Chương 1. Mở đầu

Chương này giới thiệu những phần cơ bản của một chương trình C++. Chúng ta sử dụng những ví dụ đơn giản để trình bày cấu trúc các chương trình C++ và cách thức biên dịch chúng. Các khái niệm cơ bản như là hằng, biến, và việc lưu trữ chúng trong bộ nhớ cũng sẽ được thảo luận trong chương này. Sau đây là một đặc tả sơ bộ về khái niệm lập trình.

Lập trình

Máy tính số là một công cụ để giải quyết hàng loạt các **bài toán** lớn. Một lời giải cho một bài toán nào đó được gọi là một **giải thuật** (algorithm); nó mô tả một chuỗi các bước cần thực hiện để giải quyết bài toán. Một ví dụ đơn giản cho một bài toán và một giải thuật có thể là:

> **Bài toán**: Sắp xếp một danh sách các số theo thứ tự tăng dần.
> **Giải thuật**:Giả sử danh sách đã cho là *list1*; tạo ra một danh sách rỗng, *list2*, để lưu danh sách đã sắp xếp. Lặp đi lặp lại công việc, tìm số nhỏ nhất trong *list1,* xóa nó khỏi *list1,* và thêm vào phần tử kế tiếp trong danh sách *list2,* cho đến khi *list1* là rỗng.

Giải thuật được diễn giải bằng các thuật ngữ trừu tượng mang tính chất dễ hiểu. Ngôn ngữ thật sự được hiểu bởi máy tính là ngôn ngữ máy. Chương trình được diễn đạt bằng ngôn ngữ máy được gọi là **có thể thực thi**. Một chương trình được viết bằng bất kỳ một ngôn ngữ nào khác thì trước hết cần được dịch sang ngôn ngữ máy để máy tính có thể hiểu và thực thi nó.

Ngôn ngữ máy cực kỳ khó hiểu đối với lập trình viên vì thế họ không thể sử dụng trực tiếp ngôn ngữ máy để viết chương trình. Một sự trừu tượng khác là **ngôn ngữ assembly.** Nó cung cấp những tên dễ nhớ cho các lệnh và một ký hiệu dễ hiểu hơn cho dữ liệu. Bộ dịch được gọi là **assembler** chuyển ngôn ngữ assembly sang ngôn ngữ máy.

Ngay cả những ngôn ngữ assembly cũng khó sử dụng. Những ngôn ngữ cấp cao như C++ cung cấp các ký hiệu thuận tiện hơn nhiều cho việc thi hành các giải thuật. Chúng giúp cho các lập trình viên không phải nghĩ nhiều về các thuật ngữ cấp thấp, và giúp họ chỉ tập trung vào giải thuật. **Trình biên dịch** (compiler) sẽ đảm nhiệm việc dịch chương trình viết bằng ngôn ngữ cấp cao sang ngôn ngữ assembly. Mã assembly được tạo ra bởi trình biên dịch sau đó sẽ được tập hợp lại để cho ra một chương trình có thể thực thi.

1.1. Một chương trình C++ đơn giản

Danh sách 1.1 trình bày chương trình C++ đầu tiên. Chương trình này khi chạy sẽ xuất ra thông điệp Hello World.

Danh sách 1.1

```
1  #include <iostream.h>

2  int main (void)
3  {
4        cout << "Hello World\n";
5  }
```

Chú giải

1 Hàng này sử dụng chỉ thị tiền xử lý #include để chèn vào nội dung của tập tin header iostream.h trong chương trình. iostream.h là tập tin header chuẩn của C++ và chứa đựng các định nghĩa cho xuất và nhập.

2 Hàng này định nghĩa một hàm được gọi là main. Hàm có thể không có hay có nhiều **tham số** (parameters); các tham số này luôn xuất hiện sau tên hàm, giữa một cặp dấu ngoặc. Việc xuất hiện của từ void ở giữa dấu ngoặc chỉ định rằng hàm main không có tham số. Hàm có thể có **kiểu trả về**; kiểu trả về luôn xuất hiện trước tên hàm. Kiểu trả về cho hàm main là int (ví dụ: một số nguyên). Tất cả các chương trình C++ phải có một hàm main duy nhất.Việc thực thi chương trình luôn bắt đầu từ hàm main.

