المعرفات والكلمات الرئيسية

Identifiers and Keywords

1. Identifiers and Keywords

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

- An identifier must begin with an alphabetic character or underscore _
- Alphabetic characters in an identifier can be lowercase or uppercase letters
- An identifier can contain digits, but not as the first character
- 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

2.   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.

3. Standard Input and Output

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

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


3.1 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

3.2 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

4. 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

5. 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.

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

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

```cpp
#include <iostream.h>
void 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;
}
```

6. 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:
```cpp
#include <iostream.h>
void 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;
}
```

7. Symbolic Constant

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

```cpp
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 (المحيط).

```
#include <iostream.h>
void 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;
}
```

8. 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;          double  sum;
int  x = 3;                  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

9. Operators

Operators in C++ are classified into: arithmetic, logical, relational, and bitwise operators.

Arithmetic operators

Operator	Action
-	Subtraction, also unary minus
+	Addition
*	Multiplication
/	Division
%	Modulus
--	Decrement
++	Increment

Examples:

```
z = x + y;
area_square = side * side;
area_triangle = (base*height)/2;
z = x % y;
```

Notes:

- The result of a binary operation with values of the same type is another value of the same type.

- If a binary operation is performed between values with different types, then the value with the lower type is converted to the higher type, and thus the operation is performed with values of the same type.

- When you apply / to an integer, any remainder will be truncated.

- The modulus operator % produces the remainder of an integer division. It cannot be used with floating-point types.

10. Cast (الملقى ، المرمي) operator

A cast is a special operator that forces one data type to be converted into another.

Example
Write a C++ program that reads two marks and prints the average.

```
#include <iostream.h>
void main()
{
    int  mark1, mark2, sum, count = 2;
    float  average;
    cout << "Enter first mark: " << endl;
    cin >> mark1;
    cout << "Enter second mark: " << endl;
    cin >> mark2;
    sum = mark1 + mark2;
    average = sum/count;
    cout << "The average is: " << average << endl;
}
```

if `mark1` is 90 and `mark2` is 91 then average is 90.0 not 90.5.

To compute the `average` correctly, we use cast operator as follows:

```
average = (float)sum/(float)count;
```

Note that that cast operator affects only the value used in the computation, it does not change the type of the variables `sum` and `count`.

Priority of arithmetic operations

Precedence	Operator	Associativity
1	( )	innermost first
2	Unary + - cast	right to left
3	Binary * / %	left to right
4	Binary + -	left to right

Example:

Let us solve the following equation according to the priority of operations:

$$12*m + (m*n \ \% \ 13 + m/n) * k/10$$

Assume   m = 12 , n = 5 and k = 20

Sub expression	Result	Expression after each step
m * n	60	12*m+(60%13+m/n)*k/10
60 % 13	8	12*m+(8+m/n)*k/10
m / n	2	12*m +(8+2)*k/10
8 + 2	10	12*m+10*k/10
12 * m	144	144+10*k/10
10 * k	200	144+200/10

200 /10	20	144+20
144 + 20	164	164

Example:

Write a C++ program to compute the volume of a sphere.

$$v = 4 * \pi * r^3/3$$

```cpp
#include <iostream.h>
void main()
{
  const float PI = 3.141593;
  float  radius , volume;
  cout << "Enter the radius: " << endl;
  cin >> radius;
  volume = (4.0 * PI * radius * radius * radius)/3.0;
  cout << "The volume of sphere: " << volume << endl;
}
```

Example

Write a C++ program to compute the following equation:

$$f = \frac{x^3 - 2x^2 + x - 6.3}{x^2 + 0.505x - 3.14}$$

```cpp
#include <iostream.h>
void main()
{
float x, f;
cout << "Enter a value of x: " << endl;
cin >> x;
f = (x*x*x - 2*x*x + x - 6.3)/(x*x + 0.505*x - 3.14);
cout << "f = " << f << endl;
}
```

The statement

```cpp
f = (x*x*x - 2*x*x + x - 6.3)/(x*x + 0.505*x - 3.14);
```

can also be written as

```
f = (x*x*x – 2*x*x + x – 6.3)/
    (x*x + 0.505*x – 3.14);
```

Or
```
float  numerator, denominator;
numerator = x*x*x – 2*x*x + x – 6.3;
denominator = x*x + 0.505*x – 3.14;
f = numerator / denominator;
```

11. Overflow and Underflow

When the result of an arithmetic operation exceeds the range of the variable's data type, an error called *overflow* occurs.

Example

