so nguyen duong:";
	cin >> n;
	if (n < 0)
	goto nhaplai; // nhảy đến nhãn nhaplai
	cout << n << " la so nguyen duong" << endl;
	return 0;
}
```

Ví dụ 5.14b: Minh họa lệnh goto

```
#include <iostream>
#include <math.h>
using namespace std;
int main ()
{
        int n=10;
        loop: ;
        cout << n << ", ";
        n--;
        if (n>0) goto loop;
       cout << "Kết thúc!";
       return 0;
}
```

5.5. Câu lệnh break và continue

a. Lệnh break

Công dụng của lệnh dùng để thoát ra khỏi các câu lệnh cấu trúc, chương trình sẽ tiếp tục thực hiện các câu lệnh tiếp sau câu lệnh vừa thoát.

Từ khóa break được dùng để kết thúc vòng lặp, hoặc cấu trúc switch.

Khi sử dụng trong lệnh switch, từ khóa break thường được đặt tại cuối mỗi khối lệnh mỗi nhãn case.

Ví dụ 5.15:

```
for (int i = 0; i < 100; i++)
{
<lệnh A>;          // thực hiện <lệnh A>
 break;            // dừng vòng lặp for ngay lập tức
<lệnh B>;          // <lệnh B> sẽ không được thực hiện
}
```

Ví dụ 5.16: Chương trình cho phép người dùng nhập liên tục giá trị n cho đến khi nhập giá trị âm thì dừng.

```
int main ()
{
```

```
        int n;
        while (1)
        {
          cout<<“\nNhap n: ”; cin>>n;
          if(n<0)
          break;
        }
}
```

b. Lệnh continue

Khi gặp câu lệnh continue trong 1 vòng lặp, chương trình dịch bỏ qua các lệnh còn lại trong thân vòng lặp này để bắt đầu một lần lặp mới.

Sử dụng continue khi cần dừng bước lặp hiện tại, tiếp tục luôn bước lặp mới

Ví dụ 5.17:

```
for (int i = 0; i < 100; i++)
{
    <lệnh A>;          // thực hiện <lệnh A>
    continue; // trở về đầu vòng lặp, chạy bước mới.
    <lệnh B>;          // <lệnh B> sẽ không được thực hiện.
}
```

Ví dụ 5.18: In ra màn hình giá trị từ 10 đến 20 trừ đi số 14 và số 18.

```
#include <iostream>
#include <math.h>
using namespace std;
int main ()
{
        for (int i=10 ; i<=20; i++)
        {
        if (i==14||i==18)
        continue;
        cout<<i<<",";
        }
```

```
cout<<"Ket thuc";
}
```

Ví dụ 5.19: Tính tổng các số lẻ từ 1 đến n.

```
#include <iostream>
#include <math.h>
using namespace std;
int main ()
{
        int n, sum = 0;
        cout << "Nhập n = ";
        cin >> n;
        for (int i = 0; i <= n; i++)
        {
          if (i % 2 == 0)
          {
            continue;
          }
          sum += i;
        }
        cout << "Sum = " << sum;
        return 0;
 }
```

Nhận xét: Khi i là số chắn lúc đó "i % 2 == 0;" trả về true, lệnh continue được thực hiện, dòng lệnh "sum += i;" sẽ được bỏ qua, để nhảy tới cuối thân vòng lặp, thực hiện vòng lặp tiếp theo.

Khi i là số lẻ dòng lệnh "sum += i;" sẽ được thực hiện, để tính tổng các số lẻ.

Câu hỏi thảo luận

1. Trình bày cú pháp, ý nghĩa và cách thực hiện các câu lệnh if, while và do while.
2. So sánh sự giống nhau và khác nhau giữa lệnh while và lệnh do while.
3. Trình bày lưu đồ của các lệnh if, while và do while.
4. Trình bày cách dùng các lệnh goto, break và continue. Nêu ưu nhược điểm của các lệnh đó.

5. Cho ví dụ minh họa hoạt động của lệnh if, while và do while.

Bài tập vận dụng

1. Viết chương trình tính giai thừa của một số nguyên n nhập từ bàn phím.

2. Viết chương trình tính dân số của một thành phố sau 10 năm nữa, biết rằng dân số hiện tại là 1.700.000 người, và tỉ lệ tăng dân số hằng năm của thành phố này là 1.7%.

3. Viết chương trình in ra bảng cửu chương.

4. Viết chương trình tìm ước chung lớn nhất, bội chung nhỏ nhất của 2 số nguyên M, N nhập từ bàn phím.

5. Nhập vào 1 số bất kỳ (0->9), cho biết cách đọc số vừa nhập.

6. Viết chương trình in trên màn hình các số từ 1->10, các số ngăn cách nhau bởi 1 đoạn khoảng trắng.

7. Viết chương trình tính tổng: 1 + 2 + 3 + 4 + 5 +….+ 50.

8. Viết chương trình tính tổng: 1*2 + 2*3+ 3*4 + 4*5 +…..+ n(n+1).

9. Viết chương trình tính tích: 1*2*3*4*5*….*n, trong đó n nhập từ phím.

10. Viết chương trình in bảng cửu chương từ 1 đến 5 theo hàng ngang

11. Viết chương trình hiển thị tất cả các số lẻ nhỏ hơn n, trong đó n nhập từ phím.

12. Viết chương trình tính tổng các số chẵn nhỏ hơn n, trong đó n nhập từ bàn phím.

13. Viết chương trình in ra các số là bội số của 5 nhỏ hơn n, trong đó n nhập từ phím.

14. Viết chương trình đếm số lượng số chẵn trong [n,m], trong đó n,m nhập từ phím.

15. Viết chương trình tính tổng các số tự nhiên nhỏ hơn n (sử dụng vòng lặp while)

16. Viết chương trình tìm tổng các số tự nhiên lớn nhất nhỏ hơn 100.

17. Viết chương trình tính tính tiền điện sử dụng trong tháng:

Từ 1 - 100KW: 700đồng

Từ 101 - 200KW: 1000đồng

Từ 201 - 300KW: 1500 đồng

Từ 300KW trở lên: 2000 đồng.

18. Hãy chuyển đổi câu lệnh for thành câu lệnh while:

for (int i = 0; i < 100; i++)

```
{
    for ( int j = 0; j < 200; j++ )
    cout << setw( 5 ) << i * j;
    cout << endl ;
}
```

19. Hãy chuyển đổi câu lệnh while thành câu lệnh for

```
int count = 0;
while ( count < 100 )
{
    cout << count;
    count++;
}
```

20. Số "hoàn thiện" (perfect number) là số tự nhiên có tổng các ước số (kể cả 1) bằng chính nó. VD: số tự nhiên 28 là số hoàn thiện. Viết chương trình hiển thị ra màn hình tất cả các số hoàn thiện < 100.

21. Viết chương trình nhập vào số n > 0 và hiển thị ra màn hình theo dạng sau:

Ví dụ với n = 5.

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

22. Nhập số n nguyên dương, tính và in các số chính phương từ 1^2 đến n^2 ra màn hình, mỗi số một dòng.

23. Viết chương trình tính giá trị số X sau:

$X = 1 \times 2 + 2 \times 3 + \cdots + 99 \times 100.$

24. Nhập số n nguyên dương, tính giá trị số Y dưới đây

$Y = 1 + 1/2 + 1/3 + \cdots + 1/n.$

CHƯƠNG 6: CON TRỎ VÀ CẤP PHÁT BỘ NHỚ ĐỘNG

Mục tiêu của chương

Nắm vững:

- Khái niệm con trỏ, cách khai báo và sử dụng con trỏ.
- Cấu trúc các toán tử cấp phát và giải phóng bộ nhớ cho con trỏ.
- Vận dụng lý thuyết để giải các bài tập cụ thể.

Nội dung của chương

Nghiên cứu các vấn đề liên quan đến việc sử dụng con trỏ và cách cấp phát bộ nhớ động trong ngôn ngữ C++.

6.1. Địa chỉ, phép toán &, toán tử tham chiếu *

6.1.1. Địa chỉ, phép toán &

Chúng ta đã biết các biến chính là các ô nhớ mà chúng ta có thể truy xuất dưới các tên. Các biến này được lưu trữ tại những chỗ cụ thể trong bộ nhớ.

Để tạo điều kiện truy nhập dễ dàng trở lại các biến này, bộ nhớ được đánh số, mỗi byte sẽ được ứng với một số nguyên, được gọi là địa chỉ của byte đó từ 0 đến hết bộ nhớ.

Bộ nhớ máy tính chỉ là một dãy gồm các ô nhớ 1 byte, mỗi ô có một địa chỉ xác định.

Từ đó ngoài việc thông qua tên biến người sử dụng còn có thể thông qua địa chỉ của chúng để truy nhập vào nội dung. Như vậy biến, ô nhớ và địa chỉ có quan hệ khăng khít với nhau. C++ cung cấp một toán tử một ngôi & để lấy địa chỉ của các biến (ngoại trừ biến mảng và xâu kí tự).

Nếu x là một biến thì &x là địa chỉ của x.

Kết quả của phép lấy địa chỉ (&) là một con trỏ, do đó có thể dùng để gán cho một biến pointer.

Ví dụ 6.1a:

```
int *px, num;
// px là một pointer chỉ đến biến kiểu int là num.
px = &num;
```

Chú ý: int *px, num;

```
px = &(num +1); // sai vì ( num+1) không phải là một biến cụ thể
```

Đối với biến kiểu mảng, thì tên mảng chính là địa chỉ của mảng, do đó không cần dùng đến toán tử &.

Ví dụ địa chỉ của mảng a chính là a (không phải &a).

Mặt khác địa chỉ của mảng a cũng chính là địa chỉ của byte đầu tiên mà mảng a chiếm và nó cũng chính là địa chỉ của phần tử đầu tiên của mảng a. Do vậy địa chỉ của mảng a là địa chỉ của phần tử a[0] tức &a[0].

Ví dụ 6.1b:

```
int x;   // khai báo biến nguyên x
long y; // khai báo biến nguyên dài y
cout << &x << &y; // in địa chỉ các biến x, y
char s[9]; // khai báo mảng kí tự s
cout << a; // in địa chỉ mảng s
cout << &a[0]; // in địa chỉ mảng s (tức địa chỉ s[0])
```

Các phép toán liên quan đến địa chỉ được gọi là số học địa chỉ. Các thao tác được phép trên địa chỉ vẫn phải thông qua các biến trung gian chứa địa chỉ, được gọi là biến con trỏ.

6.1.2. Toán tử tham chiếu *

Ta có toán tử tham chiếu được kí hiệu bởi dấu * cho phép lấy giá trị của vùng nhớ có địa chỉ cụ thể.

Xét lại ví dụ 6.1a, ta có:

px là một pointer chỉ đến biến num như ví dụ 6.1a, thì * px là giá trị của biến num.

Ví dụ 6.2:

a) //num là biến được khai báo và gán giá trị là 5.

```
int num = 5 ;
int *px;    // px là một con trỏ chỉ đến kiểu int
px= &num ; //px là địa chỉ của biến num.
/*giá trị của *px (tức là num) cộng thêm 3, gán cho k.
  Sau đó *px thực hiện lệnh tăng 1 đơn vị (++)*/
int k = (* px)++ + 3 ;
// Sau câu lệnh trên num = 6, k = 8.
```

b)
```
int num1 = 2, num2, *pnt;
pnt = &num1
num2 = *pnt;
```

Trong ví dụ 6.2, biến num1 được gán bằng 2.

Dòng pnt = &num1 nghĩa là biến con trỏ pnt chứa địa chỉ của biến num1.

Phép gán num2 = *pnt, dấu '*' được đặt ở phía trước biến con trỏ, thì giá trị trả về của biến này l giá trị của biến được trỏ tới bởi con trỏ pnt. Do đó, num2 có giá trị là 2.

Ví dụ 6.3: ta biết địa chỉ của biến x là 0x7ffeecb835c8, giờ thay vì gọi định danh để lấy giá trị, ta gọi địa chỉ để lấy giá trị.

```
int main ()
{
  int x = 5;
  // Bình thường lấy giá trị qua định danh
  cout << x << endl;
  // Lấy giá trị qua địa chỉ &x
  cout << *(&x) << endl;
  return 0;
}
```

Ngoài việc dùng để truy xuất giá trị trong vùng nhớ có địa chỉ cụ thể, toán tử tham chiếu còn dùng để thay đổi giá trị của vùng nhớ như cách ta dùng định danh.

Ví dụ 6.4: minh họa dùng toán tử *

```
// Cách dùng thông thường
int x = 5;
cout << x << endl;
x = 10;
cout << x << endl;
// Cách dùng toán tử tham chiếu
int y = 5;
cout << y << endl;
*(&y) = 10;
cout << y << endl;
// hoặc
cout << *(&y) << endl;
```

6.2. Con trỏ

6.2.1. Khái niệm con trỏ

Con trỏ (Pointer) là một kiểu dữ liệu đặc biệt dùng để quản lý địa chỉ của các nhớ. Một con trỏ quản lý các địa chỉ mà dữ liệu tại các địa chỉ này có kiểu T thì con trỏ đó được gọi là con trỏ kiểu T. Con trỏ kiểu T chỉ được dùng để chứa địa chỉ của biến kiểu T.

Nghĩa là con trỏ kiểu int chỉ được dùng để chứa biến kiểu int, con trỏ kiểu char chỉ được dùng chứa biến kiểu char.

Biến con trỏ là một đặc trưng mạnh của C++, nó cho phép chúng ta thâm nhập trực tiếp vào bộ nhớ để xử lý các bài toán khó.

6.2.2. 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: 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++:
  - 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 6.5a: để khai báo một biến con trỏ có kiểu là int và tên là p, ta viết như sau:

```
int *p;
```

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

```
int *p;
int* p;
```

Ví dụ 6.5b:

```
int i, j ; // khai báo 2 biến nguyên i, j
int *p, *q ; // khai báo 2 con trỏ nguyên p, q
```

```
p = &i; // cho p trỏ tới i
q = &j; // cho q trỏ tới j
cout << &i ; // hỏi địa chỉ biến i
cout << q ; // hỏi địa chỉ biến j (thông qua q)
i = 2; // gán i bằng 2
*q = 5; // gán j bằng 5 (thông qua q)
i++ ; cout << i ; // tăng i và hỏi i, i = 3
```

6.2.3. Tham chiếu và con trỏ trong C++

Điểm khác nhau giữa tham chiếu và con trỏ trong C++:

Khi một tham chiếu được khởi tạo cho một đối tượng, nó không thể bị thay đổi để tham chiếu tới đối tượng khác. Các con trỏ có thể được trỏ tới đối tượng khác tại bất kỳ thời điểm nào.

Một tham chiếu phải được khởi tạo khi nó được tạo. Các con trỏ có thể được tạo tại bất kỳ thời điểm nào.

■ Tạo tham chiếu trong C++

Ta coi một tên biến như là một label (một nhãn) được đính kèm với vị trí biến trong bộ nhớ..

Ví dụ 6.6: `int i = 10;`

Có thể khai báo các biến tham chiếu cho i như sau:

```
int&  r = i;
```

Đọc & trong các khai báo này là ”tham chiếu”.

Ví dụ 6.7: Sử dụng các tham chiếu trong C++:

```
int main ()
{
  // khai bao cac bien
  int    i;
  double d;
  // khai bao cac bien tham chieu
  int&  r = i;
  double&  s = d;
  i = 15;
  cout << "Gia tri cua i la: " << i << endl;
```

```
  cout << "Gia tri cua tham chieu toi i la: " << r  << endl;
  d = 21.5;
  cout << "Gia tri cua d : " << d << endl;
  cout << "Gia tri cua tham chieu toi d la: " << s  << endl;
  return 0;
}
```

6.2.4. 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>;

Chú ý: trong phép toán này, tên con trỏ không có dấu "*".

Ví dụ 6.8:

```
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

***<Tên con trỏ>**;

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 6.9:

```
int x = 15, y, *px;
px = &y;
*px = x;
```

Con trỏ px vẫn trỏ tới địa chỉ biến y và giá trị của biến y sẽ là 15.

6.3. Các phép toán với con trỏ

6.3.1. Phép toán gán

- Gán con trỏ với địa chỉ một biến: p = &x ;

- Gán con trỏ với con trỏ khác: p = q ; (sau phép toán gán này p, q chứa cùng một địa chỉ, cùng trỏ đến một nơi).