3 Dấu ngoặc nhọn bắt đầu thân của hàm main.

4 Hàng này là một **câu lệnh** (statement). Một lệnh là một sự tính toán để cho ra một giá trị. Kết thúc một lệnh thì luôn luôn được đánh dấu bằng dấu chấm phẩy (;). Câu lệnh này xuất ra **chuỗi** "Hello World\n" để gởi đến dòng xuất cout. Chuỗi là một dãy các ký tự được đặt trong cặp nháy kép. Ký tự cuối cùng trong chuỗi này (\n) là một ký tự xuống hàng (newline). Dòng là một đối tượng được dùng để thực hiện các xuất hoặc nhập. cout là dòng xuất chuẩn trong C++ (xuất chuẩn thường được hiểu là màn hình máy tính). Ký tự << là toán tử xuất, nó xem dòng xuất như là toán hạng trái và xem biểu thức như là toán hạng phải, và tạo nên giá trị của biểu thức được gởi đến dòng xuất. Trong trường hợp này, kết quả là chuỗi "Hello World\n" được gởi đến dòng cout, làm cho nó được hiển thị trên màn hình máy tính.

5 Dấu ngoặc đóng kết thúc thân hàm main.

1.2. Biên dịch một chương trình C++

Bảng 1.1 trình bày chương trình trong danh sách 1.1 được biên dịch và chạy trong môi trường UNIX thông thường. Phần in đậm được xem như là đầu vào (input) của người dùng và phần in thường được xem như là đáp ứng của hệ thống. Dấu nhắc ở hàng lệnh UNIX xuất hiện như là ký tự dollar($).

Bảng 1.1

```
1  $ CC hello.cc
2  $ a.out
3  Hello World
4  $
```

Chú giải

1 Lệnh để triệu gọi bộ dịch AT&T của C++ trong môi trường UNIX là CC. Đối số cho lệnh này (hello.cc) là tên của tập tin chứa đựng chương trình. Theo qui định thì tên tập tin có phần mở rộng là .c, .C, hoặc là .cc. (Phần mở rộng này có thể là khác nhau đối với những hệ điều hành khác nhau)

2 Kết quả của sự biên dịch là một tập tin có thể thực thi mặc định là a.out. Để chạy chương trình, chúng ta sử dụng a.out như là lệnh.

3 Đây là kết quả được cung cấp bởi chương trình.

4 Dấu nhắc trở về hệ thống chỉ định rằng chương trình đã hoàn tất sự thực thi của nó.

Lệnh cc chấp nhận các phần tùy chọn. Mỗi tùy chọn xuất hiện như name, trong đó name là tên của tùy chọn (thường là một ký tự đơn). Một vài tùy chọn yêu cầu có đối số. Ví dụ tùy chọn xuất (-o) cho phép chỉ định rõ tập tin có thể được cung cấp bởi trình biên dịch thay vì là a.out. Bảng 1.2 minh họa việc sử dụng tùy chọn này bằng cách chỉ định rõ hello như là tên của tập tin có thể thực thi.

Bảng 1.2

```
1  $ CC hello.cc -o hello
2  $ hello
3  Hello World
4  $
```

Mặc dù lệnh thực sự có thể khác phụ thuộc vào trình biên dịch, một thủ tục biên dịch tương tự có thể được dùng dưới môi trường MS-DOS. Trình biên dịch C++ dựa trên Windows dâng tặng một môi trường thân thiện với người dùng mà việc biên dịch rất đơn giản bằng cách chọn lệnh từ menu. Qui định tên dưới MS-DOS và Windows là tên của tập tin nguồn C++ phải có phần mở rộng là .cpp.

1.3. Việc biên dịch C++ diễn ra như thế nào

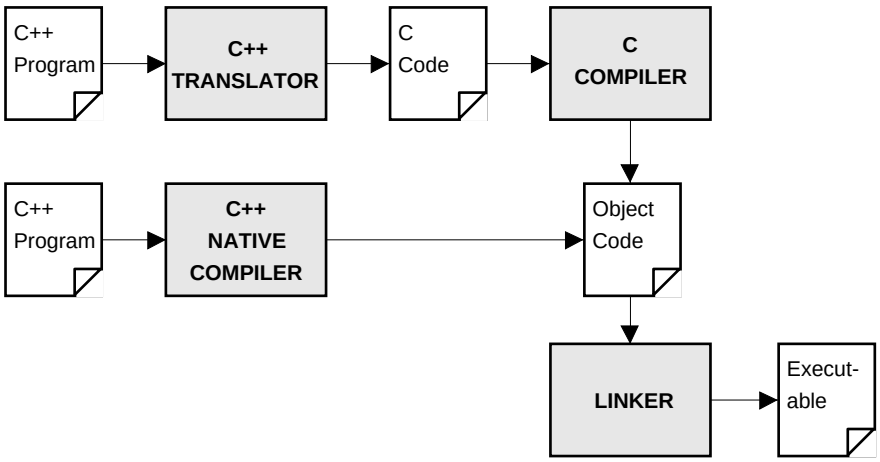
Biên dịch một chương trình C++ liên quan đến một số bước (hầu hết các bước là trong suốt với người dùng):

- Đầu tiên, **bộ tiền xử lý** C++ xem qua mã trong chương trình và thực hiện các chỉ thị được chỉ định bởi các chỉ thị tiền xử lý (ví dụ, #include). Kết quả là một mã chương trình đã sửa đổi mà không còn chứa bất kỳ một chỉ thị tiền xử lý nào cả.