```
float  x = 2.5e30;    //  x  = 2.5 × 10³⁰
float  y = 1.0e30;    //  y = 1.0 × 10³⁰
float  z;
z = x * y;
```

Here, the value of z will be 2.5e60, i.e. *overflow*. C++ generates an error message "Floating-point error: Overflow".

Similarly, when the result of an operation is too small to store in the memory allocated for the variable, an error called *underflow* occurs.

Example

```
float  x = 2.5e-30;    //  x  = 2.5 × 10⁻³⁰
float  y = 1.0e-30;    //  y = 1.0 × 10⁻³⁰
float  z;
z = x * y;
```

Here, the value of z will be 2.5e-60, i.e. *underflow*. C++ replaces this value by zero.

12. Increment / Decrement operators

are applied either in a **prefix position** (before the identifier) as in

++count, or in a **postfix position** (after the identifier) as in count++.

The statement

```
x++;
```

is equal to the statement

```
x = x + 1;
```

and

```
--y;
```

is equal to the statement

```
y = y - 1;
```

However, there is a difference between the prefix and postfix forms when you use these operators in an **expression**.

The statement

```
w  =   ++x   -   y;
```

is equivalent to the statements

```
x = x + 1;
w = x - y;
```

while the statement

```
w = x++   -   y;
```

is equivalent to the statements

```
w = x - y;
x = x + 1;
```

Example

```
#include <iostream.h>
void main()
{
int  x , y , z;
x = 2;
```

```
y = 5;
z = x++ + y;
cout<<"x="<< x <<" y="<< y <<" z="<< z << endl;
z = ++x + y--;
cout<<"x="<< x <<" y="<< y <<" z="<< z << endl;
}
```

13. Logical operators

Operator	Action
&&	AND
||	OR
!	NOT

Example:

```
(a || b) && !(a && b)
```

14. Relational operators

Operator	Action
>	Greater than
>=	Greater than or equal
<	Less than
<=	Less than or equal
==	Is equal
!=	Not equal

Result: 1 True
        0 False

Example:

```
10>5 && !(10<9) || 3<=4
```

In this case the result is true.

15. Bitwise operators

Operator	Action
&	AND

|	OR
^	Exclusive OR (XOR)
~	One's complement (NOT)
>>	Shift right
<<	Shift left

Example:

```
int  x=10, y=2, r;
r = x & y;
r= x | y;
r = x ^ y;
r = x >> 1;
```

16. Assignment Operators

```
    =
    +=
    -=
    *=
    /=
    %=
```

Example:

```
x = x + 3;        sum = sum + x;       d = d / 4.5;
x += 3;           sum += x;            d /= 4.5;

r = r % 2;
r %= 2;
```

Exercises

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

$$C^0 = \frac{5}{9}(F - 32)$$

2. Write a C++ program to perform the following equations:

a) $z = 1 - \dfrac{x^2}{2!} + \dfrac{x^4}{4!}$

b) $z = \dfrac{x^3 - 4x^2 + x}{x^2 + 2x + 2}$

c) $z = [(x - y)^2 - (x + y)]/32$

3. Consider the arithmetic expressions

   1.   a * b / (-c * 31 % 13) * d

   2.   a * (b * b) – (c * b) + d

   Write the order in which the operations will be executed?

4. What is the computation sequence of the following expression

             (a + b / (c - 5)) / ((d + 7) / (e - 37) / 3)

   if a=10 , b=20 , c=14 , d=8, and e=40.

5. For each the following algebraic expressions write an equivalent C++ arithmetic expression.

a) $\dfrac{a^3 - b^2}{c^2 + 25}$

b) $\dfrac{1}{x} + \dfrac{1}{x^2} + \dfrac{1}{x^3} + \dfrac{1}{x^4}$

c) $x + y^2 + \dfrac{t}{z}$

6. Determine the values of the variables for each of the following C++ statements:

a)   z = x++*y;

b)   z = 2*++x*y;

c) `x += 4+--y/x---3;`

d) `y %= x;`

Assume that x=4 , y=6. Assume that all variables are integers.