Ví dụ 6.10:

```
int i = 5 ; // khai báo và khởi tạo biến i = 5
int *p, *q, *r ; // khai báo 3 con trỏ nguyên p, q, r
p = q = r = &i ; // cùng trỏ tới i
```

6.3.2. Phép toán tăng giảm địa chỉ p ± n

Con trỏ trỏ đến thành phần thứ n sau (trước) p.

Một đơn vị tăng giảm của con trỏ bằng kích thước của biến được trỏ.

Như vậy, phép toán tăng, giảm con trỏ cho phép làm việc thuận lợi trên mảng.

Nếu con trỏ đang trỏ đến mảng (tức đang chứa địa chỉ đầu tiên của mảng), việc tăng con trỏ lên 1 đơn vị sẽ dịch chuyển con trỏ trỏ đến phần tử thứ hai, … Từ đó ta có thể cho con trỏ chạy từ đầu đến cuối mảng bằng cách tăng con trỏ lên từng đơn vị như trong câu lệnh for dưới đây.

Ví dụ 6.11:

```
int c[10] = { 1, 2, 3, 4, 5 }, *p, *q;
p = c; cout << *p ;    // cho p trỏ đến mảng c, *p = c[0] = 1
p += 2; cout << *p ; // *p = c[2] = 3 ;
q = p - 1 ; cout << *q ;
for (int i=0; i<10; i++) cout << *(p+i) ; // in toàn bộ mảng c.
```

6.3.3. Phép toán tự tăng giảm

p++, p--, ++p, --p: tương tự p+1 và p-1, có chú ý đến tăng (giảm) trước, sau.

Ví dụ 6.12:

```
int b[2] = {1, 3}, *p = b;
(*p)++ ; // tăng (sau) giá trị nơi p trỏ ≡ tăng b[0] thành 2
++(*p) ; // tăng (trước) giá trị nơi p trỏ ≡ tăng a[0] thành 2
*(p++) ; // lấy giá trị nơi p trỏ (1) và tăng trỏ p (tăng sau), p → b[1]
*(++p) ; // tăng trỏ p (tăng trước), p → b[1] và lấy giá trị nơi p trỏ (3)
```

6.3.4. Hiệu của 2 con trỏ

Phép toán hiệu của 2 con trỏ chỉ thực hiện được khi p và q là 2 con trỏ cùng trỏ đến các phần tử của một dãy dữ liệu nào đó trong bộ nhớ.

Khi đó hiệu p - q là số thành phần giữa p và q.

Chú ý: p - q không phải là hiệu của 2 địa chỉ mà là số thành phần giữa p và q.

6.3.5. Phép toán so sánh

Các phép toán so sánh cũng được áp dụng đối với con trỏ, thực chất là so sánh giữa địa chỉ của hai nơi được trỏ bởi các con trỏ này.

Các phép so sánh <, <=, >, >= chỉ áp dụng cho hai con trỏ trỏ đến phần tử của cùng một mảng dữ liệu nào đó.

Thực chất của phép so sánh này chính là so sánh chỉ số của 2 phần tử được trỏ bởi 2 con trỏ đó.

Ví dụ 6.13a :

```
float a[100], *p, *q ;
p = a ; // p trỏ đến mảng (tức p trỏ đến a[0])
q = &a[3] ; // q trỏ đến phần tử thứ 3 (a[3]) của mảng
cout << (p < q) ; // 1
cout << (p + 3 == q) ; // 1
cout << (p > q - 1) ; // 0
cout << (p >= q - 2) ; // 0
for (p=a ; p < a+100; p++) cout << *p ; // in toàn bộ mảng a
```

Ví dụ 6.13b: in ra địa chỉ của biến được định nghĩa:

```
int main ()
{
  int  bien1;
  char bien2[10];
  cout << "Dia chi cua bien1 la: ";
  cout << &bien1 << endl;
  cout << "Dia chi cua bien2 la: ";
  cout << &bien2 << endl;
  return 0;
}
```

Ví dụ 6.13c: Minh họa một số phép toán quan trọng với con trỏ (định nghĩa biến con trỏ, gán địa chỉ của biến đến một con trỏ, truy cập các giá trị biến địa chỉ trong biến con trỏ..)

```
int main ()
{
  int  bien1 = 500;      // khai bao bien.
  int  *nv;              // bien con tro nv
  nv = &bien1;           // luu tru dia chi cua bien1 vao bien con tro nv
  cout << "Gia tri cua bien1 la: ";
  cout << bien1 << endl;
  // In dia chi duoc luu tru trong bien con tro nv
  cout << "Dia chi duoc luu tru trong bien con tro nv la: ";
  cout << nv << endl;
  // Truy cap gia tri co san tai dia chi cua bien con tro
  cout << "Gia tri cua *nv la: ";
  cout << * nv << endl;
  return 0;
}
```

6.4. Cấp phát bộ nhớ động

Cấp phát bộ nhớ động được sử dụng để cấp phát vùng nhớ cho các biến cục bộ, tham số của hàm.

Bộ nhớ được cấp phát tại thời điểm chương trình đang chạy, khi chương trình đi vào một khối lệnh. Các vùng nhớ được cấp phát sẽ được thu hồi khi chương trình đi ra khỏi một khối lệnh.

Kích thước vùng cần cấp phát cũng phải được cung cấp rõ ràng.

6.4.1. Toán tử new

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 6.14: khai báo

```
int *p1;
p1 = new int;
```

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

```
int *p2;
p2 = new int(10);
```

Ví dụ 6.15: kiểm tra việc cấp phát có thành công hay không thông qua kiểm tra con trỏ p bằng hay khác Null.

```
int main ()
    float *p ;
    int n ;
    cout << "Số lượng cần cấp phát = "; cin >> n;
    p = new double[n];
    if (p == Null) {
        cout << "Không đủ bộ nhớ" ;
        exit(0) ;
    }
```

Ghi chú: lệnh exit(0) cho phép thoát khỏi chương trình, để sử dụng lệnh này cần khai báo file tiêu đề <process.h>.

6.4.2. Toán tử delete

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 6.16:

```
int *p3 = new int(2); // Khai báo con trỏ p3, cấp phát bộ nhớ và gán giá trị ban đầu
                      cho p3 là 2.
delete p3;            // Giải phóng vùng nhớ vừa cấp cho p3.
```

Chú 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 *p4 = new int(5);     // Khai báo con trỏ p4, cấp phát bộ nhớ
                          // và gán giá trị ban đầu cho p4 là 5.
delete p4;                // Giải phóng vùng nhớ vừa cấp cho p4.
int A[5] = {5, 10, 15, 20, 25};
p4 = A;                   // Cho p4 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 *p1 = new int(5);     // *p1 = 5.
int *p2 = p1;             // p2 trỏ đến cùng địa chỉ p1.
*p2 += 3;                 // *p1 = *p2 = 8.
delete p1;                // Giải phóng cả p1 lẫn p2.
```

- 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 *p3 = new int(10);    // p3 được cấp bộ nhớ và *p3 = 10
*p3 = new int(20);        // p3 trỏ đến địa chỉ khác và *p3 = 20.
                          // địa chỉ cũ của p3 vẫn bị coi là bận.
```

Câu hỏi thảo luận

1. Trình bày cú pháp, và các cách sử dụng con trỏ.
2. Cho 2 ví dụ minh họa về dùng con trỏ để lưu địa chỉ của biến.
3. Trình bày các phép toán với con trỏ mỗi phép toán cho ví dụ minh họa.
4. Cho ví dụ minh họa về lấy giá trị của biến do con trỏ trỏ đến.
5. Trình bày cú pháp và cách dùng các toán tử new và delete trong cấp phát bộ nhớ động.

Bài tập vận dụng

1. Viết chương trình giải bài tập cộng hai số sử dụng con trỏ.
2. Sử dụng con trỏ viết chương trình hoán vị hai số A và B.
3. Chọn đáp án đúng:

a. int *pa = 20;

b. int *pa = new int{20};

c. int *pa = new int(20);

d. int *pa = new int[20];

4. Trong các khai báo con trỏ sau, chọn đáp án đúng:

a. int A*;

b. *int A;

c. int* A, B;

d. int* A, *B;

e. int *A, *B;

5. Cho p, q là các con trỏ trỏ đến biến nguyên x = 5.

Đặt *p = *q + 1; Hỏi *q ?

CHƯƠNG 7: MẢNG VÀ XÂU KÝ TỰ

Mục tiêu của chương

Nắm vững:

- Khái niệm, cách khai báo và cách sử dụng mảng một chiều, mảng hai chiều và mảng con trỏ.
- Khai báo xâu và sử dụng cấu trúc
- Vận dụng lý thuyết để giải các bài tập cụ thể.

Nội dung của chương

Nghiên cứu các kiến thức liên quan đến: mảng một chiều, mảng hai chiều, mảng con trỏ và xâu ký tự.

7.1. Mảng (Arrays)

Mảng là một tập hợp các phần tử cố định có cùng một kiểu, gọi là kiểu phần tử. Kiểu phần tử có thể là: ký tự, số, chuỗi ký tự;

Ta có thể chia mảng làm 2 loại: mảng một chiều và mảng nhiều chiều (trong tài liệu này tập trung vào mảng hai chiều).

Kiểu mảng cho phép giải quyết nhiều bài toán lập trình. Ví dụ: bài toán tìm kiếm, bài toán sắp xếp trên 1 dãy các số liệu..

7.1.1. Mảng một chiều

a. Khái niệm

Mảng là tập hữu hạn các phần tử có cùng kiểu dữ liệu được sắp kề nhau liên tục trong bộ nhớ. Tất cả các thành phần đều có cùng tên là tên của mảng.

Để phân biệt các thành phần với nhau, các thành phần sẽ được đánh số thứ tự từ 0 cho đến hết mảng. Khi cần nói đến thành phần cụ thể nào của mảng ta sẽ dùng tên mảng và kèm theo số thứ tự của thành phần đó. Ví dụ: mảng các số nguyên, các số thực, các kí tự, …

b. Khai báo

Có các dạng sau:

Dạng 1: Khai báo mảng với số phần tử xác định

<tên kiểu> <tên mảng>[số thành phần] ; //không khởi tạo

Dạng 2: Vừa khai báo vừa gán giá trị

<tên kiểu> <tên mảng>[số thành phần] = { dãy giá trị } ; //có khởi tạo

Dạng 3: Khai báo mảng với số phần tử không xác định và gán giá trị

<tên kiểu> <tên mảng>[ ] = { dãy giá trị } ; //có khởi tạo

Trong đó:

- tên kiểu là kiểu dữ liệu của các thành phần, các thành phần này có kiểu giống nhau. (ta còn gọi các thành phần là phần tử).

- <tên mảng> là một biến và để phân biệt với các biến thông thường ta còn gọi là biến mảng (tuân theo qui tắc đặt tên biến).

- [số thành phần] là một hằng số nguyên, cho biết số lượng phần tử tối đa trong mảng.

Dạng khai báo thứ 2: cho phép khởi tạo mảng bởi dãy giá trị trong cặp dấu {}, mỗi giá trị cách nhau bởi dấu phảy (,), các giá trị này sẽ được gán lần lượt cho các phần tử của mảng bắt đầu từ phần tử thứ 0 cho đến hết dãy.

Số giá trị có thể bé hơn số phần tử. Các phần tử mảng chưa có giá trị sẽ không được xác định cho đến khi trong chương trình nó được gán một giá trị nào đó.

Dạng khai báo thứ 3: cho phép vắng mặt số phần tử, trường hợp này số phần tử được xác định bởi số giá trị của dãy khởi tạo. Do đó nếu vắng mặt cả dãy khởi tạo là không được phép (chẳng hạn khai báo int a[] là sai).

Một mảng dữ liệu được lưu trong bộ nhớ bởi dãy các ô liên tiếp nhau.

Số lượng ô bằng với số thành phần của mảng và độ dài (byte) của mỗi ô đủ để chứa thông tin của mỗi thành phần. Ô đầu tiên được đánh thứ tự bởi 0, ô tiếp theo bởi 1, và tiếp tục cho đến hết. Như vậy nếu mảng có n thành phần thì ô cuối cùng trong mảng sẽ được đánh số là n - 1.

Ví dụ 7.1:

// Khai báo mảng 1 chiều gồm 10 phần tử thuộc kiểu nguyên.

int A[10];

Ta có thể coi mảng A là một dãy liên tiếp các phần tử trong bộ nhớ như sau:

Vị trí	0	1	2	3	4	5	6	7	8	9
Tên phần tử	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]

// Khai báo mảng 1 chiều gồm 5 phần tử thuộc kiểu nguyên và khởi tạo giá trị ban đầu cho các phần tử.

int A[5] = {3,5,4,6,2};

// Khai báo mảng 1 chiều, phần tử đầu = 3, các phần tử còn lại = 0.

int B[5] = {3};

// Khai báo mảng C chứa được tối đa 100 số nguyên dài.

long C[100] ;

// Khai báo biến chứa 2 vectơ a, b trong không gian 3 chiều.

float a[3] , b[3];

// Khai báo 2 phân số a, b; trong đó a = 1/3 và b = 3/5:

int a[2] = {1, 3}, b[2] = {3, 5};

c. Truy xuất các phần tử của mảng

Các phần tử mảng có thể được truy xuất thông qua chỉ số của nó trong mảng.

Các phần tử mảng được đánh số thứ tự bắt đầu từ 0, số thứ tự này gọi là chỉ số mảng. Các phần tử mảng có thể được truy xuất như sau:

<tên biến mảng>[chỉ số]

d. Nhập, xuất giá trị cho các phần tử mảng 1 chiều

Nhập dữ liệu cho mảng 1 chiều

```
cout << "Nhap so phan tu cua day:"; cin >> n;
cout << "Nhap gia tri cho cac phan tu cua day:\n"
for (i=0; i<n; i++)
{
   cout << "a[" << i << "] = " ;
   cin >> a[i];
}
```

Xuất các giá trị của mảng 1 chiều ra màn hình

```
cout << "Day da nhap la:" << endl;
for (i=0; i<n; i++) cout << a[i] << " ";
```

e. Một số ví dụ minh họa

Ví dụ 7.2: Cho một mảng A gồm 100 phần tử thuộc kiểu thực, hãy viết chương trình nhập mảng và sắp xếp mảng theo thứ tự tăng dần của dãy số.

```
#include <iostream>
#include<iomanip>
using namespace std;
int main ()
{
float a[100], i, j, n, tg;
```

```
cout << "Cho biết số phần tử n = " ; cin >> n ;
for (i=0; i<n; i++) {cout<<"a[" <<i<< "] = "; cin >> a[i] ;} // nhập dữ liệu
for (i=0; i<n; i++)
            {
             for (j=i+1; j<n; j++)
             if (a[i] > a[j]) { tg = a[i]; a[i] = a[j]; a[j] = tg; } // đổi chỗ
             }
for (i=0; i<n; i++)
cout << a[i] ; // in kết quả
return 0;
}
```

Ví dụ 7.3: Cho một mảng B gồm 100 phần tử thuộc kiểu thực, hãy viết chương trình nhập mảng và tìm phần tử có giá trị nhỏ nhất trong mảng. In ra phần tử này và vị trí của nó trong mảng.

```
#include <iostream>
#include<iomanip>
using namespace std;
int main ()
{
   float B[100], i, n, min, vt;
   cout << "Nhập số phần tử của dãy: " ; cin >> n;
   for (i=0; i<n; i++)
      {
      cout << "B[" << i << "] = " ; cin >> B[i];
      }
   min = B[0]; vt = 0;
   for (i=1; i<n; i++)
   if (B[i] < min )
      {
      min = B[i]; vt = i;
      }
   cout << "Số nhỏ nhất là " << min << "tại vị trí " << vt;
```

}

Ví dụ 7.4: Cho một mảng a gồm 50 phần tử thuộc kiểu thực, hãy viết chương trình nhập mảng và đếm số phần tử dương trong mảng.

```
# include <iostream>
# include<iomanip>
using namespace std;
int main ()
{
   float a[50]; int i, n, dem;
   cout << "Nhap so phan tu cua mang: " ; cin >> n;
   for (i=0; i<n; i++)
      {
      cout << "a[" << i << "] = " ; cin >> a[i];
      }
   dem = 0 ;
   for (i=0; i<n; i++)
   if (a[i]>0) dem++;
   cout << "Day da nhap la:" << endl;
   for (i=0; i<n; i++)
   cout << a[i] << " ";
   cout << "So phan tu duong trong mang la " << dem << endl;
   return 0;
}
```

7.1.2. Mảng hai chiều

a. Khái niệm

C++ đưa ra kiểu dữ liệu mảng hai chiều để biểu diễn các loại dữ liệu phức tạp như ma trận hoặc các bảng biểu có nhiều tiêu chí, hoặc các tọa độ 2 chiều.