- Sau đó, **trình biên dịch** C++ dịch các mã của chương trình. Trình biên dịch có thể là một trình biên dịch C++ thật sự phát ra mã assembly hay mã máy, hoặc chỉ là trình chuyển đổi dịch mã sang C. Ở trường hợp thứ hai, mã C sau khi được dịch ra sẽ tạo thành mã assembly hay mã máy thông qua trình biên dịch C. Trong cả hai trường hợp, đầu ra có thể không hoàn chỉnh vì chương trình tham khảo tới các thủ tục trong thư viện còn chưa được định nghĩa như một phần của chương trình. Ví dụ Danh sách 1.1 tham chiếu tới toán tử << mà thực sự được định nghĩa trong một thư viện IO riêng biệt.
- Cuối cùng, **trình liên kết** hoàn tất mã đối tượng bằng cách liên kết nó với mã đối tượng của bất kỳ các module thư viện mà chương trình đã tham khảo tới. Kết quả cuối cùng là một tập tin thực thi.

Hình 1.1 minh họa các bước trên cho cả hai trình chuyển đổi C++ và trình biên dịch C++. Thực tế thì tất cả các bước trên được triệu gọi bởi một lệnh đơn (như là CC) và người dùng thậm chí sẽ không thấy các tập tin được phát ra ngay lập tức.

Hình 1.1 Việc biên dịch C++



1.4. Biến

Biến là một tên tượng trưng cho một vùng nhớ mà dữ liệu có thể được lưu trữ trên đó hay là được sử dụng lại. Các biến được sử dụng để giữ các giá trị dữ liệu vì thế mà chúng có thể được dùng trong nhiều tính toán khác nhau trong một chương trình. Tất cả các biến có hai thuộc tính quan trọng:

- **Kiểu** được thiết lập khi các biến được định nghĩa (ví dụ như: integer, real, character). Một khi đã được định nghĩa, kiểu của một biến C++ không thể được chuyển đổi.

- **Giá trị** có thể được chuyển đổi bằng cách gán một giá trị mới cho biến. Loại giá trị của biến có thể nhận phụ thuộc vào kiểu của nó. Ví dụ, một biến số nguyên chỉ có thể giữ các giá trị nguyên (chẳng hạn, 2, 100, -12).

Danh sách 1.2 minh họa sử dụng một vài biến đơn giản.

Danh sách 1.2

```
1   #include <iostream.h>

2   int main (void)
3   {
4       int         workDays;
5       float  workHours, payRate, weeklyPay;

6       workDays = 5;
7       workHours = 7.5;
8       payRate = 38.55;
9       weeklyPay = workDays  * workHours * payRate;
10      cout << "Weekly Pay = "<< weeklyPay<< '\n';
11  }
```

Chú giải

4 Hàng này định nghĩa một biến int (kiểu số nguyên) tên là workDays, biến này đại diện cho số ngày làm việc trong tuần. Theo như luật chung, trước tiên một biến được định nghĩa bằng cách chỉ định kiểu của nó, theo sau đó là tên biến và cuối cùng là được kết thúc bởi dấu chấm phẩy.

5 Hàng này định nghĩa ba biến float (kiểu số thực) lần lượt thay cho số giờ làm việc trong ngày, số tiền phải trả hàng giờ, và số tiền phải trả hàng tuần. Như chúng ta thấy ở hàng này, nhiều biến của cùng kiểu có thể định nghĩa một lượt qua việc dùng dấu phẩy để ngăn cách chúng.

6 Hàng này là một câu lệnh gán. Nó gán giá trị 5 cho biến workDays. Vì thế, sau khi câu lệnh này được thực thi, workDays biểu thị giá trị 5.

7 Hàng này gán giá trị 7.5 tới biến workHours.

8 Hàng này gán giá trị 38.55 tới biến payRate.

9 Hàng này tính toán số tiền phải trả hàng tuần từ các biến workDays, workHours, và payRate (* là toán tử nhân). Giá trị kết quả được lưu vào biến weeklyPay.

