Identifiers and Keywords

Identifiers are the names of variables, functions, labels, and various other user-defined objects. Rules of a valid identifier:

1. An identifier must begin with an alphabetic character or underscore _
2. Alphabetic characters in an identifier can be lowercase or uppercase letters
3. An identifier can contain digits, but not as the first character
4. An identifier can be of any length

Keywords are the commands that make up the C++ language. These keywords cannot be used for identifiers.

Example:

Determine which of the following names are valid identifiers.

1. xsum	2. x_sum	3. tax-rate
4. perimeter	5. sec^2	6. degrees_C
7. count	8. void	9. f(x)
10. m/s	11. Final_Value	12. w1.1

Solution

1. valid
2. valid
3. invalid character (-), replacement tax_rate
4. valid
5. invalid character (^), replacement sec_sqrd
6. valid
7. valid
8. invalid, keyword, replacement void_term
9. invalid characters (( )), replacement fx
10. invalid character (/), replacement m_per_s
11. valid
12. invalid character (.), replacement w1_1

Constants and Variables

Constants are specific values such as 2 , 3.1416 , or -15.

Variables are memory locations that are assigned a name or identifier.

Example:

```
int  x , y=1;
float  mark;
double total , average;
char  ch = 'A';
```

Notes:

- We can declare variables anywhere in the **`main()`** function.
- We must declare a variable before we use it.
- C++ is case sensitive, i.e. it distinguishes uppercase letters from lowercase letters. Thus, `Total`, `TOTAL`, and `total` represent three different variables.

Standard Input and Output

In order to use input and output statements in our programs, we must include the library:

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

a. **cout STATEMENT**

is used to print values and explanatory text to the screen.

```
cout << "age = " << age << " years" << endl;
```

if the value of age is 20, the output of the statement is

age = 20 years

b. **cin STATEMENT**

is used to enter values from the keyboard when a program is executed.

```
cin >> age;
```

`cin` also allows us to enter multiple input values:

```
cin >> length >> width;
```

Note:

`cin` statement is usually preceded by `cout` statement to describe the information the user should enter from the keyboard:

```
cout<< "Enter the length and width:" << endl;
cin >> length >> width;
```

The output of the above statements is:

Enter the length and width of a rectangle:

15.5 10

Escape sequences

are used with `cout` statement to produce a formatted output. The backslash (\) is used to form these sequences.

Sequence	Character represented
\a	alert (bell) character
\b	backspace
\n	new line
\r	carriage return
\t	horizontal tab
\v	vertical tab
\\	backslash
\?	question mark
\'	single quote
\"	double quote

Examples:

```
cout << "\"The End.\"" << endl;
```

Output is: "The End."

```
cout << "Column1\tColumn2\tColumn3" << endl;
```

Output is: Column1 Column2 Column3

```
cout << "Your mark is: \n90";
```

Output is: your mark is:

90

Data Types

are used to specify the types of values that will be contained in variables.

In C++, data types are classified into:

- *Numeric data types* are either integers (short, int, long) or floating-point values (float, double, long double).
- *Nonnumeric data types* are alphabetic and special characters (char).

Type	Size (Bytes)	Range
char	1	-128 to 127
short	2	-32,768 to 32,767
int	2	-32,768 to 32,767
long	4	-2,147,483,648 to 2,147,483,647
float	4	$3.4 \times (10^{-38})$ to $3.4 \times (10^{+38})$
double	8	$1.7 \times (10^{-308})$ to $1.7 \times (10^{+308})$
long double	10	$3.4 \times (10^{-4932})$ to $1.1 \times (10^{+4932})$

Example:
Write a C++ program that computes the area of rectangle with given (fixed) length and width.

```
#include <iostream>
int main()
{
  int  length , width , area;
  length = 4;
  width = 5;
  area = length * width;
  std::cout<<"The area of rectangle is: "<<area<<std::endl;
  return 0;
}
```

Example:
Write a C++ program that reads the length and width of a rectangle, and prints its area.

```
#include <iostream>
using namespace std;
int main()
{
  int  length , width , area;
```

```
  cout << "Enter length :" << endl;
  cin >> length;
  cout << "Enter width :" << endl;
  cin >> width;
  area = length * width;
  cout <<"The area of rectangle is: "<< area <<endl;
  return 0;
}
```

The "using" directive can be used as "using namespace std;" which enables a program to use all the names in any standard C++ header (such as <iostream>) that a program might include. From this point forward in the lectures, we'll use the preceding directive in our programs.

Character

Each character in the computer is represented by ASCII (American Standard Code for Information Interchange). Examples of ASCII Codes:

Character	ASCII Code	Integer Equivalent
new line, \n	0001010	10
$	0100100	36
%	0100101	37
3	0110011	51
A	1000001	65
a	1100001	97
b	1100010	98

A total of 128 characters can be represented in ASCII.

Example:

```
#include <iostream>
using namespace std;
int main()
{
     char  letter = 'A' , symbol = '$';
     char  c = 97;
     char  ch;
     cout << "Enter character :" << endl;
     cin >> ch;
     cout << "letter = " << letter << endl
          << "Symbol = " << symbol << endl
```

```
          << "c = " << c << endl
          << "ch = " << ch << endl;
     return 0;
}
```

Symbolic Constant

is declared by using the keyword **`const`**. It is usually defined with uppercase identifier. For example:

```
const double PI = 3.141593;
```

Now statements that need to use the value of π can use the symbolic constant `PI` instead of 3.141593.

Note: The value of symbolic constant cannot be changed.

Example:
Write a C++ Program that reads the radius of a circle and prints its area and perimeter.

```
#include <iostream>
using namespace std;
int main()
{
     const double PI = 3.141593;
     double  radius , area , perimeter;
     cout << "Enter the radius: " << endl;
     cin >> radius;
     area = PI * radius * radius;
     perimeter = 2 * PI * radius;
     cout << "The area of circle: " << area << endl;
     cout << "The perimeter of circle: " << perimeter
          << endl;
     return 0;
}
```

Assignment Statements

An assignment statement is used to assign a value to an identifier.

```
identifier = expression;
```

where **expression** can be constant, another variable, or the result of an operation.

```
double  sum = 10.5;
int  x = 3;
```

```
double  sum;
int  x;
.
.
sum = 10.5;
x = 3;
```

Multiple assignments are also allowed in C++:

```
x = y = z = 0;
```

We can also assign a value from one variable to another:

```
sum = total_marks;
```

If we assign a value of one data type to a variable of a different data type then a conversion must occur during the execution of the statement. Sometimes the conversion can result in information being lost.

```
int  a;
.
.
a = 12.8;
```

Here, the variable **a** will store only 12 because it is defined as integer. To make the **numeric conversion** works properly, we convert the value to a higher data type according to the following order:

high: long double
double
float
long
int
low: short

Exercises

1. Write a C++ program that asks for a temperature in Fahrenheit and displays it in Celsius. Use the formula:

$$C^0 = \frac{5}{-}(F - 32)$$

2. What is wrong with the following "declarations" and "comment":

```
int x = y = 22 ;
cout <<" Hello, /* change? */ world. \n ";
```

3. How do the following two statements differ:

```
char ch = 'A';
char ch = 65;
```

4. Determine which of the following a valid identifier is. If it is not valid, tell why:

a. r2d3
b. H20
c. secondCousinOnceRemoved
d. 2ndBirtday
e. The-Arab_nation
f. _Time_
g. _12345
h. x(3)
i. cost_in_$