Mảng 2 chiều m dòng n cột được xem như là một bảng hình chữ nhật chứa m*n phần tử cùng kiểu dữ liệu (còn gọi là ma trận m*n). Nó là sự mở rộng trực tiếp của mảng 1 chiều. Nói cách khác, mảng 2 chiều là mảng 1 chiều mà mỗi phần tử của nó là một mảng 1 chiều.

b. Khai báo

<tên kiểu> <tên mảng> [<số dòng>] [<số cột>];

- <tên kiểu> là kiểu dữ liệu của các thành phần, các thành phần này có kiểu giống nhau.

- <tên mảng> tuân theo qui tắc đặt tên biến.

Trong khai báo cũng có thể được khởi tạo bằng dãy các dòng giá trị, các dòng cách nhau bởi dấu phẩy, mỗi dòng được bao bởi cặp ngoặc {} và toàn bộ giá trị khởi tạo nằm trong cặp dấu {}.

Ví dụ 7.4:

//Khai báo mảng 2 chiều gồm 6 phần tử thuộc kiểu nguyên (2 dòng 3 cột).

int A[2][3];

//Khai báo mảng 2 chiều và khởi tạo giá trị ban đầu cho các phần tử của mảng.

int B[2][3] = {3,5,6,2,4,1};

//Khai báo mảng 2 chiều và khởi tạo giá trị ban đầu cho các phần tử của mảng 2 chiều.

int C[2][3] = {{3,5,6},{2,4,1}};

//Khai báo mảng và khởi tạo mảng 2 chiều với tất cả các phần tử có giá trị bằng 0.

int D[2][3] = {0};

//Khai báo 2 ma trận 4 hàng 5 cột A, B chứa các số nguyên.

int A[3][4], B[3][4];

*// Khai báo mảng 2 chiều có 4*5 phần tử là số thực.*

float m[4][5];

Trong trường hợp này, ta đã khai báo cho một ma trận có tối đa là 4 dòng, mỗi dòng có tối đa là 5 cột.

Dòng\Cột	0	1	2	3	4
0	m[0][0]	m[0][1]	m[0][2]	m[0][3]	m[0][4]
1	m[1][0]	m[1][1]	m[1][2]	m[1][3]	m[1][4]
2	m[2][0]	m[2][1]	m[2][2]	m[2][3]	m[2][4]
3	m[3][0]	m[3][1]	m[3][2]	m[3][3]	m[3][4]

c. Truy xuất các phần tử của mảng 2 chiều

Ta có thể truy xuất một phần tử của mảng hai chiều bằng cách viết ra tên mảng theo sau là hai chỉ số đặt trong hai cặp dấu ngoặc vuông. Chẳng hạn ta viết m[2][3].

Các phần tử mảng có thể được truy xuất như sau:

Tên mảng[chỉ_số_hàng][chỉ_số_cột]

d. Nhập, xuất giá trị cho các phần tử mảng 2 chiều

Nhập dữ liệu cho mảng hai chiều

```
cout << "Nhap so hang, so cot cua ma tran:" ;
cin >> m >> n;
cout << "Nhap gia tri cho cac phan tu cua ma tran:\n";
for (i=0; i<m; i++)
for (j=0; j<n; j++)
{
    cout << "a[" << i << "][" << j << "] = " ;
    cin >> a[i][j];
}
```

Xuất các giá trị của mảng hai chiều ra màn hình

```
cout << "Ma tran da nhap la:" << endl;
for (i=0; i<m; i++)
{
    for (j=0; j<n; j++) cout << a[i][j] << " ";
    cout << endl;
}
```

e. Một số ví dụ minh họa

Ví dụ 7.5: Viết chương trình nhập, in và tìm phần tử lớn nhất của một ma trận.

```
#include <iostream>
#include<iomanip>
using namespace std;
int main ()
{
float a[10][10] ;
int m, n ; // số hàng, cột của ma trận
```

```
int i, j ; // các chỉ số trong vòng lặp
int amax, imax, jmax ; // số lớn nhất và chỉ số của nó
cout << "Nhập số hàng và cột: " ; cin >> m >> n ;
for (i=0; i<m; i++)
for (j=0; j<n; j++)
   {
   cout << "a[" << i << "," << j << "] = " ; cin >> a[i][j] ;
   }
   amax = a[0][0]; imax = 0; jmax = 0;
   for (i=0; i<m; i++)
   for (j=0; j<n; j++)
   if (amax < a[i][j])
{
   amax = a[i][j]; imax = i; jmax = j;
}
cout << "Ma trận đã nhập\n" ;
cout << setiosflags(ios::showpoint) << setprecision(1) ;
for (i=0; i<m; i++)
for (j=0; j<n; j++)
{
   if (j==0) cout << endl;
   cout << setw(6) << a[i][j] ;
}
cout << "Số lớn nhất là " << setw(6) << amax << endl;
cout << "tại vị trí (" << imax << "," << jmax << ")" ;
return 0;
}
```

Ví dụ 7.6: Viết chương trình tính tổng mỗi hàng, mỗi cột của một ma trận có kích cỡ n x m, và nếu là ma trận vuông thì tính tổng đường chéo.

```
#include <iostream>
#include<iomanip>
using namespace std;
```

```
int main ()
{
  int A[10][10], m, n, x, y, sum=0;
  //Tao mot ma tran A
  cout << "Nhap so hang va so cot cua ma tran A : \n"; cin>>n>>m;
  cout << "Nhap cac phan tu cua ma tran A : \n";
  for (x=1;x<n+1;++x)
  for (y=1;y<m+1;++y)
  cin >>A[x][y];
   //Tim tong gia tri cua hang
   for (x=1;x<n+1;++x)
   {
     A[x][m+1]=0;
     for (y=1;y<m+1;++y)
     A[x][m+1]=A[x][m+1]+A[x][y];
   }
//Tim tong gia tri cua cot
   for (y=1;y<m+1;++y)
   {
     A[n+1][y]=0;
     for (x=1;x<n+1;++x)
     A[n+1][y]+=A[x][y];
   }
   cout << "\nMa tran A:Tong hang (cot cuoi)" << " va Tong cot (hang cuoi) : \n";
   for (x=1;x<n+1;++x)
  {
      for (y=1;y<m+2;++y)
     cout << A[x][y] << "     ";
     cout << "\n";
  }
   //In tong moi hang
   x = n+1;
```

```
for (y=1;y<m+1;++y)
cout << A[x][y] << "     ";
cout << "\n";
if (m==n)
{
for (x=1; x<m+1; x++)
  for (y=1; y<n+1; y++)
    if  (x==y)
      sum+=A[x][y];
    else
      if (y==m-(x+1))
      sum+=A[x][y];
 }
cout << "Tong cac phan tu tren duong cheo la : " << sum << endl;
return 0;
}
```

7.1.3. Mảng con trỏ

7.1.3.1. Con trỏ và mảng một chiều

Giữa mảng và con trỏ có một sự liên hệ rất chặt chẽ. Những phần tử của mảng có thể được xác định bằng chỉ số trong mảng, bên cạnh đó chúng cũng có thể được xác lập qua biến con trỏ.

Ví dụ 7.7: int C[30]; // khi đó địa chỉ của mảng C sẽ trùng với địa chỉ phần tử đầu tiên của mảng C (là &C[0]):

```
C = &C[0];
```

Quan hệ giữa con trỏ và mảng

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

Ví dụ 7.8:

```
int B[5] = {5, 7, 9, 11, 13};
int *pb = B;
```

Con trỏ pb sẽ trỏ đến mảng B, tức là trỏ đến địa chỉ của phần tử B[0], cho nên hai khai báo sau là tương đương:

```
pb = B;
```

pb = &B[0];

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

Phép toán trên con trỏ và mảng

Khi một con trỏ trỏ đến mảng, thì các phép toán tăng giảm trên con trỏ sẽ tương ứng với phép dịch chuyển trên mảng.

Ví dụ 7.9: int A[5] = {5, 7, 9, 11, 13};

int *pa = &A[2];

Con trỏ pa sẽ trỏ đến địa chỉ của phần tử A[2] và giá trị của pa là: *pa = A[2] = 9.

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].

Phép toán: pa = pa – 1; sẽ đưa con trỏ pa trỏ đến địa chỉ của phần tử A[1].

Chú 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.
- 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.

Ví dụ 7.10: Viết một thủ tục sắp xếp các phần tử của một mảng bằng con trỏ.

```
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;
    }
```

```
}
```

Ví dụ 7.11: Minh họa việc sử dụng mảng giống như con trỏ và sử dụng con trỏ giống như mảng.

```
#include <iostream>
#include<iomanip>
using namespace std;
int main ()
{
   int A[] = {9, 7, 12, 8, 6, 5}, n= 6;
   int *P; //Khai báo P là con trỏ kiểu int
   for (int i=0; i<n; i++) {//Xử lý mảng như con trỏ
      cout <<"Dia chi A["<<i<<"]="<<(A+i);
      cout <<"Gia tri A["<<i<<"]="<<*(A+i)<<endl;
      }
   P = A; //P trỏ đến A và xử lý P như mảng
   for (int i=0; i<n; i++) {
      cout <<"Dia chi A["<<i<<"]="<<&P[i];
      cout <<"Gia tri A["<<i<<"]="<<P[i]<<endl;
      }
   system("Pause");
   return 0;
 }
```

7.1.3.2. Con trỏ và mảng hai chiều

Ví dụ 7.12a:

```
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];

Ví dụ 7.12b: Cho phép nhập và in một mảng 2 chiều m*n (m dòng, n cột) thông qua con trỏ p. Nhập liên tiếp m*n số vào mảng và in thành ma trận m dòng, n cột.

```
int main()
{
clrscr();
float a[m][n], *p;
int i, j;
p = (float*) a;
for (i=0; i<m*n; i++) cin >> *(p+i); // nhập như dãy mxn phần tử
*(p+2*n+3) = 100; *(p+4*n) = 100; // gán a[2,3] = a[4][0] = 100
for (i=0; i<m; i++) // in lại dưới dạng ma trận
    {
    for (j=0; j<n; j++) cout << *(p+i*n+j);
    cout << endl;
    }
return 0;
}
```

Chú ý: việc lấy địa chỉ phần tử a[i][j] của mảng thực a là không chính xác. Tức: viết p = &a[i][j] có thể dẫn đến kết quả sai.

7.1.3.3. 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;

7.1.3.4. Mảng c**on trỏ**

a. Khái niệm chung:

Thực chất một con trỏ cũng là một biến thông thường có tên gọi (ví dụ p, q, …), do đó cũng giống như biến, nhiều biến cùng kiểu có thể tổ chức thành một mảng với tên gọi chung, ở đây cũng vậy nhiều con trỏ cùng kiểu cũng được tổ chức thành mảng.

Như vậy mỗi phần tử của mảng con trỏ là một con trỏ trỏ đến một mảng nào đó. Nói cách khác một mảng con trỏ cho phép quản lý nhiều mảng dữ liệu cùng kiểu.

b. Cách khai báo:

<kiểu> *a[size];

Ví dụ 7.13: int *a[10];

Khai báo một mảng chứa 10 con trỏ. Mỗi con trỏ a[i] chứa địa chỉ của một mảng nguyên nào đó.

7.1.3.5. Mảng **xâu kí tự**

Là trường hợp riêng của mảng con trỏ nói chung, trong đó kiểu cụ thể là char.

Mỗi thành phần mảng là một con trỏ trỏ đến một xâu kí tự, có nghĩa các thao tác tiến hành trên *a[i] như đối với một xâu kí tự.

Ví dụ 7.14 : Nhập vào và in ra một bài thơ.

```
int main()
{
    clrscr();
    char *dong[100]; // khai báo 100 con trỏ kí tự (100 dòng)
    int i, n;
    cout << "so dong = "; cin >> n ; // nhập số dòng thực sự
    cin.ignore(); // loại dấu ↵ trong lệnh cin ở trên
    for (i=0; i<n; i++)
    {
        dong[i] = new char[80]; // cấp bộ nhớ cho dòng i
        cin.getline(dong[i],80); // nhập dòng i
```

```
    }
    for (i=0; i<n; i++) cout << dong[i] << endl; // in kết quả
    getch();
}
```

7.2. Xâu ký tự

7.2.1. Định nghĩa

Xâu (chuỗi) ký tự, gọi tắt là xâu (chuỗi):

Là một dãy ký tự liên tiếp, tạo bởi các chữ cái, chữ số, ký hiệu (+, -, &, !, …) và dấu trắng (dấu cách, dấu xuống dòng, …)

Xâu phải được đặt giữa hai dấu nháy kép "…"

7.2.2. Khai báo

Có các dạng sau:

Dạng 1: **char <tên xâu>[độ dài]** ; // không khởi tạo.

Dạng 2: **char <tên xâu>[độ dài] = xâu kí tự** ; // có khởi tạo.

Dạng 3: **char <tên xâu>[] = xâu kí tự** ; // có khởi tạo.

- Độ dài mảng là số kí tự tối đa có thể có trong xâu. Độ dài thực sự của xâu chỉ tính từ đầu mảng đến dấu kết thúc xâu (không kể dấu kết thúc xâu '\0').

- Dạng 2: khai báo xâu có kèm theo khởi tạo xâu, đó là dãy kí tự đặt giữa cặp dấu nháy kép.

- Dạng 3: dạng khai báo này tự chương trình sẽ quyết định độ dài của mảng bởi xâu khởi tạo (bằng độ dài xâu + 1).

Ví dụ 7.15:

```
char hoten[26] ;            // xâu họ tên chứa tối đa 25 kí tự.
char monhoc[31] = "NNLT C++" ;
char thang[] = "Mười hai" ; // độ dài mảng = 9.
charstr[] = "Hello";        //Vừa khai báo vừa khởi đầu cho xâu ký tự.
```

Phương thức nhập xâu

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

cin.getline(s,n) để nhập xâu kí tự.

Trong đó:

s là xâu cần nhập nội dung.

n-1 là số kí tự tối đa của xâu.

7.2.3. Một số hàm xử lí xâu (#include <string.h>)

- **strcpy(s, t) ;**

Hàm sẽ sao chép toàn bộ nội dung của xâu t vào cho xâu s.

Chú ý: để sử dụng hàm này cần đảm bảo độ dài của mảng s ít nhất cũng bằng độ dài của mảng t. Trong trường hợp ngược lại kí tự kết thúc xâu sẽ không được ghi vào s và điều này có thể gây treo máy khi chạy chương trình.

- **strncpy(s, t, n);**

Sao chép n kí tự của t vào s.

Hàm này chỉ làm nhiệm vụ sao chép, không tự động gắn kí tự kết thúc xâu cho s.

Do vậy người sử dụng phải thêm câu lệnh đặt kí tự '\0' vào cuối xâu s sau khi sao chép xong.

- **strcat(s, t);**

Nối một bản sao của t vào sau s (thay cho phép +).

Hàm sẽ loại bỏ kí tự kết thúc xâu s trước khi nối thêm t.

Việc nối sẽ đảm bảo lấy cả kí tự kết thúc của xâu t vào cho s (nếu s đủ chỗ) vì vậy người sử dụng không cần thêm kí tự này vào cuối xâu.

- **strncat(s, t, n);**

Nối bản sao n kí tự đầu tiên của xâu t vào sau xâu s.

Hàm tự động đặt thêm dấu kết thúc xâu vào s sau khi nối xong.

Cũng giống strcat hàm đòi hỏi độ dài của s phải đủ chứa kết quả.

Tương tự, có thể sử dụng cách viết strncat(s, t+k, n) để nối n kí tự từ vị trí thứ k của xâu t cho s.

- **strcmp(s, t);**

Hàm so sánh 2 xâu s và t (thay cho các phép toán so sánh).

Giá trị trả lại là hiệu 2 kí tự khác nhau đầu tiên của s và t.

Từ đó, nếu s1 < s2 thì hàm trả lại giá trị âm, bằng 0 nếu s1==s2, và dương nếu s1 > s2.

- **strncmp(s, t) ;**

Giống hàm strcmp(s, t) nhưng chỉ so sánh tối đa n kí tự đầu tiên của hai xâu.

- **strcmpi(s, t) ;**

Giống như hàm strcmp(s, t) nhưng không phân biệt chữ hoa, thường.

- **strupr(s);**

Hàm đổi xâu s thành in hoa, và cũng trả lại xâu in hoa đó.

- **strlwr(s);**

Hàm đổi xâu s thành in thường, kết quả trả lại là xâu s.

- **strlen(s) ;**

Hàm trả giá trị là độ dài của xâu s.

7.2.4. Một số ví dụ

Ví dụ 7.16a: Viết chương trình minh họa một số hàm sao chép, nối chuỗi và tính độ dài của chuỗi như sau:

```
#include <iostream>
#include<iomanip>
using namespace std;
int main ()
    {
        char chuoi1[10] = "Khoa";
        char chuoi2[10] = "CNTT";
        char chuoi3[10];
        int  len ;
        // sao chep chuoi1 vao trong chuoi3
        strcpy( chuoi3, chuoi1);
        cout << "strcpy( chuoi3, chuoi1) : " << chuoi3 << endl;
        // noi hai chuoi: chuoi1 va chuoi2
        strcat( chuoi1, chuoi2);
        cout << "strcat( chuoi1, chuoi2): " << chuoi1 << endl;
        // tong do dai cua chuoi1 mot sau khi thuc hien noi chuoi
        len = strlen(chuoi1);
        cout << "Dung ham strlen(chuoi1) de tinh do dai chuoi1: " << len << endl;
        return 0;
    }
```

Ví dụ 7.16b: Viết chương trình thống kê số chữ 'c' xuất hiện trong xâu s.