10-12 Các hàng này xuất ba mục tuần tự là: chuỗi "Weekly Pay = ", giá trị của biến weeklyPay, và một ký tự xuống dòng.

Khi chạy, chương trình sẽ cho kết quả như sau:

```
Weekly Pay = 1445.625
```

Khi một biến được định nghĩa, giá trị của nó **không được định nghĩa** cho đến khi nó được gán cho một giá trị thật sự. Ví dụ, weeklyPay có một giá trị không được định nghĩa cho đến khi hàng 9 được thực thi. Việc gán giá trị cho một biến ở lần đầu tiên được gọi là **khởi tạo**. Việc chắc chắn rằng một

biến được khởi tạo trước khi nó được sử dụng trong bất kỳ công việc tính toán nào là rất quan trọng.

Một biến có thể được định nghĩa và khởi tạo cùng lúc. Điều này được xem như là một thói quen lập trình tốt bởi vì nó giành trước khả năng sử dụng biến trước khi nó được khởi tạo. Danh sách 1.3 là một phiên bản sửa lại của danh sách 1.2 mà có sử dụng kỹ thuật này. Trong mọi mục đích khác nhau thì hai chương trình là tương tương.

Danh sách 1.3

```
1  #include <iostream.h>

2  int main (void)
3  {
4        int          workDays = 5;
5        float  workHours = 7.5;
6        float  payRate = 38.55;
7        float  weeklyPay = workDays * workHours * payRate;

8        cout << "Weekly Pay = ";
9        cout << weeklyPay;
10       cout << '\n';
11 }
```

1.5. Xuất/nhập đơn giản

Cách chung nhất mà một chương trình giao tiếp với thế giới bên ngoài là thông qua các thao tác xuất nhập hướng ký tự đơn giản. C++ cung cấp hai toán tử hữu dụng cho mục đích này là >> cho nhập và << cho xuất. Chúng ta đã thấy ví dụ của việc sử dụng toán tử xuất << rồi. Danh sách 1.4 sẽ minh họa thêm cho việc sử dụng toán tử nhập >>.

Danh sách 1.4

```
1  #include <iostream.h>

2  int main (void)
3  {
4        int          workDays = 5;
5        float  workHours = 7.5;
6        float  payRate, weeklyPay;

7        cout << "What is the hourly pay rate? ";
8        cin >> payRate;

9        weeklyPay = workDays * workHours * payRate;
10       cout << "Weekly Pay = ";
11       cout << weeklyPay;
12       cout << '\n';
13 }
```

Chú giải

7 Hàng này xuất ra lời nhắc nhở What is the hourly pay rate? để tìm dữ liệu nhập của người dùng.

8 Hàng này đọc giá trị nhập được gõ bởi người dùng và sao chép giá trị này tới biến payRate. Toán tử nhập >> lấy một dòng nhập như là toán hạng trái (cin là dòng nhập chuẩn của C++ mà tương ứng với dữ liệu được nhập vào từ bàn phím) và một biến (mà dữ liệu nhập được sao chép tới) như là toán hạng phải.

9-13 Phần còn lại của chương trình là như trước.

Khi chạy, chương trình sẽ xuất ra màn hình như sau (dữ liệu nhập của người dùng được in đậm):

```
What is the hourly pay rate? 33.55
Weekly Pay = 1258.125
```

Cả hai << và >> trả về toán hạng trái như là kết quả của chúng, cho phép nhiều thao tác nhập hay nhiều thao tác xuất được kết hợp trong một câu lệnh. Điều này được minh họa trong danh sách 1.5 với trường hợp cho phép nhập cả số giờ làm việc mỗi ngày và số tiền phải trả mỗi giờ.

Danh sách 1.5

```
1  #include <iostream.h>

2  int main (void)
3  {
4       int          workDays = 5;
5       float  workHours, payRate, weeklyPay;

6       cout << "What are the work hours and the hourly pay rate? ";
7       cin >> workHours >> payRate;

8       weeklyPay = workDays * workHours * payRate;
9       cout << "Weekly Pay = " << weeklyPay << '\n';
10 }
```

Chú giải

7 Hàng này đọc hai giá trị nhập được nhập vào từ người dùng và chép tương ứng chúng tới hai biến workHours và payRate. Hai giá trị cần được tách biệt bởi một không gian trống (chẳng hạn, một hay là nhiều khoản trắng hay là các ký tự tab). Câu lệnh này tương đương với:

```
(cin >> workHours) >> payRate;
```

Vì kết quả của >> là toán hạng trái, (cin >> workHours) định giá cho cin mà sau đó được sử dụng như là toán hạng trái cho toán tử >> kế tiếp.

