

# Chapter 2 – Identifiers, Constants, Objects, & Assignment Statements

## 1. Identifiers

*Identifiers* are the names given to anything in C++ that needs a name. Valid identifiers in C++ must be formed according to a few simple rules. An identifier may start with an alphabetic character (A-Z, a-z) or the underscore ( \_ ), then may include any number of alphabetic characters, underscores, or digits (0, 1, 2, ..., 9). Spaces may not be included in an identifier.

### Examples of Valid Identifiers

Sam, SUE, \_go, nUmBeR, v1, v2, v3

### Examples of Invalid Identifiers

~~1~~, ~~2~~, ~~3~~, one potatoe, \$pay, a twit, dot period

Lastly, C++ is case sensitive. For example, the identifier “Sam” is not the same identifier as “SAM” or “sam”;

## 2. Constants, Objects, and Assignment Statements

Types and classes are not directly used in C++ programs. Instead, types and classes are expressed as *literal constants* or *objects*.

*Literal constants* are simple values, such as the number 42 and all of the state examples in Chapter 1. In general, it is only possible to use literal constants for the built-in types.

*Objects* are named instances of types and classes. Each object has all of the functionality of its type or class. One way to think of this is to regard the type or class as a set of blueprints or plans and the object as the finished product. Objects of the built-in types or other classes serve the same function as variables in other programming languages. Objects are sometimes referred to as *variable objects* or *variable identifiers*.

Objects must be declared before being used. Once declared, objects can have their state information altered, be used in expressions and have their member functions called.

Objects can be declared in statements of the following forms.

```
class-or-type object;
class-or-type object, object;
class-or-type object, object, ..., object;
```

Here are some example object declarations.

```
int year;
int month, day;
double rate;
```

Objects can be assigned values via the *assignment* statement. The assignment statement has the following form:

```
object = expression;
```

In the assignment statement, the expression on the right of the equal sign is evaluated and the results of that evaluation are copied into the *object* on the left of the equal sign. Here are some examples of assignment statements.

```
firstInitial = 'R';
hourlyRate = 7.55;
cookiesLeft = cookies % number_of_guests;
```

Assignment statements are very common statements. In addition, they are very versatile and can be used inside of some other statements (with the semicolon omitted). Declaration statements followed by assignments are valid and usual, but can be shortened to the following forms:

```
class object = expression;  
class object = expression, object = expression, ..., object = expression;
```

#### **Long Version Examples**

```
long cookiesLeft;  
cookiesLeft = cookies % numberOfGuests;  
char firstInitial;  
firstInitial = 'R';
```

#### **Shortened Version Examples**

```
long cookiesLeft = cookies % numberOfGuests;  
char firstInitial = 'R';
```

Placing *const* before a declaration of an object that is using the shortened version of declaration and assignment produces a *named constant* identifier. For example:

```
const long YEARS = 100;  
const long DAYS_IN_YEAR = 365;
```

Unlike a variable identifier, a named constant identifier may not have its value changed while the programming is running.

### **3. Example Program: As Time Goes By**

Here is an example program to calculate the number of days, weeks, hours, and minutes in 100 years. (As in this example, named constants are often declared outside of a function in order that they may be shared with all functions declared after them.)

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

const long YEARS = 100;
const long DAYS_IN_YEAR = 365;

void main()
{
    long leap_days = YEARS / 4;
    long days = YEARS * DAYS_IN_YEAR + leap_days;
    long weeks = days / 7;
    long hours = days * 24;

    cout << "100 years contain " << weeks << " weeks, "
         << days << " days, "
         << hours << " hours, and"
         << hours * 60 << " minutes.";
}
```

This program outputs:

*100 years contain 5217 weeks, 36525 days, 876600 hours, and 52596000 minutes.*

#### **Exercises**

1. What is an identifier?
2. Which of the following identifiers are valid?
 

Dog	_Cat	1984	_1984
one-two	one two	one_two	al
3. What is a literal constant?
4. What is an object?
5. What is declaring an object?
6. What is a variable identifier?
7. What is a named constant identifier?
8. What is an assignment statement?

9. What are the values of A, B, and C after each assignment statement in the example program has been executed? If the value is unknown, write a question mark in the blank. What is the final output of the program?

```

#include <iostream>
using namespace std;
void main()
{
    int a, b, c;
    a = 3;
    b = a + 2;
    c = b + a;
    a = b + c;
    cout << a << " " << b << " " << c;
}

```

A	B	C

### **Programming Problem 2.1**

Use the following declarations to write a program to calculate and output the area of a plot of land. Units are feet. Output should be expressed in terms of square feet.

```

double length = 88.5;
double width = 100.0;

```

### **Programming Problem 2.2**

Calculate and output the following powers of 2:  $2^1$ ,  $2^2$ ,  $2^3$ ,  $2^4$ ,  $2^5$ . In this program, each power of 2 should be assigned to its own object and the output should be produced from that object.