```
#include <iostream>
#include<iomanip>
using namespace std;
int main ()
    {
       const int MAX = 100;
       char s[MAX+1];
       int sokitu = 0;
       cin.getline(s, MAX+1);
       for (int i=0; i < strlen(s); i++)
       if (s[i] = 'c ') sokitu++;
       cout << "Số kí tự = " << sokitu << endl ;
    }
```

Câu hỏi thảo luận

1. Trình bày cú pháp, ý nghĩa của mảng 1 chiều, mảng 2 chiều.

2. Trình bày khái niệm và cách khai báo kiểu xâu ký tự.

3. Trình bày cú pháp và công dụng của các hàm trong xâu ký tự. Mỗi hàm cho một ví dụ minh họa.

4. Trình bày quy trình nhập, xuất dữ liệu đối với mảng 1 chiều, mảng 2 chiều.

Bài tập vận dụng

1. Cho mảng A có n phần tử là những giá trị nguyên (n: hằng số). Viết chương trình để

- Tính trung bình cộng các phần tử dương trong mảng.
- Sắp xếp mảng theo thứ tự giảm dần.

2. Viết chương trình nhập vào một dãy gồm n số nguyên. In ra màn hình:

- Một dòng gồm các số lẻ của dãy
- Một dòng gồm các số chẵn của dãy
- Một dòng gồm các số >=100 của dãy
- Một dòng gồm các số <100 của dãy.

3. Viết chương trình nhập một dãy n số nguyên (n: hằng số). In ra màn hình các giá trị khác nhau của dãy số này.

Ví dụ: - Nhập vào 5 20 15 5 20 17

- In ra: 5 20 15 17

4. Cho một dãy gồm n số nguyên dương có 3 chữ số, hãy sắp xếp dãy theo thứ tự tăng dần của chữ số hàng đơn vị; hàng đơn vị bằng nhau thì sắp xếp theo thứ tự tăng dần của hàng chục; hàng đơn vị và hàng chục bằng nhau thì sắp theo thứ tự tăng dần của hàng trăm.

5. Cho 2 mảng A, B đã được sắp xếp theo thứ tự tăng dần. Hãy trộn hai mảng đó lại với nhau để có mảng thứ 3 là mảng C với điều kiện mảng C cũng được sắp xếp theo thứ tự tăng dần ngay sau khi trộn.

6. Cho dãy đã được sắp tăng dần. Chèn thêm vào dãy phần tử x sao cho dãy vẫn sắp xếp tăng dần.

7. Nhập ma trận A và in ra ma trận đối xứng của nó.

8. Cho một ma trận thực kích thước m*n. Tìm:

- Số nhỏ nhất, lớn nhất (kèm chỉ số) của ma trận.
- Số nhỏ nhất, lớn nhất (kèm chỉ số) của đường chéo chính của ma trận.

9. Tính tổng các phần tử có trong mảng sử dụng con trỏ.

10. Tính tổng các phần tử chẵn có trong mảng sử dụng con trỏ.

11. Tính tổng các phần tử nguyên tố có trong mảng sử dụng con trỏ.

12. Tìm phần tử lớn nhất có trong mảng sử dụng con trỏ.

13. Đếm số phần tử chẵn có trong mảng sử dụng con trỏ.

14. Đếm số phần tử lớn nhất có trong mảng sử dụng con trỏ.

15. In ra vị trí của phần tử lớn nhất đầu tiên có trong mảng sử dụng con trỏ.

16. Thêm một phần tử vào đầu mảng sử dụng con trỏ.

17. Viết chương trình khởi tạo mảng và nhập danh sách tên người và sử dụng con trỏ để trỏ tới vị trí thứ n thì sẽ ra tên người đó.

18. Hãy nhập một xâu kí tự. In ra màn hình đảo ngược của xâu đó.

19. In ra vị trí kí tự trắng đầu tiên từ bên trái (phải) một xâu kí tự.

20. Nhập xâu. Tính số từ có trong xâu. In mỗi dòng một từ.

21. Nhập xâu họ tên, in ra họ, tên dưới dạng viết hoa.

22. Thay kí tự x trong xâu s bởi kí tự y (s, x, y được đọc vào từ bàn phím)

23. Xoá mọi kí tự x có trong xâu s (s, x được đọc vào từ bàn phím).

24. Cho xâu kí tự (dạng con trỏ) s. Hãy thống kê tần xuất xuất hiện của các kí tự có trong s. In ra màn hình theo thứ tự giảm dần của các tần xuất (tần xuất là tỉ lệ % số lần xuất hiện của x trên tổng số kí tự trong s).

25. Cho p, q là các con trỏ cùng trỏ đến kí tự c.

Đặt *p = *q + 1. Có thể khẳng định: *q = *p - 1 ?

26. Cho p, q là các con trỏ trỏ đến biến nguyên x = 5.

Đặt *p = *q + 1; Hỏi *q ?

27. 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 ?

CHƯƠNG 8: KIỂU DỮ LIỆU CẤU TRÚC

Mục tiêu của chương

Nắm vững:

- Khái niệm, cách khai báo kiểu dữ liệu cấu trúc.
- Truy nhập các thành phần kiểu cấu trúc.
- Đặt tên kiểu dữ liệu bằng lệnh typedef.
- Vận dụng lý thuyết để giải các bài tập cụ thể.

Nội dung của chương

Nghiên cứu dữ liệu kiểu cấu trúc.

8.1. Khái niệm và định nghĩa cấu trúc

Cấu trúc (struct) là kiểu dữ liệu phức hợp được xây dựng từ những kiểu dữ liệu khác.

Kiểu struct cho phép lưu trữ thông tin về một đối tượng với đầy đủ các thuộc tính của nó trong một biến. Mỗi thuộc tính là một biến thành phần. Kiểu dữ liệu của các biến này có thể là kiểu cơ sở hay kiểu mảng, thậm chí một kiểu struct khác.

Ví dụ dữ liệu kiểu cấu trúc là danh sách sinh viên trong đó mỗi sinh viên được mô tả bằng: mã số, họ tên, năm sinh, địa chỉ, năm nhập học, điểm trúng tuyển.

8.2. Khai báo cấu trúc

- Để tạo ra một kiểu cấu trúc người sử dụng cần phải khai báo tên của kiểu (là một tên gọi do người sử dụng 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 cấu trúc>

{

<Kiểu dữ liệu 1> <Tên thuộc tính 1>;

<Kiểu dữ liệu 2> <Tên thuộc tính 2>;

…

<Kiểu dữ liệu n> <Tên thuộc tính n>;

};

Trong đó:

- struct là từ khóa mà ngôn ngữ C++ cung cấp.

- <Tên cấu trúc>: là tên do người dùng tự định nghĩa, tuân thủ theo quy tắc đặt tên biến trong C++. Tên này sẽ trở thành tên của kiểu dữ liệu có cấu trúc tương ứng.
- <Thuộc tính>: mỗi thuộc tính của cấu trúc được khai báo như khai báo một biến thuộc kiểu dữ liệu thông thường, gồm có kiểu dữ liệu và tên biến tương ứng. Mỗi khai báo thuộc tính phải kết thúc bằng dấu chấm phẩy ";" như một câu lệnh C++ thông thường.

Khai báo biến kiểu cấu trúc cũng giống như khai báo các biến kiểu cơ sở dưới dạng:

<tên cấu trúc> <danh sách biến> ;

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

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

Ví dụ 8.1: Khai báo cấu trúc có tên "Tenhanvien" gồm mã số, họ tên, lương có thể viết như sau:

```
struct Tenhanvien
{
int maso;
char hoten[30];
float luong;
};
```

hoặc:

```
struct Tenhanvien {int maso; char hoten[30]; float luong;}
```

Ví dụ 8.2: Khai báo cấu trúc Date gồm ngày (day), tháng (month), năm (year).

```
struct Date
{
int day ;
int month;
int year;
}; nghihe = { 1, 6, 2019 } ;
```

Một biến nghihe (nghỉ hè) cũng được khai báo kèm cùng kiểu này và được khởi tạo (các giá trị khởi tạo này lần lượt gán cho các thành phần theo đúng thứ tự trong khai báo, tức day = 1, month = 6 và year = 2019.

Ví dụ 8.3: Khai báo cấu trúc Sinhvien gồm có các trường hoten (họ tên), ns (ngày sinh), gt(giới tính) dưới dạng số (qui ước 1: nam, 2: nữ), diem (điểm).

```
struct Sinhvien
{
char hoten[25] ;
ngaythang ns;
int gt;
float diem ;
} x, *p, TIN12[30];
```

Khai báo cùng với cấu trúc Sinhvien có các biến x, con trỏ p và mảng TIN12 với 30 phần tử kiểu Sinhvien.

Ví dụ 8.4: Khai báo cấu trúc có tên Books

```
struct Books
{
  char  tieude[50];
  char  tacgia[50];
  char  chude[100];
  int   book_id;
};
```

Ví dụ 8.5:

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 ; }
```

8.3. Đặt tên kiểu dữ liệu bằng typedef

Để tránh phải dùng từ khoá struct mỗi khi khai báo biến cấu trúc, ta có thể dùng từ khóa typedef khi định nghĩa cấu trúc:

typedef struct

{

<Kiểu dữ liệu 1> <Tên thuộc tính 1>;

<Kiểu dữ liệu 2> <Tên thuộc tính 2>;

…

<Kiểu dữ liệu n> <Tên thuộc tính n>;

} <Tên kiểu dữ liệu cấu trúc>;

Trong đó:

Tên kiểu dữ liệu cấu trúc: là tên kiểu dữ liệu của cấu trúc vừa định nghĩa. Tên này sẽ được dùng như một kiểu dữ liệu thông thường khi khai báo biến cấu trúc.

Ví dụ 8.6: Khai báo cấu trúc Nhanvien dùng từ khoá typedef để định nghĩa cấu trúc như sau:

```
typedef struct
{
    char tennv[20]; // Tên nhân viên
    int tuoi;           // Tuổi nhân viên
    char chuvu[20]; // Chức vụ của nhân viên
    float luong;        // Lương của nhân viên
} Nhanvien;
```

Khi đó, muốn có hai biến là Nhvien1 và Nhvien2 có kiểu cấu trúc Nhanvien, ta chỉ cần khai báo như sau mà không cần từ khoá struct:

```
Nhanvien  Nhvien1, Nhvien2;
```

Lệnh typedef thường được sử dụng để định nghĩa các kiểu dữ liệu phức hợp thành một tên duy nhất.

Ví dụ 8.7: lệnh typedef int * PTR_INT;

định nghĩa kiểu dữ liệu con trỏ nguyên. Sau này khi cần khai báo một con trỏ nguyên, ta chỉ cần viết:

```
PTR_INT px; // tương đương với khai báo int * px;
```

Ví dụ 8.8: ta định nghĩa phân số như là kiểu cấu trúc

```
typedef struct
{
    int tuso, mauso;
}Phanso;
```

Sau đó, ta định nghĩa các biến kiểu Phanso

```
Phanso ps1, ps2;
```

8.4. Thao tác trên biến cấu trúc

Các thao tác trên cấu trúc bao gồm:

- Khai báo và khởi tạo giá trị ban đầu cho biến cấu trúc.
- Truy nhập đến các thuộc tính của cấu trúc.

8.4.1. Khởi tạo giá trị ban đầu cho cấu trúc

Khởi tạo biến có cấu trúc đơn

Biến cấu trúc được khai báo theo các cách sau:

- Nếu định nghĩa không dùng typedef

struct <Tên cấu trúc> <tên biến>;

- Nếu định nghĩa dùng typedef

<Tên kiểu dữ liệu cấu trúc> <tên biến>;

Ngoài ra, ta có thể khởi tạo các giá trị cho các thuộc tính của cấu trúc ngay khi khai báo bằng các cú pháp sau:

- Nếu định nghĩa không dùng typedef

```
struct <Tên cấu trúc> <tên biến> = {
<giá trị thuộc tính 1>,
<giá trị thuộc tính 2>,
…
<giá trị thuộc tính n>
};
```

Hoặc:

```
// Nếu định nghĩa dùng typedef
<Tên kiểu dữ liệu cấu trúc> <tên biến> = {
<giá trị thuộc tính 1>,
<giá trị thuộc tính 2>,
```

…

<giá trị thuộc tính n>

};

Trong đó:

Giá trị thuộc tính: là giá trị khởi đầu cho mỗi thuộc tính, có kiểu phù hợp với kiểu dữ liệu của thuộc tính. Mỗi giá trị của thuộc tính được phân cách bằng dấu phẩy “,”.

Ví dụ 8.9: Minh họa định nghĩa cấu trúc:

```
typedef struct {
    char tennv[20];  // Tên nhân viên
    int tuoi;        // Tuổi nhân viên
    char chuvu[20]; // Chức vụ của nhân viên
    float luong;     // Lương của nhân viên
} Nhanvien;
```

Có thể khai báo và khởi tạo cho một biến như sau:

```
Nhanvien nv1 = {
    “Nguyen Van A”,
    27,
    “To truong”,
    8000000
};
```

Khởi tạo các biến có cấu trúc lồng nhau

Trong trường hợp các cấu trúc lồng nhau, phép khởi tạo cũng thực hiện như thông thường với phép khởi tạo cho tất cả các cấu trúc con.

Ví dụ 8.10: Với khai báo cấu trúc như sau:

```
typedef struct {
    int day;
    int month;
    int year;
} Date;
```

và:

```
typedef struct {
    char tennv[20];  // Tên nhân viên
```

```
    Date ngsinh;            // Ngày sinh của nhân viên
    char chucvu [20];       // Chức vụ của nhân viên
    float luong;            // Lương của nhân viên
} Nhanvien;
```

Thì khai báo và khởi tạo một biến có kiểu Nhanvien có thể thực hiện như sau:

```
Nhanvien nv1 = {
    "Nguyen Van A",
    {15, 05, 2000},         // Khởi tạo cấu trúc con
    "To truong",
    8000000
};
```

8.4.2. Truy nhập đến thuộc tính của cấu trúc

Truy nhập đến thuộc tính của cấu trúc được thực hiện bằng cú pháp:

<Tên biến cấu trúc>.<tên thuộc tính>

Ví dụ 8.11: Với một biến cấu trúc kiểu Nhanvien đơn:

```
Nhanvien nv1 = {
    "Nguyen Van A",
    27,
    "To truong",
    8000000
    };
```

ta có thể truy xuất như sau:

```
cout << nv1.name;   // hiển thị ra "Nguyen Van A"
nv1.age += 1;       // Tăng số tuổi lên 1
```

Đối với kiểu cấu trúc lồng nhau, phép truy nhập đến thuộc tính được thực hiện lần lượt từ cấu trúc cha đến cấu trúc con.