9 Hàng này là kết quả của việc kết hợp từ hàng 10 đến hàng 12 trong danh sách 1.4. Nó xuất "Weekly Pay = ", theo sau đó là giá trị của biến weeklyPay, và cuối cùng là một ký tự xuống dòng. Câu lệnh này tương đương với:

```
((cout << "Weekly Pay = ") << weeklyPay) << '\n';
```

Vì kết quả của << là toán hạng trái, (cout << "Weekly Pay = ") định giá cho cout mà sau đó được sử dụng như là toán hạng trái của toán tử << kế tiếp.

Khi chạy, chương trình sẽ hiển thị như sau:

```
What are the work hours and the hourly pay rate? 7.5  33.55
Weekly Pay = 1258.125
```

1.6. Chú thích

Chú thích thường là một đoạn văn bản. Nó được dùng để giải thích một vài khía cạnh của chương trình. Trình biên dịch bỏ qua hoàn toàn các chú thích trong chương trình. Tuy nhiên các chú thích này là có ý nghĩa và đôi khi là rất quan trọng đối với người đọc (người xem các mã chương trình có sẵn) và người phát triển phần mềm. C++ cung cấp hai loại chú thích:

- Những gì sau // (cho đến khi kết thúc hàng mà nó xuất hiện) được xem như là một chú thích.
- Những gì đóng ngoặc trong cặp dấu /* và */ được xem như là một chú thích.

Danh sách 1.6 minh họa việc sử dụng cả hai hình thức này.

Danh sách 1.6

```
1  #include <iostream.h>
2  /* Chuong trinh nay tinh toan tong so tien phai tra hang tuan cho mot cong nhan dua tren tong so gio
3  lam viec va so tien phai tra moi gio. */
4
   int main (void)
5  {
6      int     workDays = 5;          // so ngay lam viec trong tuan
7      float   workHours = 7.5;       // so gio lam viec trong ngay
8      float   payRate = 33.50;       // so tien phai tra moi gio
9      float   weeklyPay;             // tong so tien phai tra moi tuan
10
       weeklyPay = workDays * workHours * payRate;
11     cout << "Weekly Pay = " << weeklyPay << '\n';
12 }
13
```

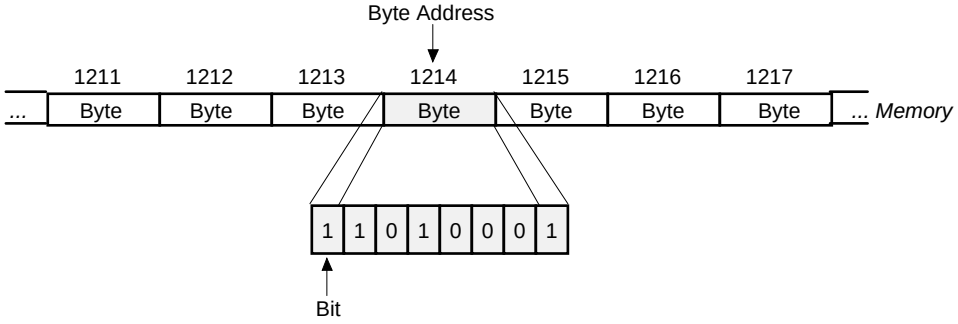
Các chú thích nên được sử dụng để tăng cường (không phải gây trở ngại) việc đọc một chương trình. Một vài điểm sau nên được chú ý:

- Chú thích nên dễ đọc và dễ hiểu hơn sự giải thích thông qua mã chương trình. Thà là không có chú thích nào còn hơn có một chú thích phức tạp dễ gây lầm lẫn một cách không cần thiết.
- Sử dụng quá nhiều chú thích có thể dẫn đến khó đọc. Một chương trình chứa quá nhiều chú thích làm bạn khó có thể thấy mã thì không thể nào được xem như là một chương trình dễ đọc và dễ hiểu.
- Việc sử dụng các tên mô tả có ý nghĩa cho các biến và các thực thể khác trong chương trình, và những chỗ thụt vào của mã có thể làm giảm đi việc sử dụng chú thích một cách đáng kể, và cũng giúp cho lập trình viên dễ đọc và kiểm soát chương trình.

1.7. Bộ nhớ

Máy tính sử dụng bộ nhớ truy xuất ngẫu nhiên (RAM) để lưu trữ mã chương trình thực thi và dữ liệu mà chương trình thực hiện. Bộ nhớ này có thể được xem như là một chuỗi tuần tự các **bit nhị phân** (0 hoặc 1). Thông thường, bộ nhớ được chia thành những nhóm 8 bit liên tiếp (gọi là **byte**). Các byte được định vị liên tục. Vì thế mỗi byte có thể được chỉ định duy nhất bởi **địa chỉ** (xem Hình 1.2).