Ví dụ 8.12: Với một biến cấu trúc kiểu Nhanvien lồng nhau:

```
Nhanvien nv1 = {
    "Nguyen Van A",
    {15, 05, 2000},
    "To truong",
    8000000
```

```
};
```

ta có thể truy xuất như sau:

```
cout << nv1.name;   // hiển thị ra “Nguyen Van A”
nv1. ngsinh.day = 16; // Sửa lại ngày sinh thành 16
```

Ví dụ 8.13: Minh họa cách truy cập các thành phần của cấu trúc trong C++.

```
struct Books
{
  char  tieude[50];
  char  tacgia[50];
  char  chude[100];
  int   book_id;
};
int main ( )
{
  struct Books QuyenSach1;
  // chi tiet ve quyen sach thu nhat
  strcpy(QuyenSach1.tieude, "Ngon ngu Lap trinh Java");
  strcpy(QuyenSach1.tacgia, "Nguyen Van A");
  strcpy(QuyenSach1.chude, "Lap trinh");
  QuyenSach1.book_id = 1225;
  // in thong tin ve QuyenSach1
  cout << "Tieu de cua Quyen sach thu nhat la: " << QuyenSach1.tieude <<endl;
  cout << "Tac gia cua Quyen sach thu nhat la: " << QuyenSach1.tacgia <<endl;
  cout << "Chu de cua Quyen sach thu nhat la: " << QuyenSach1.chude <<endl;
  cout << "ID cua Quyen sach thu nhat la: " << QuyenSach1.book_id <<endl;
  cout <<"\n\n================================\n\n" <<endl;
  return 0;
}
```

Câu hỏi thảo luận

1. Trình bày cú pháp khai báo kiểu cấu trúc, cho ví dụ minh họa một kiểu cấu trúc.

2. Trình bày cú pháp khai báo kiểu cấu trúc bằng typedef, cho ví dụ.

3. Trình bày cú pháp truy nhập đến các thuộc tính của cấu trúc.

Bài tập vận dụng

1. Hãy định nghĩa kiểu dữ liệu PhanSo đại diện cho kiểu phân số. Viết chương trình cho phép người dùng thực hiện các phép cộng, trừ, nhân, chia 2 phân số.

2. Viết chương trình thực hiện phân tích thống kê cho một lớp học khoảng 20 sinh viên. Thông tin của mỗi sinh viên bao gồm ID, tên, tuổi, điểm tổng kết học kì 1, điểm tổng kết học kì 2. Những thông tin cần thống kê bao gồm:

Điểm trung bình cuối năm của cả lớp.

Điểm tổng kết cuối năm của sinh viên nào là cao nhất.

Liệt kê danh sách những sinh viên có tiến bộ trong học tập (điểm tổng kết học kì 2 cao hơn điểm tổng kết học kì 1).

3. Viết chương trình nhập vào danh sách sản phẩm (MaSP, tên SP, đơn giá SP) mua trong tháng. Yêu cầu in ra danh sách sản phẩm vừa nhập, và danh sách những mặt hàng có giá bán <20.000.

4. Viết chương trình nhập vào danh sách cán bộ giảng viên của trường A gồm (mã giảng viên, họ tên, số môn dạy, học kỳ). Yêu cầu in ra danh sách giảng viên vừa nhập và danh sách giảng viên dạy 5 môn trở lên trong học kỳ.

5. Viết chương trình tạo struct công nhân gồm các trường: họ tên, năm sinh và giờ làm.

- Nhập một số n, tạo và nhập một mảng n công nhân.

- Liệt kê tên các công nhân nhận thưởng biết rằng công nhân nhận thưởng là các công nhân có giờ làm > 40 giờ.

6. Để quản lý các hộ dân của một phường, người ta lưu trữ các thông tin sau: Mã hộ (char[5]), Tên chủ hộ (char[25]), Số thành viên (int), Mức thu nhập (float). Viết chương trình thực hiện các công việc sau:

- Khai báo dữ liệu kiểu cấu trúc lưu trữ thông tin cho một hộ dân.

- Nhập vào từ bàn phím một danh sách các hộ dân

- Hiển thị thông tin những hộ dân có số thành viên lớn hơn 5 (với đầy đủ thông tin, dưới dạng bảng).

7. Cho một danh sách n sinh viên (0<n<100). Thông tin về mỗi sinh viên gồm Mã sv (char[8]), Họ tên (char[30]), năm sinh (int), điểm trung bình (float). Viết chương trình thực hiện các công việc sau:

- Khai báo dữ liệu kiểu cấu trúc lưu trữ thông tin cho một sinh viên.
- Nhập vào từ bàn phím một danh sách sinh viên.
- Sắp xếp danh sách sinh viên theo thứ tự giảm dần của tuổi, in ra danh sách sau khi đã sắp xếp (với đầy đủ thông tin dưới dạng bảng).

8. Để quản lý các vận động viên của một đội bóng, người ta lưu trữ các thông tin sau: Mã vận động viên (char[5]), Tên vận động viên (char[25]), Tuổi (int), Vị trí (char[15]). Viết chương trình thực hiện các công việc sau:

- Khai báo dữ liệu kiểu cấu trúc lưu trữ thông tin cho một vận động viên.
- Nhập vào từ bàn phím một danh sách các vận động viên của đội bóng.
- Tìm và in ra thông tin của các vận động viên ở vị trí Hậu vệ (với đầy đủ thông tin dưới dạng bảng).

CHƯƠNG 9: HÀM

Mục tiêu của chương

Nắm vững:

- Khái niệm, cách khai báo hàm.
- Các tham số trong lời gọi hàm**.**
- Cấp phát và phạm vi hoạt động của các biến.
- Cấu trúc chung của hàm đệ qui.

Nội dung của chương

Nghiên cứu các kiến thức liên quan đến hàm và hàm đệ quy.

9.1. Khai báo hàm

9.1.1. Giới thiệu

Có 2 loại hàm trong ngôn ngữ lập trình C++:

- Hàm thư viện do chương trình dịch cung cấp.
- Hàm tự tạo do người sử dụng định nghĩa.

Hàm thư viện (library functions): chương trình dịch C++ cung cấp một thư viện các hàm tiện ích giúp giảm nhẹ công sức lập trình và cho phép người sử dụng tập trung nhiều hơn vào logic của vấn đề đang giải quyết. Để sử dụng các hàm này trong chương trình, cần chèn vào đầu chương trình các tập tin tiêu đề chứa các khai báo và định nghĩa hằng, biến, hàm nguyên mẫu, bằng lệnh sau: #include <tên tập tin>.

Hàm giúp cho việc phân đoạn chương trình thành những môđun riêng rẽ, hoạt động độc lập với ngữ nghĩa của chương trình lớn, có nghĩa một hàm có thể được sử dụng trong chương trình này mà cũng có thể được sử dụng trong chương trình khác, dễ cho việc kiểm tra và bảo trì chương trình.

Hàm có một số đặc trưng:

• Nằm trong hoặc ngoài văn bản có chương trình gọi đến hàm. Trong một văn bản có thể chứa nhiều hàm,

• Được gọi từ chương trình chính (main), từ hàm khác hoặc từ chính nó (đệ quy),

• Không lồng nhau.

• Có 3 cách truyền giá trị: Truyền theo tham trị, tham biến và tham trỏ.

9.1.2. **K**hai báo hàm

<kiểu_dữ_liệu_trả_lại> <tên_hàm>(danh_sách_tham_số_hình_thức);

Trong đó:

- Kiểu_dữ_liệu là kiểu dữ liệu trả về của hàm.

- Danh_sách_tham_số_hình_thức: là các cặp kiểu dữ liệu, tên tham số cách nhau bởi dấu phẩy.

Ví dụ 9.1a:
```
int nhan2so(int x, int y);
int phepcong(int a, int b);
double binhphuong(double c);
```

9.1.3. Định nghĩa hàm

a. Hàm có trả về giá trị

```
<kiểu_dữ_liệu>  <tên_hàm>(danh_sách_tham_số_hình_thức)
{
  khai báo cục bộ của hàm ;
  dãy lệnh của hàm ;
  return (biểu thức trả về);
}
```

Câu lệnh return có thể nằm ở vị trí bất kỳ trong phần câu lệnh, tuỳ thuộc mục đích của hàm. Khi gặp câu lệnh return chương trình tức khắc thoát khỏi hàm và trả lại giá trị của biểu thức sau return như giá trị của hàm.

Ví dụ 9.1b:

```
int phepnhan(int x, int y)
{
    int z;
    z=x*y;
    return(z);
}
```

Ví dụ 9.1c: Ví dụ sau định nghĩa hàm tính luỹ thừa n (với n nguyên) của một số thực bất kỳ. Hàm này có hai đầu vào (đối thực x và số mũ nguyên n) và đầu ra (giá trị trả lại) kiểu thực với độ chính xác gấp đôi là xn.

```
double luythua(float x, int n)
{
   int i ; // biến chỉ số
   double kq = 1 ; // để lưu kết quả
   for (i=1; i<=n; i++)
       kết quả *= x ;
   return kq;
}
```

b. Hàm không trả về giá trị

Nếu hàm không trả lại giá trị (tức kiểu hàm là void), khi đó có thể có hoặc không có câu lệnh return, nếu có thì đằng sau return sẽ không có biểu thức giá trị trả lại.

Ví dụ 9.1.d: Hàm xoá màn hình 100 lần, hàm chỉ làm công việc cẩn thận xoá màn hình nhiều lần để màn hình thật sạch, nên không có giá trị gì để trả lại.

```
void xmh()
{
    int i;
    for (i=1; i<=100; i++) clrscr();
    return ;
}
```

Hàm main() thông thường có hoặc không có giá trị trả về cho hệ điều hành khi chương trình chạy xong, vì vậy ta thường khai báo kiểu hàm là int main() hoặc void main() và câu lệnh cuối cùng trong hàm thường là return 1 hoặc return.

Trường hợp bỏ qua từ khoá void nhưng trong thân hàm không có câu lệnh return chương trình sẽ ngầm hiểu hàm main() trả lại một giá trị nguyên nhưng vì không có nên khi dịch chương trình ta sẽ gặp lời cảnh báo "Cần có giá trị trả lại cho hàm" (một lời cảnh báo không phải là lỗi, chương trình vẫn chạy bình thường).

Để tránh bị quấy rầy về những lời cảnh báo này chúng ta có thể đặt thêm câu lệnh return 0; (nếu không khai báo void main()) hoặc khai báo kiểu hàm là void main() và đặt câu lệnh return vào cuối hàm.

9.2. Tham số trong lời gọi hàm

Lời gọi hàm được phép xuất hiện trong bất kỳ biểu thức, câu lệnh của hàm khác.

Nếu lời gọi hàm lại nằm trong chính bản thân hàm đó thì ta gọi là đệ quy.

Lời gọi hàm có dạng:

Tên_hàm(danh_sách_tham_số_thực_sự);

Trong đó:

- Tên hàm: đặt tên theo quy tắc đặt tên biến.

- Danh sách tham số thực sự: gồm các giá trị cụ thể để gán lần lượt cho các tham số hình thức của hàm.

Khi hàm được gọi thực hiện thì tất cả những vị trí xuất hiện của tham số hình thức sẽ được gán cho giá trị cụ thể của tham số thực sự tương ứng trong danh sách, sau đó hàm tiến hành thực hiện các câu lệnh của hàm (để tính kết quả).

Các tham số thực sự có thể là các hằng, các biến hoặc biểu thức.

Khi một hàm được gọi, nơi gọi tạm thời chuyển điều khiển đến thực hiện dòng lệnh đầu tiên trong hàm được gọi.

Sau khi kết thúc thực hiện hàm, điều khiển lại được trả về thực hiện tiếp câu lệnh sau lệnh gọi hàm của nơi gọi.

Ví dụ 9.2a: Giả sử ta cần tính giá trị của phép nhân, thay cho việc tính trực tiếp, ta có thể gọi hàm phepnhan() trong ví dụ sau:

```
#include <iostream>
using namespace std;
 int phepnhan(int x, int y)          // x, y là các tham số hình thức.
{
      int z;
      z = a*b;                       // định nghĩa hàm
      return (z);
}
  int main ()
  {
     int r;
     r = phepnhan(7,3);              // 7, 3 là các tham số thật sự.
     cout << "Ket qua la " << r;
     return 0;
  }
```

Ví dụ 9.2b:

```
#include <iostream>
using namespace std;
int phepnhan(int x, int y);                    // x, y là các tham số hình thức.
int main ()
  {
    int r;
    r = phepnhan(7,3);                         // 7,3 là các tham số thật sự.
    cout << "Ket qua la " << r;
    return 0;
  }
```

```
int phepnhan(int x, int y)
{
    int z;                                     // định nghĩa hàm
    z = a*b;
    return (z);
}
```

9.3. Cấp phát và phạm vi hoạt động của các biến

Một biến có thể được gán cho một bí danh mới, và khi đó chỗ nào xuất hiện biến thì cũng tương đương như dùng bí danh và ngược lại.

Một bí danh như vậy được gọi là một biến tham chiếu, ý nghĩa thực tế của nó là cho phép "tham chiếu" tới một biến khác cùng kiểu của nó, tức sử dụng biến khác nhưng bằng tên của biến tham chiếu.

Có các loại biến sau:

- Biến thường với tên thường.
- Biến con trỏ với dấu * trước tên.
- Biến tham chiếu với dấu &.

Cú pháp khai báo

<kiểu biến> &<tên biến tham chiếu> = <tên biến được tham chiếu>;

Ý nghĩa: khai báo trên cho phép người sử dụng tạo ra một biến tham chiếu mới và cho nó tham chiếu đến biến được tham chiếu (cùng kiểu và phải được khai báo từ trước).

Khi đó biến tham chiếu còn được gọi là bí danh của biến được tham chiếu.

Chú ý: không có cú pháp khai báo chỉ tên biến tham chiếu mà không kèm theo khởi tạo.

a. Phạm vi hoạt động của các biến

- Biến toàn cục được khai báo bên ngoài tất cả các hàm.
- Biến cục bộ được khai báo bên trong một hàm hoặc một khối lệnh.
- Sau khi được khai báo, biến toàn cục có thể sử dụng ở bất cứ vị trí nào trong chương trình
- Phạm vi của biến cục bộ chỉ giới hạn trong khối lệnh (trong cặp {}) nơi nó được khai báo.

Ví dụ 9.3: Minh họa biến toàn cục, biến cục bộ.

```
#include <iostream>
using namespace std;
int x =10; int y=5;                    // x, y là biến toàn cục.
int phepcong(int a, int b) {           // a, b là biến cục bộ.
    int r;                             // r là biến cục bộ.
    r = a+b;
    return r;
}
int pheptru(int a, int b) {             // a, b là biến cục bộ.
    int r;                             // r là biến cục bộ.
    r = a - b;
    return r;
}
int main () {
cout << phepcong(x,y) ;
cout << pheptru(x,y) ;
return 0;
}
```

b. Phạm vi và vòng đời của các biến

- Đa số các biến có phạm vi cục bộ.
- Các khai báo hàm thường có phạm vi toàn cục.
- Các hằng thường được khai báo ở phạm vi toàn cục.

Ví dụ 9.4a: Tính diện tích hình tròn.

```
#include <iostream>
using namespace std;
```

```
const double pi=3.14;
int main () {
    double r = 7.0;
    double dientich;
    dientich = pi*r*r;
    cout << dientich;
    return 0;
}
```

Vòng đời là khoảng thời gian thuật toán có thể thực hiện đọc/ghi giá trị cho biến.

Ví dụ 9.4b: Minh họa vòng đời các biến.

```
int x;
cin >> x ; // Nhập giá trị cho x
{
    int y = 2 ;
    if (x == 3)
    {
        int z = 4;
    }
}
```

Kết quả chương trình:

Vòng đời của x: từ dòng 1 đến dòng 9.

Vòng đời của y: từ dòng 4 đến dòng 9.

Vòng đời của z: z chỉ có vòng đời khi x=3.

c. Các cách truyền tham số

- **Truyền theo tham trị**

Truyền theo tham trị: truyền giá trị cho các tham số hình thức.

Trong phương pháp này, chương trình dịch cấp phát vùng nhớ riêng cho từng tham số hình thức, sau đó sao chép giá trị của tham số thực tương ứng vào các tham số hình thức.

Sau khi kết thúc thực hiện hàm, chương trình dịch sẽ thu hồi các vùng nhớ đã cấp phát cho các tham số hình thức, và các biến cục bộ khai báo bên trong hàm.

Như vậy, mọi sự thay đổi trị của các tham số hình thức đều không ảnh hưởng đến các tham số thực bên ngoài hàm.

▪ **Truyền theo tham biến**

Truyền theo tham biến (còn gọi là truyền bằng địa chỉ): truyền nơi lưu trữ trong bộ nhớ của các biến thực sự cho các biến tham chiếu.

Chương trình dịch sẽ truyền địa chỉ của các tham số thực tương ứng cho các tham số hình thức. Hay nói cách khác tham số hình thức là tên gọi khác của tham số thực.

Như vậy, mọi sự thay đổi giá trị của tham số hình thức bên trong hàm chính là thay đổi giá trị của tham số thực bên ngoài hàm.

Để phân biệt cách truyền tham chiếu, với cách truyền tham trị, ta đặt kí hiệu & trước tên biến hình thức.

Ví dụ 9.5a: Đổi chỗ giá trị 2 biến.

```
void doicho(int &x, int &y)
{
    int t = x; x = y; y = t;
}
```

Ví dụ 9.5b: Lời gọi hàm doicho

```
int a = 5, b = 3;
doicho(a, b);
cout << a << b;
```

9.4. Hàm đệ quy

9.4.1. Khái niệm

Đệ quy trong C++ là quá trình trong đó nếu hàm gọi lại chính nó thì ta gọi hàm là đệ qui.

Cú pháp:

```
kiểu_trả_về tên_ham() {
  // code
  ten_ham();
}
```

Ví dụ 9.6: Viết chương trình tính giai thừa sử dụng hàm đệ quy.

```
#include <iostream>
using namespace std;
int giaithua(int n) {
  if (n == 0)
```

```
        return 1;
    else
        return (n * giaithua(n - 1));
}
int main() {
    cout << "n = " ; cin >> n;
    cout << giaithua(n);
    return 0;
}
```

Ví dụ 9.7**:** Tìm UCLN của 2 số a, b. Bài toán có thể được định nghĩa dưới dạng đệ qui như sau:

- Nếu a = b thì UCLN = a
- Nếu a > b thì UCLN(a, b) = UCLN(a-b, b)
- Nếu a < b thì UCLN(a, b) = UCLN(a, b-a)

Từ đó ta có chương trình đệ qui để tính UCLN của a và b như sau.

```
int UCLN(int a, int b) // qui uoc a, b > 0
{
    if (a < b) UCLN(a, b-a);
    if (a == b) return a;
    if (a > b) UCLN(a-b, b);
}
```

Phương pháp đệ qui thường được dùng để giải các bài toán có đặc điểm:

Bài toán giải có các trường hợp riêng gọi là trường hợp suy biến hay cơ sở, trong trường hợp này hàm được tính bình thường mà không cần gọi lại chính nó,

Đối với trường hợp tổng quát, bài toán có thể giải được bằng bài toán cùng dạng nhưng với tham số khác có kích thước nhỏ hơn tham số ban đầu. Và sau một số bước hữu hạn biến đổi cùng dạng, bài toán đưa được về trường hợp suy biến.

9.4.2. Cấu trúc chung của hàm đệ qui

Dạng thức chung của một chương trình đệ qui như sau:

```
if (trường hợp suy biến)
{
        trình bày cách giải          // giả định đã có cách giải
}
else                                 // trường hợp tổng quát
{
        gọi lại hàm với tham số "bé" hơn
}
```

Ví dụ 9.7: Chương trình tính dãy số Fibonacci sử dụng phương pháp đệ quy. (Chỉ số của số Fibonacci tính từ 0, ví dụ: F0 = 0, F1 = 1, F2 = 1, F3 = 2)

```
int fibonacci(int n) {
   if (n < 0) {
      return -1;
   } else if (n == 0 || n == 1) {
      return n;
   } else {
      return fibonacci(n - 1) + fibonacci(n - 2);
   }
}
int main() {
   int i;
   cout << "5 so dau tien cua day so Fibonacci: \n";
      for (i = 0; i < 5; i++) {
      cout << fibonacci(i) << " ";
   }
}
```

Câu hỏi ôn tập

1. Trình bày cách khai báo hàm, định nghĩa hàm, cách gọi hàm.

2. Trình bày các phương pháp truyền tham số cho hàm (truyền theo tham trị, truyền theo tham biến).

3. Trình bày khái niệm và **c**ấu trúc chung của hàm đệ qui.

Bài tập vận dụng

1. Viết hàm trả về phần nguyên của phép chia hai số nguyên.

2. Sử dụng hàm đệ quy viết chương trình nhập n từ bàn phím (n nguyên, dương), tính tổng S(n) = 1 + 2 +…+ n.

3. Viết hàm đệ quy tính tổng chẵn S(n) = 2 + 4 +…+ 2n với (n nguyên, dương). Viết chương trình cho phép nhập n từ bàn phím sau đó gọi hàm tính tổng chẵn để xuất kết quả ra màn hình.

4. Viết hàm đệ quy tính tích S(n) = 1 x 2 x…x n, với (n nguyên, dương). Viết chương trình cho phép nhập n từ bàn phím sau đó gọi hàm tính tích để xuất kết quả ra màn hình.

5. Cho trước số tự nhiên n. Viết hàm kiểm tra số n có phải là số đối xứng hay không.

TÀI LIỆU THAM KHẢO

[1]. Bùi Thế Tâm, Giáo trình tin học Đại cương, nhà xuất bản Giao thông vận tải năm 2005.

[2]. Phạm Hồng Thái, "Bài giảng ngôn ngữ lập trình C++", Đại học quốc gia Hà nội, năm 2003.

[3]. PGS.TS. Trần Đình Quế và KS. Nguyễn Mạnh Hùng, Ngôn ngữ lập trình C++, Học viện bưu chính viễn thông, năm 2006.

[4]. Joel Adams & Larry Nyhoff, "C++ An Introduction to Computing", Prentice Hall 2002, Third Edition.

[5]. By H. M. Deitel, P. J. Deitel, "C++ How to Program"- Fourth Edition, Prentice Hall, New Jersey, 2003, ISBN: 0-13-038474.

[6]. David Vandevoorde, Nicolai M. Josuttis and Douglas Gregor, "C++ Templates: The Complete Guide 2nd Edition", Addison-Wesley Professional; 2 edition, September 18, 2017.

[7]. Published by Microsoft Press, "Introducing Windows 10 for IT Professionals Preview Edition", Copyright 2015 © Microsoft Corporation .

[8]. Published by John Wiley & Sons, Inc, "Windows 10 for dummier", 111 River Street, Hoboken, NJ 07030- 5774, www.wiley.com.

[9]. Joel Adams & Larry Nyhoff, "C++ An Introduction to Computing", Prentice Hall 2002, Third Edition.

PHỤ LỤC 1

Phần phụ lục này sẽ giới thiệu cho người sử dụng về cách cài đặt, khởi động và sử dụng Turbo C++ IDE để soạn thảo, biên dịch, bắt lỗi, và thực thi các chương trình C++.

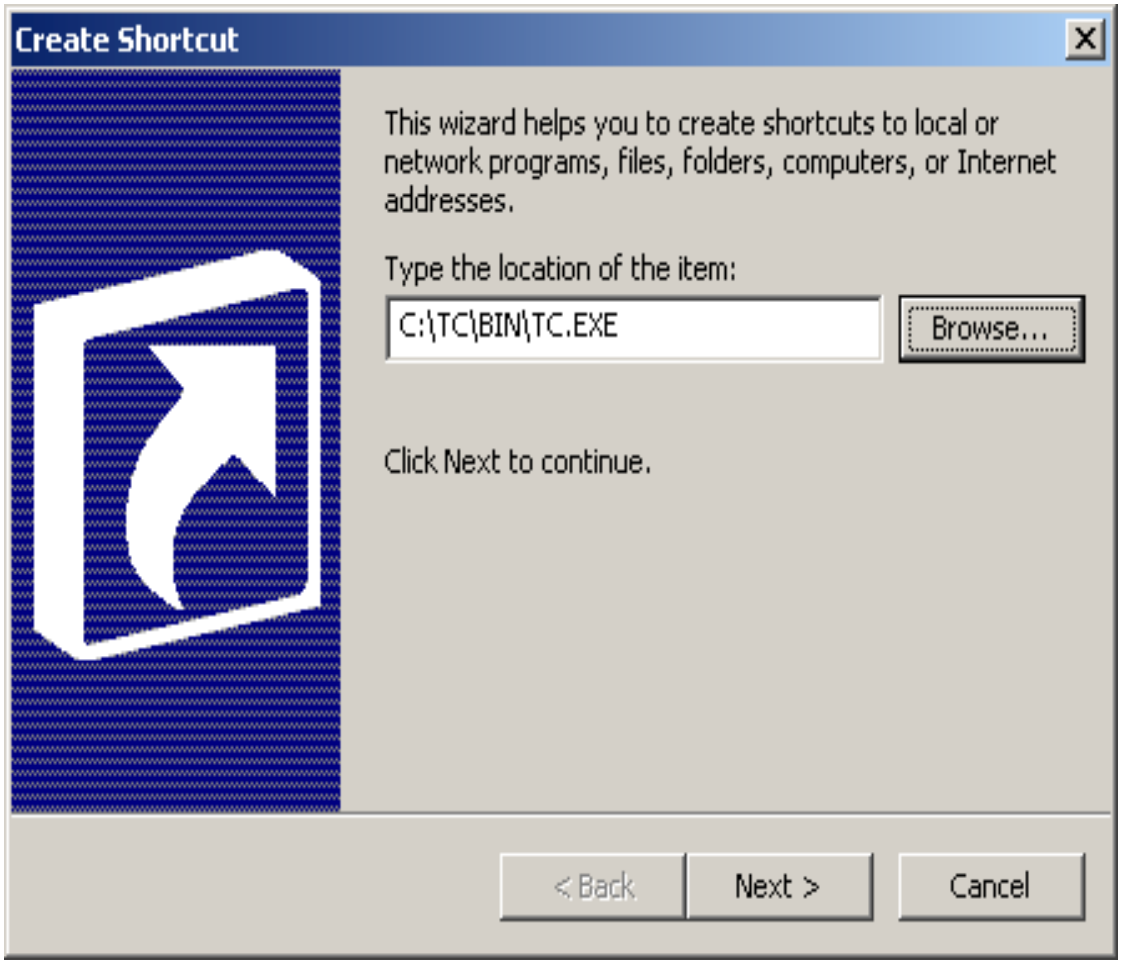
Cài đặt và khởi động Turbo C

Người sử dụng phải sử dụng trình INSTALL để cài đặt Turbo C++. Tất cả các tập tin cài đặt sẽ được giải nén và chép tới hệ thống máy tính của người sử dụng một cách thích hợp. người sử dụng không thể thực hiện việc này thủ công được.

Để bắt đầu cài đặt, chuyển đến thư mục chứa bộ nguồn cài đặt và kích hoạt trình INSTALL để tiến hành cài đặt. Trình INSTALL cài cả hai trình biên dịch và các công cụ vào hệ thống của bạn. Thư mục mặc định của Turbo C++ là C:\TC.

Để khởi động chương trình Turbo C++ người sử dụng chuyển vào thư mục C:\TC\BIN và kích hoạt TC.exe.

Tuy nhiên để có thể khởi động nhanh ta có thể tạo một shortcut đến C:\TC\Bin\TC.exe như hình A.1 sau.

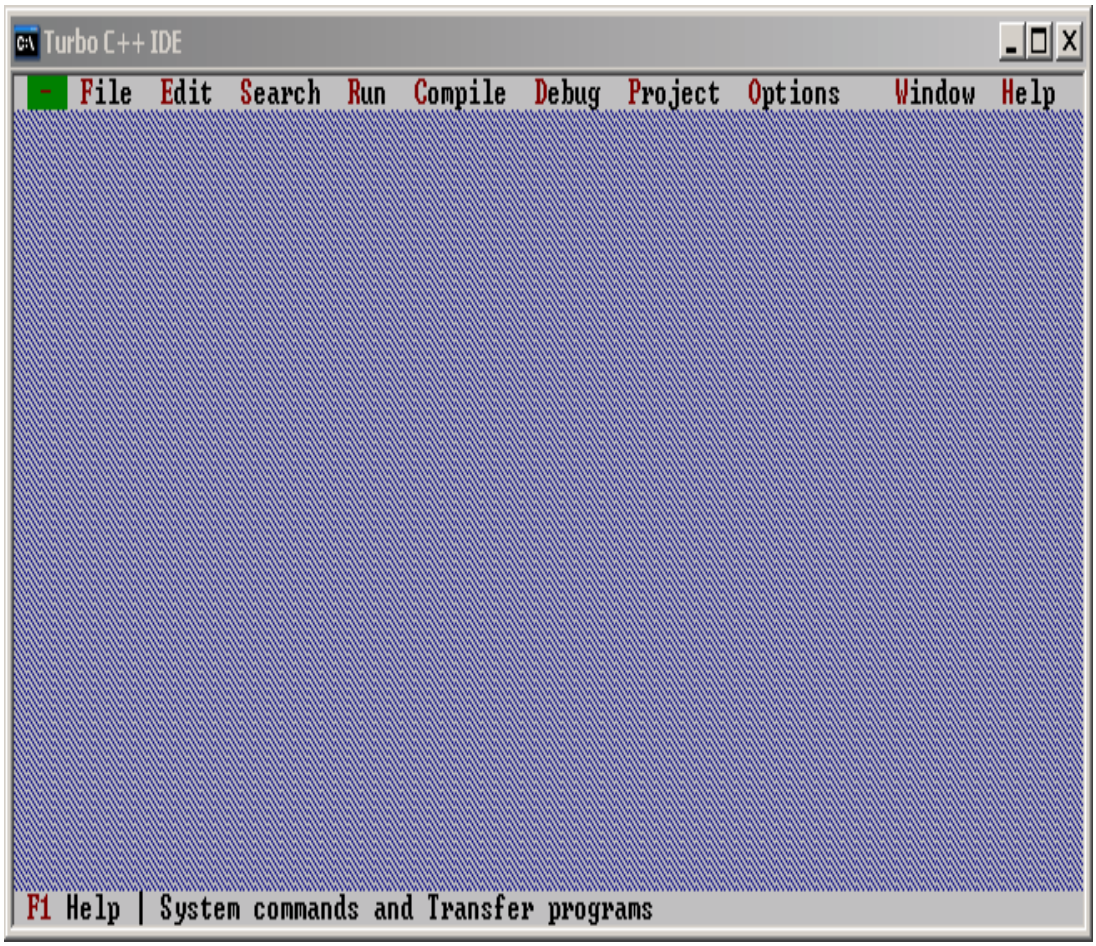


Turbo C++ IDE

Turbo C++ IDE cung cấp mọi thứ mà người sử dụng cần để viết, soạn thảo, biên dịch, quản lý, chạy, liên kết, và bắt lỗi chương trình.

Thanh trình đơn ngang ở đỉnh của màn hình cho phép người sử dụng thực hiện hầu hết các chức năng đã nêu. Người sử dụng có thể kích hoạt thanh trình đơn này bằng một trong hai cách:

- Nhấn phím F10, hoặc
- Kích chuột vào một vị trí bất kỳ trên thanh trình đơn này



Thanh trình đơn ngang chứa lần lượt từ trái qua phải các trình đơn con sau:

File, Edit, Search, Run, Compile, Debug, Project, Options, Window, và Help.