Hình 1.2 Các bit và các byte trong bộ nhớ

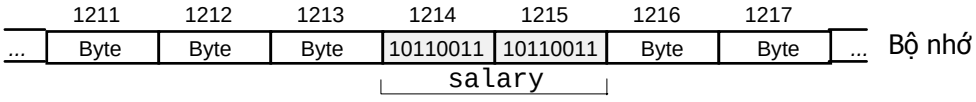


Trình biên dịch C++ phát ra mã có thể thực thi mà sắp xếp các thực thể dữ liệu tới các vị trí bộ nhớ. Ví dụ, định nghĩa biến

```
int salary = 65000;
```

làm cho trình biên dịch cấp phát một vài byte cho biến salary. Số byte cần được cấp phát và phương thức được sử dụng cho việc biểu diễn nhị phân của số nguyên phụ thuộc vào sự thi hành cụ thể của C++. Trình biên dịch sử dụng địa chỉ của byte đầu tiên của biến salary được cấp phát để tham khảo tới nó. Việc gán trên làm cho giá trị 65000 được lưu trữ như là một số nguyên bù hai trong hai byte được cấp phát (xem Hình 1.3).

Hình 1.3 Biểu diễn của một số nguyên trong bộ nhớ.



một số nguyên 2 byte ở địa chỉ 1214

Trong khi việc biểu diễn nhị phân chính xác của một hạng mục dữ liệu là ít khi được các lập trình viên quan tâm tới thì việc tổ chức chung của bộ nhớ và sử dụng các địa chỉ để tham khảo tới các hạng mục dữ liệu là rất quan trọng.

1.8. Số nguyên

Biến số nguyên có thể được định nghĩa là kiểu short, int, hay long. Chỉ khác nhau là số int sử dụng nhiều hơn hoặc ít nhất bằng số byte như là một số short, và một số long sử dụng nhiều hơn hoặc ít nhất cùng số byte với một số int. Ví dụ, trên máy tính cá nhân thì một số short sử dụng 2 byte, một số int cũng 2 byte, và một số long là 4 byte.

```
short age = 20;
int salary = 65000;
long price = 4500000;
```

Mặc định, một biến số nguyên được giả sử là có dấu (chẳng hạn, có một sự biểu diễn dấu để mà nó có thể biểu diễn các giá trị dương cũng như là các giá trị âm). Tuy nhiên, một số nguyên có thể được định nghĩa là không có dấu bằng cách sử dụng từ khóa unsigned trong định nghĩa của nó. Từ khóa signed cũng được cho phép nhưng hơi dư thừa.

```
unsigned short age = 20;
unsigned int salary = 65000;
unsigned long price = 4500000;
```

Số nguyên (ví dụ, 1984) luôn luôn được giả sử là kiểu int, trừ khi có một hậu tố L hoặc l thì nó được hiểu là kiểu long. Một số nguyên cũng có thể được đặc tả sử dụng hậu tố là U hoặc u., ví dụ:

```
1984L  1984l  1984U  1984u  1984LU  1984ul
```

1.9. Số thực

Biến số thực có thể được định nghĩa là kiểu float hay double. Kiểu double sử dụng nhiều byte hơn và vì thế cho miền lớn hơn và chính xác hơn để biểu diễn các số thực. Ví dụ, trên các máy tính cá nhân một số float sử dụng 4 byte và một số double sử dụng 8 byte.

```
float  interestRate = 0.06;
double      pi = 3.141592654;
```

Số thực (ví dụ, 0.06) luôn luôn được giả sử là kiểu double, trừ phi có một hậu tố F hay f thì nó được hiểu là kiểu float, hoặc một hậu tố L hay l thì nó được hiểu là kiểu long double. Kiểu long double sử dụng nhiều byte hơn kiểu double cho độ chính xác tốt hơn (ví dụ, 10 byte trên các máy PC). Ví dụ:

```
0.06F       0.06f 3.141592654L          3.141592654l
```

Các số thực cũng có thể được biểu diễn theo cách ký hiệu hóa khoa học. Ví dụ, 0.002164 có thể được viết theo cách ký hiệu hóa khoa học như sau:

```
2.164E-3          or          2.164e-3
```