Để chọn một trình đơn này người sử dụng có thể thực hiện một trong hai cách sau:

Cách 1:

- - Kích hoạt thanh trình đơn ngang
- - Sử dụng phím mũi tên trái và phải để di chuyển khung sáng màu xanh đến trình đơn cần chọn

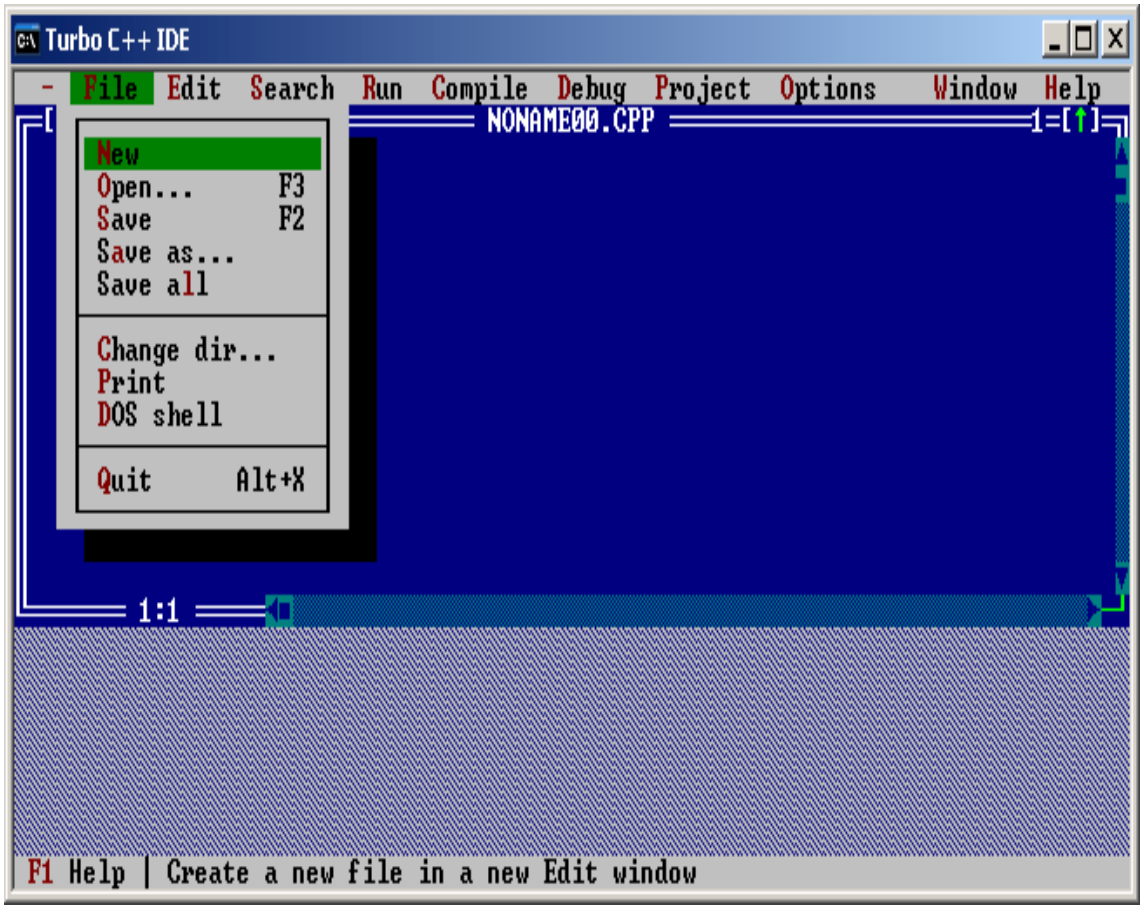
Cách 2:

□ - Sử dụng phím nóng<Alt> + <ký tự đầu tiên màu đỏ của tên trình đơn>.

Trình đơn File

Trình đơn này có thể được chọn thông qua tổ hợp phím <Alt>+<F>. Trình đơn này cung cấp những lệnh để tạo ra:

- Những tập tin mới (File > New)
- Mở những tập tin có sẵn (File > Open)□
- Lưu các tập tin đã soạn thảo (File > Save / Save As / Save All)□
- Chuyển thư mục (File > Change dir)□
- In các tập tin (File > Print)□
- Thoát tạm về DOS (File > DOS shell)□
- Thoát khỏi Turbo C++ (File > Quit)



Khi người sử dụng chọn **File**>**New,** một cửa sổ soạn thảo mới với tên mặc định là NONAMExx.CPP (xx là thay cho số từ 00 đến 31).

Cửa sổ này tự động kích hoạt để cho phép người sử dụng soạn thảo mã chương trình. Các tập tin NONAME được sử dụng như vùng soạn thảo tạm thời.

Turbo C++ sẽ nhắc người sử dụng đặt tên cho tập tin khi bạn lưu tập tin đó.

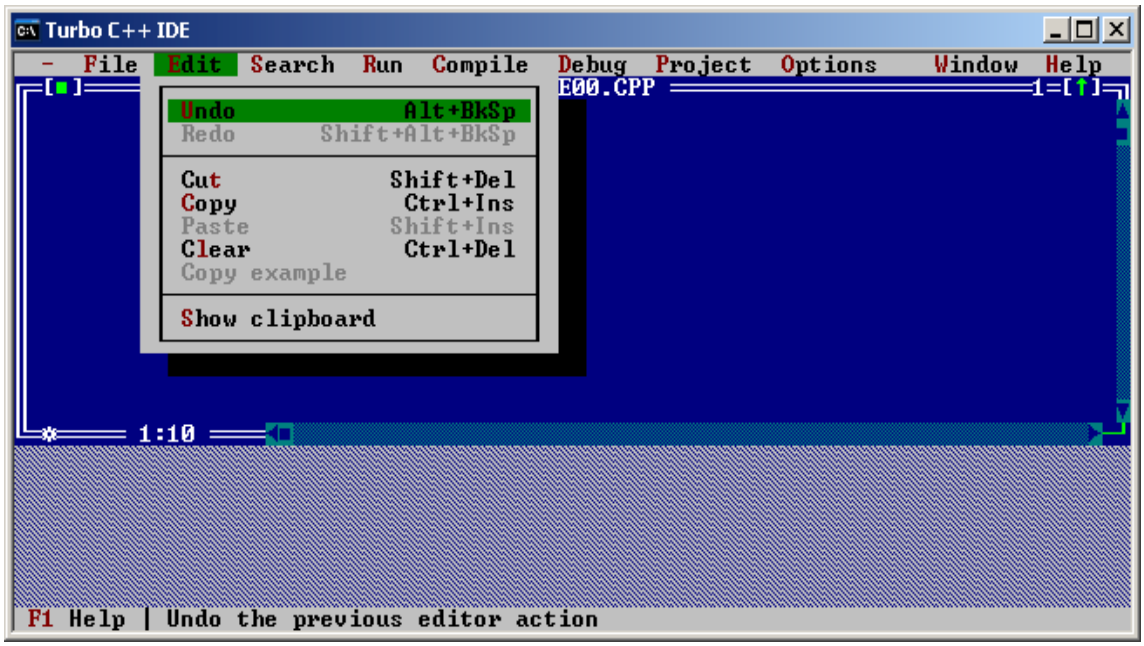
Trình đơn Edit

Trình đơn này có thể được chọn thông qua tổ hợp phím <Alt>+<E>.

Trình đơn này cung cấp các lệnh để cắt, sao chép, và dán văn bản trong cửa sổ soạn thảo.

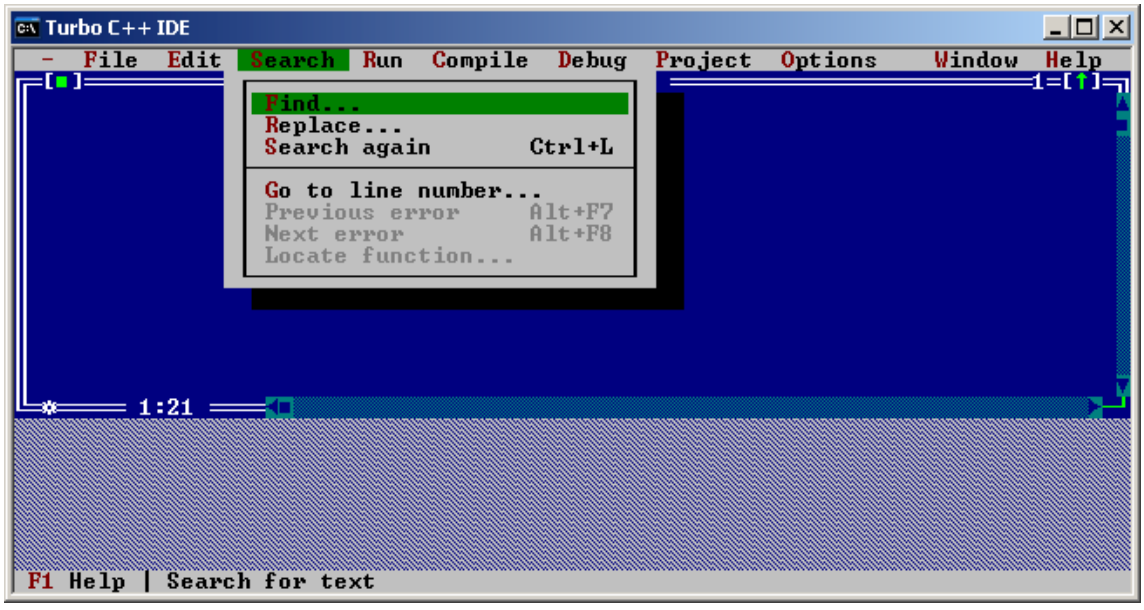
Người sử dụng cũng có thể hủy bỏ các chuyển đổi và trở lại những chuyển đổi mà người sử dụng đã hủy bỏ.

Người sử dụng có thể mở một cửa sổ lưu trữ tạm thời để xem hay soạn thảo nội dung của nó, và chép văn bản từ các cửa sổ thông điệp, xuất hay trợ giúp.



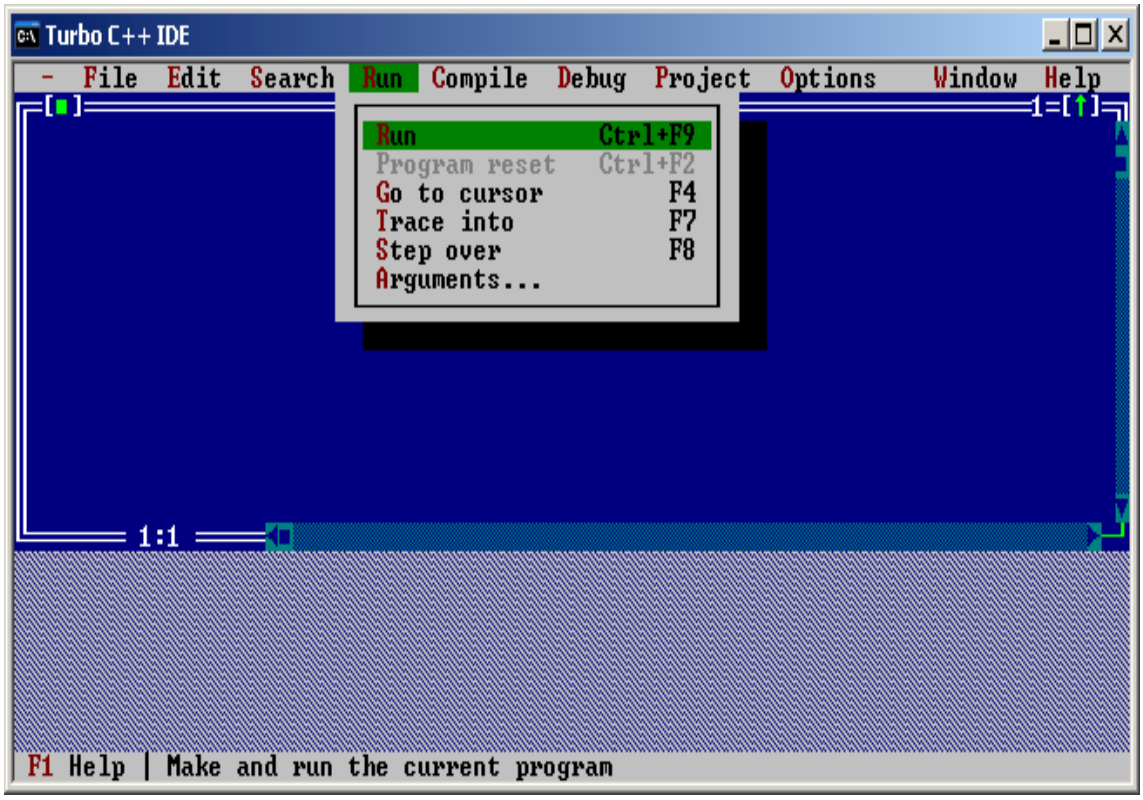
Trình đơn Search

Trình đơn này cung cấp những lệnh để tìm kiếm văn bản, khai báo hàm, và định vị lỗi trong các tập tin chương trình.



Trình đơn Run

Trình đơn này cung cấp các lệnh để thực thi chương trình, bắt đầu và kết thúc các phần bắt lỗi được minh họa



Người sử dụng chọn **Run** > **Run** hoặc nhấn tổ hợp phím **Ctrl + F9** để thực thi chương trình bằng cách sử dụng bất kỳ đối số nào mà người sử dụng đã truyền cho nó trong lệnh Arguments.

Nếu mã nguồn đã được sửa đổi từ lần biên dịch cuối cùng thì lệnh Run cũng triệu gọi Project Manager để biên dịch lại và liên kết với chương trình của người sử dụng.

Người sử dụng cũng có thể chọn **Run** > **Go to cursor** hoặc nhấn phím **F4** để cho chương trình chạy đến hàng mà con nháy đang định vị trong cửa sổ soạn thảo hiện tại. Nếu con nháy đang nằm ở hàng mà không chứa một lệnh có thể thực thi thì Turbo C++ sẽ hiển thị một cảnh báo.

Để chạy chương trình của người sử dụng ở chế độ từng lệnh một thì có thể chọn **Run** > **Trace into** hoặc nhấn phím **F7.**

Thanh sáng sẽ xuất hiện tại lệnh đang được thực hiện để giúp người sử dụng kiểm soát được quá trình thực thi của chương trình.

Người sử dụng có thể sử dụng lệnh này lặp đi lặp lại nhiều lần đến khi lệnh cuối cùng trong chương trình được thực hiện thành công.

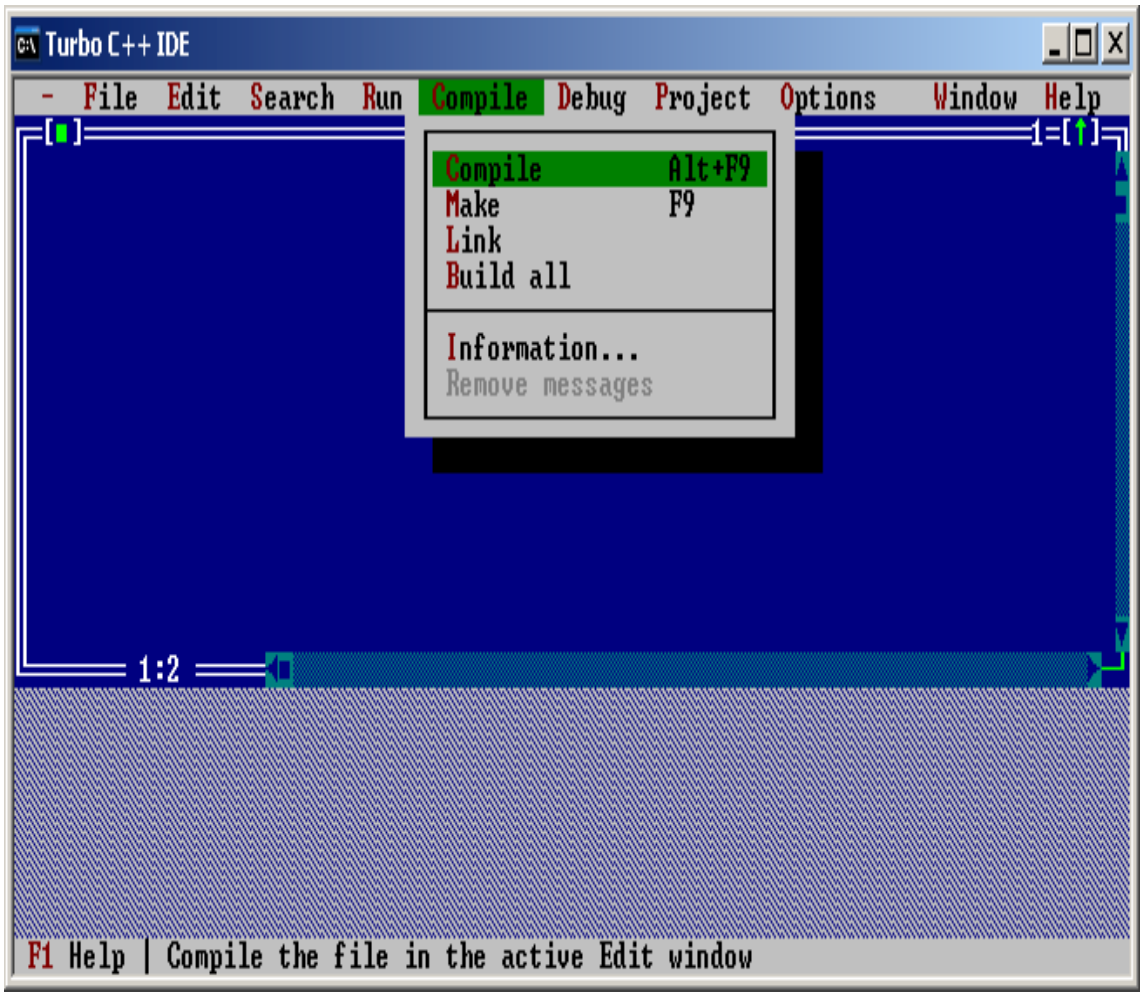
Lệnh **Run** > **Step over** tương đương với nhấn phím **F8** sẽ thực thi lệnh kế tiếp trong hàm hiện tại. Nó không dò theo những lời gọi hàm mức thấp hơn nên hữu dụng trong trường hợp người sử dụng muốn chạy hàm đang bắt lỗi mà không muốn rẽ sang những hàm khác.

Trình đơn Complie

Người sử dụng sử dụng những lệnh trên trình đơn này để biên dịch chương trình trong cửa sổ soạn thảo đang hoạt động, hoặc tạo hoặc xây dựng dự án của mình.

Để sử dụng các lệnh Compile, Make, Build, và Link thì phải có một tập tin đang mở trong cửa sổ soạn thảo đang hoạt động.

Bên cạnh đó các lệnh Make, Build, Link đòi hỏi cần phải có một dự án đã được định nghĩa.

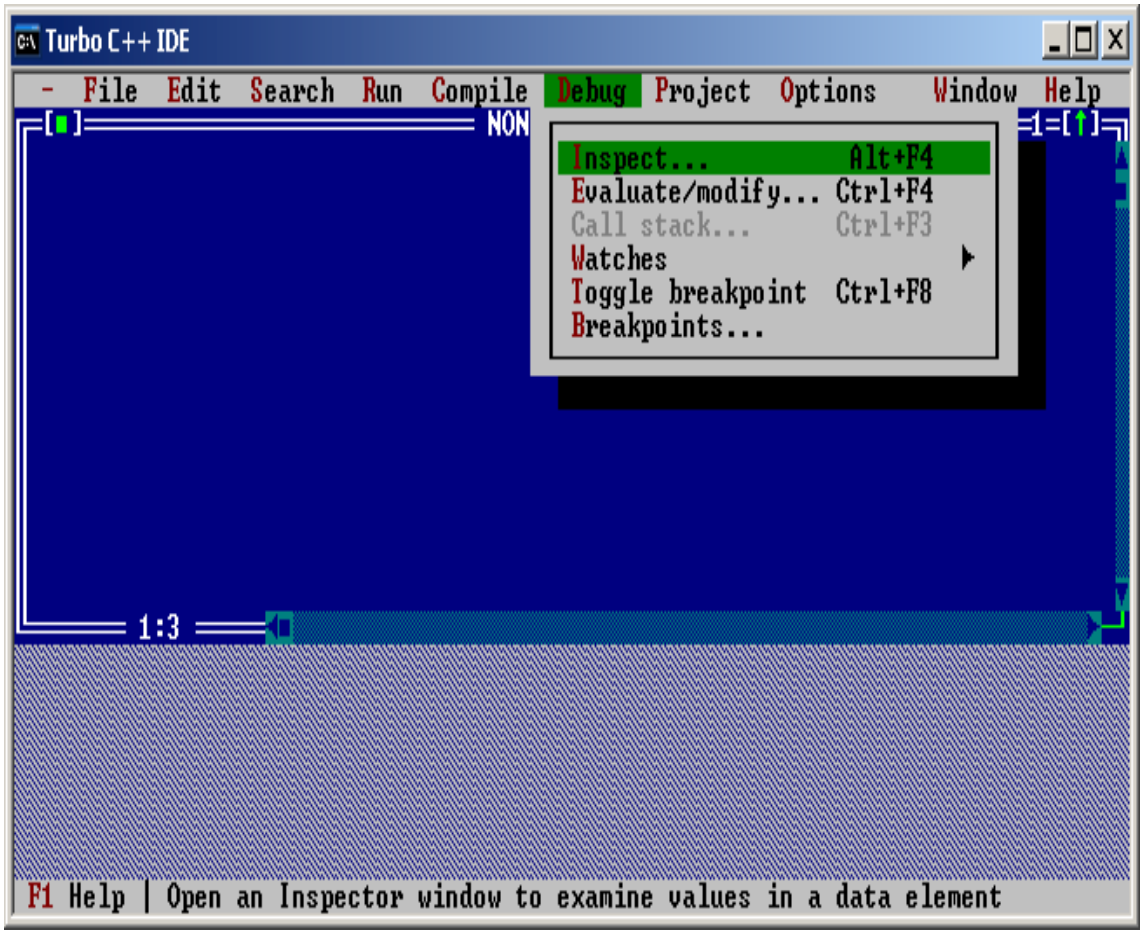


Lệnh **Run** > **Compile** tương đương với nhấn tổ hợp phím **Alt + F9** được sử dụng để biên dịch một tập tin .C hoặc .CPP sang một tập tin .OBJ.

Khi Turbo C++ đang biên dịch thì một hộp trạng thái bật lên để hiển thị kết quả của việc biên dịch gồm có: số hàng đã biên dịch, số lỗi và cảnh báo, và bộ nhớ sẵn dùng. Khi quá trình biên dịch kết thúc hãy nhấn một phím bất kỳ để loại bỏ hộp trạng thái này. Nếu có bất kỳ lỗi nào xảy ra thì cửa sổ Message được kích hoạt để hiển thị và tô sáng lỗi đầu tiên.

Trình đơn Debug

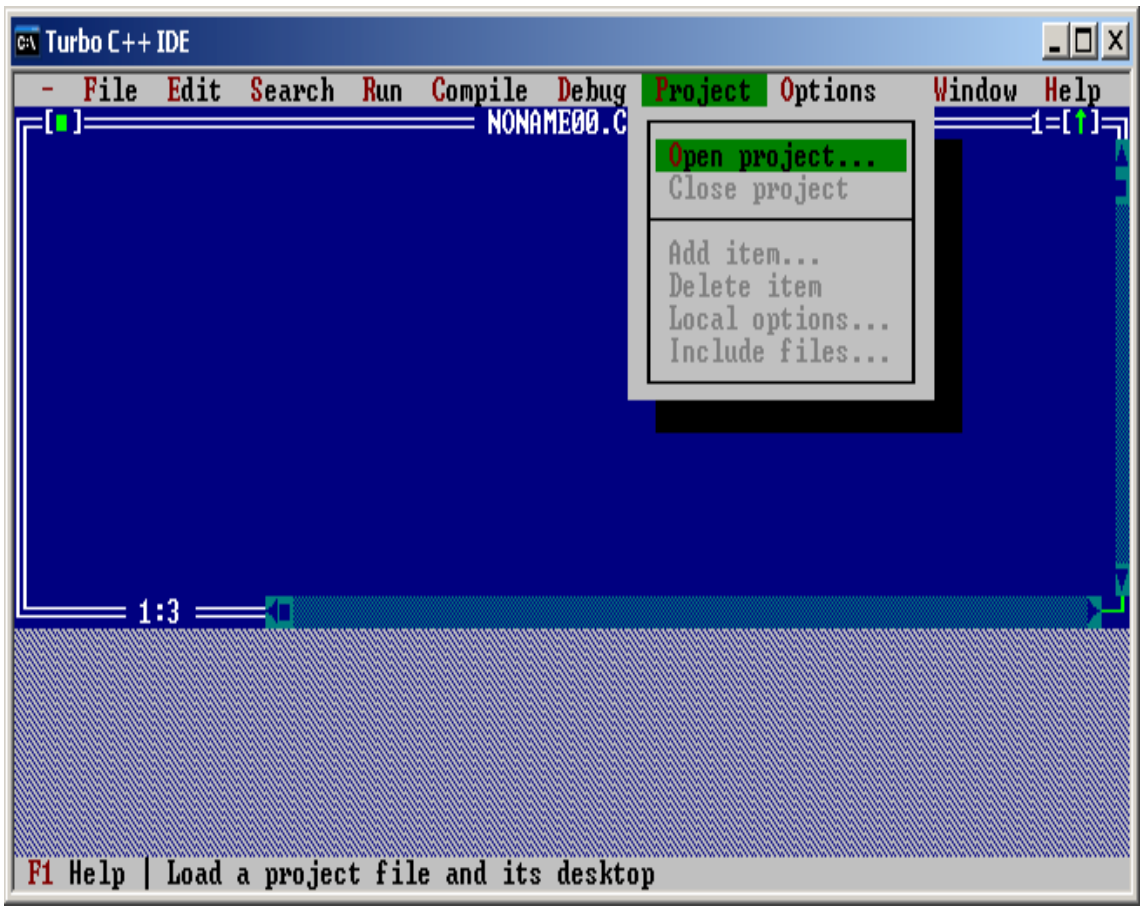
Các lệnh trên trình đơn này điều khiển tất cả các đặc tính của trình bắt lỗi tích hợp. Người sử dụng có thể chỉ định thông tin bắt lỗi nào cần được phát ra, thông tin nào không cần được phát ra trong hộp thoại thông báo lỗi.



Trình đơn Project

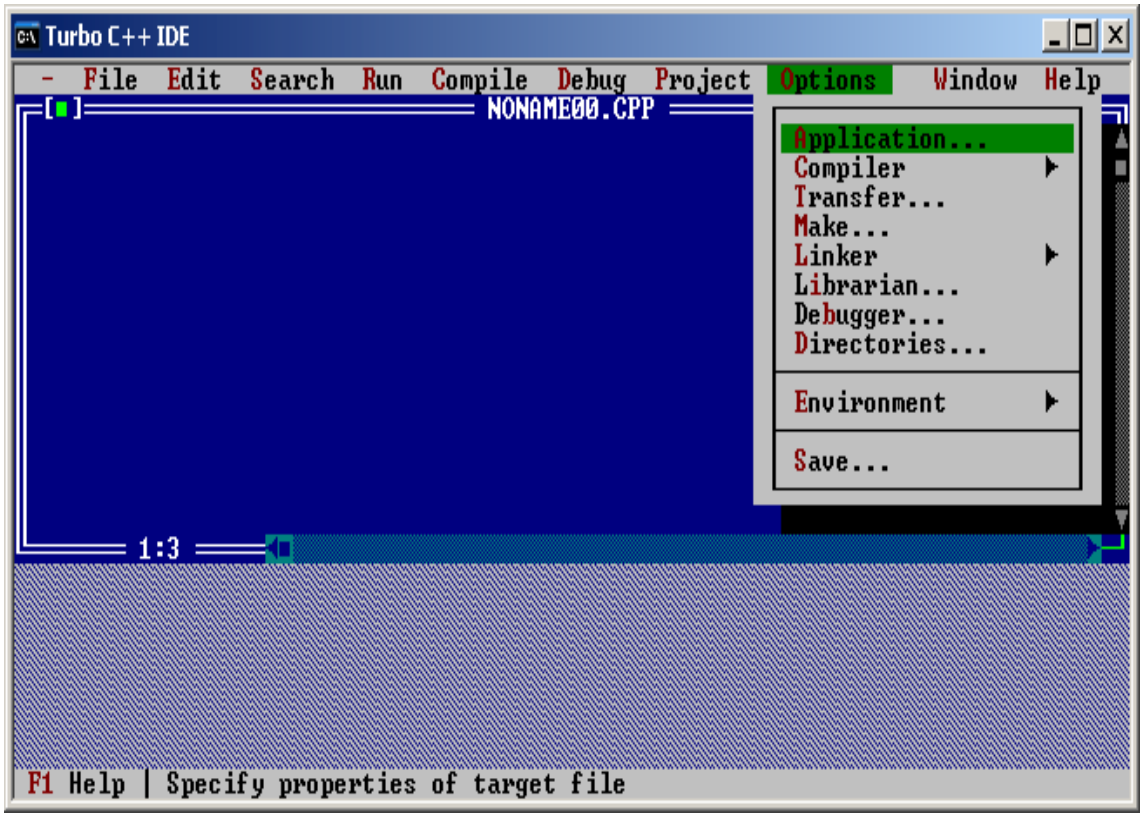
Trình đơn này chứa đựng tất cả các lệnh quản lý dự án để thực hiện các công việc:

- Tạo ra và mở một dự án
- Thêm và xóa các tập tin trong dự án
- Thiết lập các tùy chọn cho một tập tin trong dự án
- Chỉ định các tập tin nguồn cần được dịch
- Xem các tập tin được chèn vào trong dự án



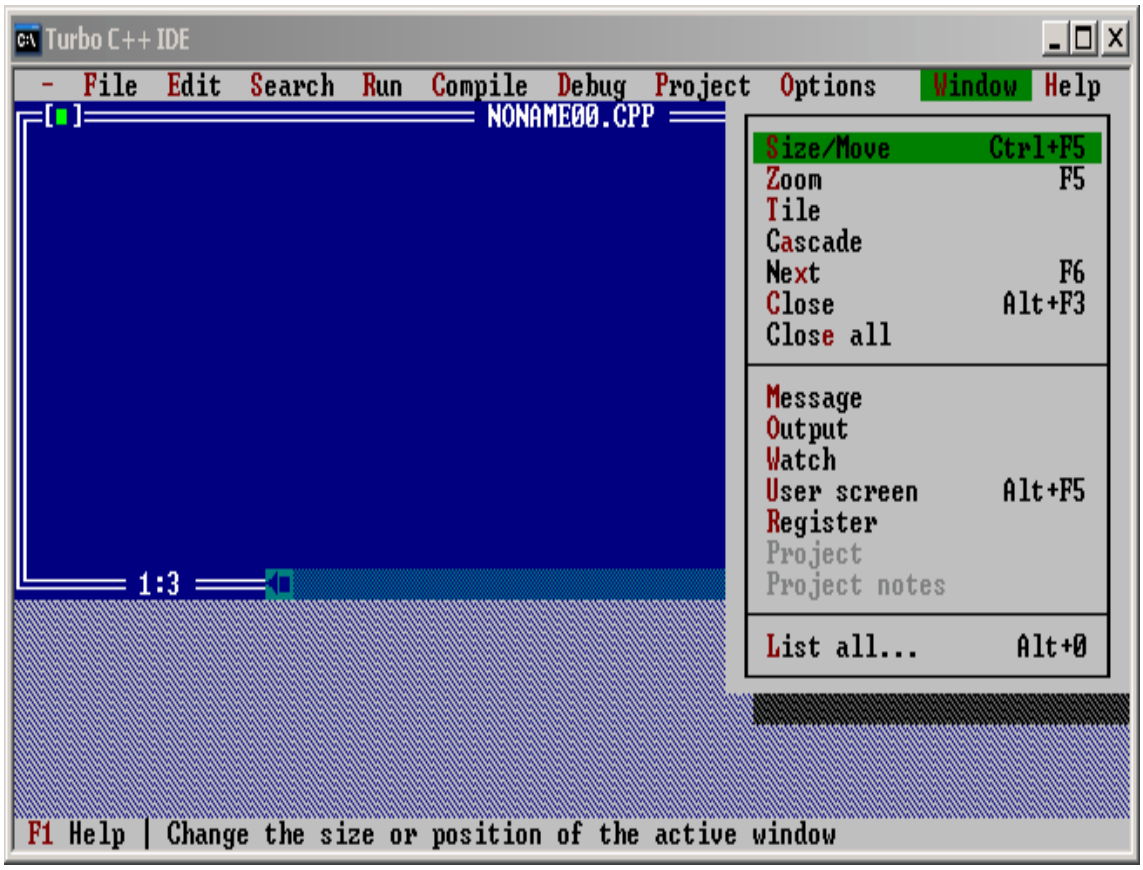
Trình đơn Options

Trình đơn này chứa đựng các lệnh cho phép người sử dụng xem và chuyển đổi các thiết lập mặc định trong Turbo C++.



Trình đơn Window

Trình đơn này chứa các lệnh để quản lý cửa sổ. Phần lớn các cửa sổ người sử dụng mở ra từ trình đơn này có tất cả các phần tử cơ bản của cửa sổ như thanh cuộn, một hộp đóng, và các biểu tượng phóng đại.



Trình đơn Help

Trình đơn này cung cấp các chức năng trợ giúp hỗ trợ cho lập trình viên. Hệ thống trợ giúp cung cấp thông tin cho tất cả các khía cạnh của IDE và Turbo C++.