Ký tự E (hay e) thay cho *số mũ* (exponent). Cách ký hiệu hóa khoa học được thông dịch như sau:

$$2.164E\text{-}3 = 2.164 \times 10^{-3} = 0.002164$$

1.10.Ký tự

Biến ký tự được định nghĩa là kiểu char. Một biến ký tự chiếm một byte đơn để lưu giữ *mã* cho ký tự. Mã này là một giá trị số và phụ thuộc *hệ thống mã ký tự* đang được dùng (nghĩa là phụ thuộc máy). Hệ thống chung nhất là ASCII (American Standard Code for Information Interchange). Ví dụ, ký tự *A* có mã ASCII là 65, và ký tự *a* có mã ASCII là 97.

```
char  ch = 'A';
```

Giống như số nguyên, biến ký tự có thể được chỉ định là có dấu hoặc không dấu. Mặc định (trong hầu hết các hệ thống) char nghĩa là signed char. Tuy nhiên, trên vài hệ thống thì nó có nghĩa là unsigned char. Biến ký tự có dấu có thể giữ giá trị số trong miền giá trị từ -128 tới 127. Biến ký tự không dấu có thể giữ giá trị số trong miền giá trị từ 0 tớ 255. Kết quả là, cả hai thường được dùng để biểu diễn các số nguyên nhỏ trong chương trình (và có thể được đánh dấu các giá trị số như là số nguyên):

```
signed char      offset = -88;
unsigned char    row = 2, column = 26;
```

Ký tự được viết bằng cách đóng dấu ký tự giữa cặp nháy đơn (ví dụ, 'A'). Các ký tự mà không thể in ra được biểu diễn bằng việc sử dụng các mã escape. Ví dụ:

```
'\n'      // xuống hàng mới
'\r'      // phím xuống dòng
```

```
'\t'    // phím tab ngang
'\v'    // phím tab dọc
'\b'    // phím lùi
```

Các dấu nháy đơn, nháy đôi và ký tự gạch chéo ngược cũng có thể sử dụng ký hiệu escape:

```
'\''    // trích dẫn đơn (')
'\"'    // trích dẫn đôi (")
'\\'    // dấu vạch chéo ngược (\)
```

Ký tự cũng có thể được chỉ định rõ sử dụng giá trị mã số của chúng. Mã escape tổng quát \ooo (nghĩa là, 3 ký tự số cơ số 8 theo sau một dấu gạch chéo ngược) được sử dụng cho mục đích này. Ví dụ (giả sử ASCII):

```
'\12'   // hàng mới (mã thập phân = 10)
'\11'   // tab ngang (mã thập phân = 9)
'\101'  // 'A' (mã thập phân = 65)
'\0'    // rỗng (mã thập phân = 0)
```

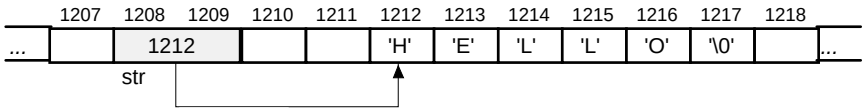
1.11.Chuỗi

Chuỗi là một dãy liên tiếp các ký tự được kết thúc bằng một ký tự null. **Biến chuỗi** được định nghĩa kiểu char* (nghĩa là, con trỏ ký tự). Con trỏ đơn giản chỉ là một vị trí trong bộ nhớ. (Các con trỏ sẽ được thảo luận trong chương 5). Vì thế biến chuỗi chứa đựng địa chỉ của ký tự đầu tiên trong chuỗi. Ví dụ, xem xét định nghĩa:

```
char * str = "HELLO";
```

Hình 1.4 minh họa biến chuỗi và chuỗi "HELLO" có thể xuất hiện như thế nào trong bộ nhớ.

Hình 1.4 Chuỗi và biến chuỗi trong bộ nhớ



Chuỗi được viết bằng cách đóng ngoặc các ký tự của nó bên trong cặp dấu nháy kép (ví dụ, "HELLO"). Trình biên dịch luôn luôn thêm vào một ký tự null tới một hằng chuỗi để đánh dấu điểm kết thúc. Các ký tự chuỗi có thể được đặc tả sử dụng bất kỳ ký hiệu nào dùng để đặc tả các ký tự. Ví dụ:

```
"Name\tAddress\tTelephone"          // các từ phân cách
"ASCII character 65: \101"          // 'A' được đặc tả như '101'
```

Chuỗi dài có thể nới rộng qua khỏi một hàng đơn, trong trường hợp này thì mỗi hàng trước phải được kết thúc bằng một dấu vạch chéo ngược. Ví dụ:

```
"Example to show \
the use of backslash for \
writing a long string"
```

Dấu \ trong ngữ cảnh này có nghĩa là phần còn lại của chuỗi được tiếp tục trên hàng kế tiếp. Chuỗi trên tương đương với chuỗi được viết trên hàng đơn như sau:

```
"Example to show the use of backslash for writing a long string"
```

Một lỗi lập trình chung thường xảy ra là lập trình viên thường nhầm lẫn một chuỗi ký tự đơn (ví dụ, "A") với một ký tự đơn (ví dụ, 'A'). Hai điều này là không tương đương. Chuỗi ký tự đơn gồm 2 byte (ký tự 'A' được theo sau là ký tự '\0'),trong khi ký tự đơn gồm chỉ một byte duy nhất.

Chuỗi ngắn nhất có thể có là chuỗi rỗng ("") chỉ chứa ký tự null.

1.12.Tên

Ngôn ngữ lập trình sử dụng tên để tham khảo tới các thực thể khác nhau dùng để tạo ra chương trình. Chúng ta cũng đã thấy các ví dụ của một loại các tên (nghĩa là tên biến) như thế. Các loại khác gồm: tên hàm, tên kiểu, và tên macro.

Sử dụng tên rất tiện lợi cho việc lập trình, nó cho phép lập trình viên tổ chức dữ liệu theo cách thức mà con người có thể hiểu được. Tên không được đưa vào mã có thể thực thi được tạo ra bởi trình biên dịch. Ví dụ, một biến temperature cuối cùng trở thành một vài byte bộ nhớ mà được tham khảo tới bởi các mã có thể thực thi thông qua địa chỉ của nó (không thông qua tên của nó).

C++ áp đặt những luật sau để xây dựng các tên hợp lệ (cũng được gọi là các **định danh**). Một tên chứa một hay nhiều ký tự, mỗi ký tự có thể là một chữ cái (nghĩa là, 'A'-'Z' và 'a'-'z'), một số (nghĩa là, '0'-'9'), hoặc một ký tự gạch dưới ('_'), ngoại trừ ký tự đầu tiên không thể là một số. Các ký tự viết hoa và viết thường là khác nhau.Ví dụ:

```
salary          // định danh hợp lệ
salary2         // định danh hợp lệ
2salary         // định danh không hợp lệ (bắt đầu với một số)
_salary         // định danh hợp lệ
Salary          // hợp lệ nhưng khác với salary
```

C++ không có giới hạn số ký tự của một định danh. Tuy nhiên, hầu hết thi công lại áp đặt sự giới hạn này nhưng thường đủ lớn để không gây bận tâm cho các lập trình viên (ví dụ 255 ký tự).

Một số từ được giữ bởi C++ cho một số mục đích riêng và không thể được dùng cho các định danh. Những từ này được gọi là **từ khóa** (keyword) và được tổng kết trong bảng 1.3:

Bảng 1.3 Các từ khóa C++.

asm	continue	float	new	signed	try
auto	default	for	operator	sizeof	typedef
break	delete	friend	private	static	union
case	do	goto	protected	struct	unsigned
catch	double	if	public	switch	virtual
char	else	inline	register	template	void
class	enum	int	return	this	volatile
const	extern	long	short	throw	while

Bài tập cuối chương 1

1.1 Viết chương trình cho phép nhập vào một số đo nhiệt độ theo độ Fahrenheit và xuất ra nhiệt độ tương đương của nó theo độ Celsius, sử dụng công thức chuyển đổi:

$$^\circ C = \frac{5}{9}(^\circ F - 32)$$

Biên dịch và chạy chương trình. Việc thực hiện của nó giống như thế này:

```
Nhiet do theo do Fahrenheit: 41
41 do Fahrenheit = 5 do Celsius
```

1.2 Hàng nào trong các hàng sau biểu diễn việc định nghĩa biến là không hợp lệ?

```
int    n = -100;
unsigned int i = -100;
signed int = 2.9;
long m = 2, p = 4;
int    2k;
double x = 2 * m;
float y = y * 2;
unsigned double z = 0.0;
double d = 0.67F;
float f = 0.52L;
signed char = -1786;
char c = '$' + 2;
sign char h = '\111';
```

```
char *name = "Peter Pan";
unsigned char *num = "276811";
```

1.3 Các định danh nào sau đây là không hợp lệ?

```
identifier
seven_11
_unique_
gross-income
gross$income
2by2
default
average_weight_of_a_large_pizza
variable
object.oriented
```

1.4 Định nghĩa các biến để biểu diễn các mục sau đây:

- Tuổi của một người.
- Thu nhập của một nhân viên.
- Số từ trong một từ điển.
- Một ký tự alphabet.
- Một thông điệp chúc mừng.