SSCM 1313
C++ COMPUTER PROGRAMMING

Chapter 2:
Variable and Data

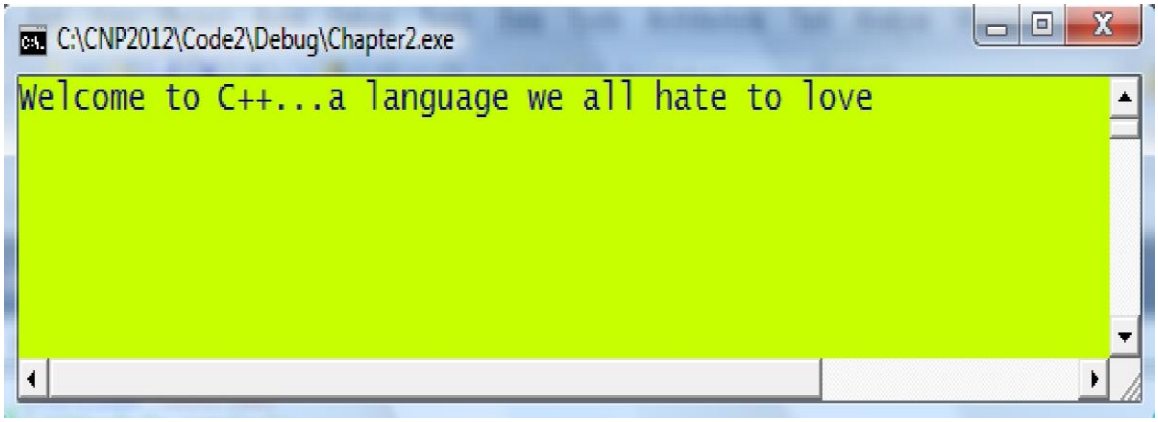
Authors:
Farhana Johar
Professor Dr. Shaharuddin Salleh

Chapter 2

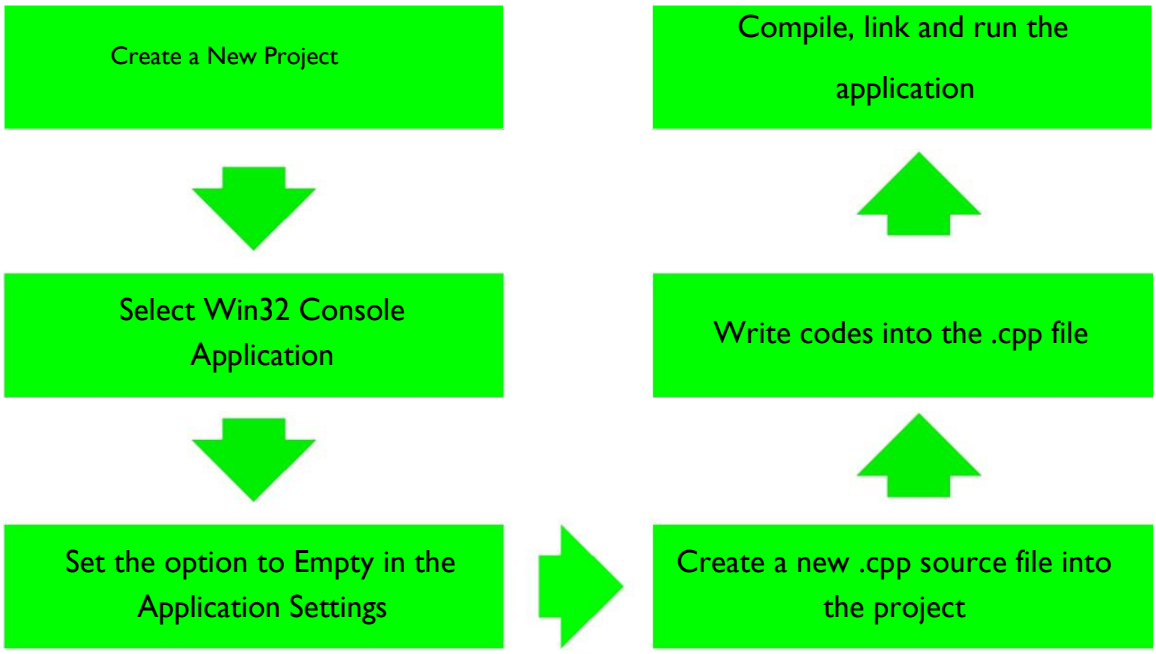
Variable and Data

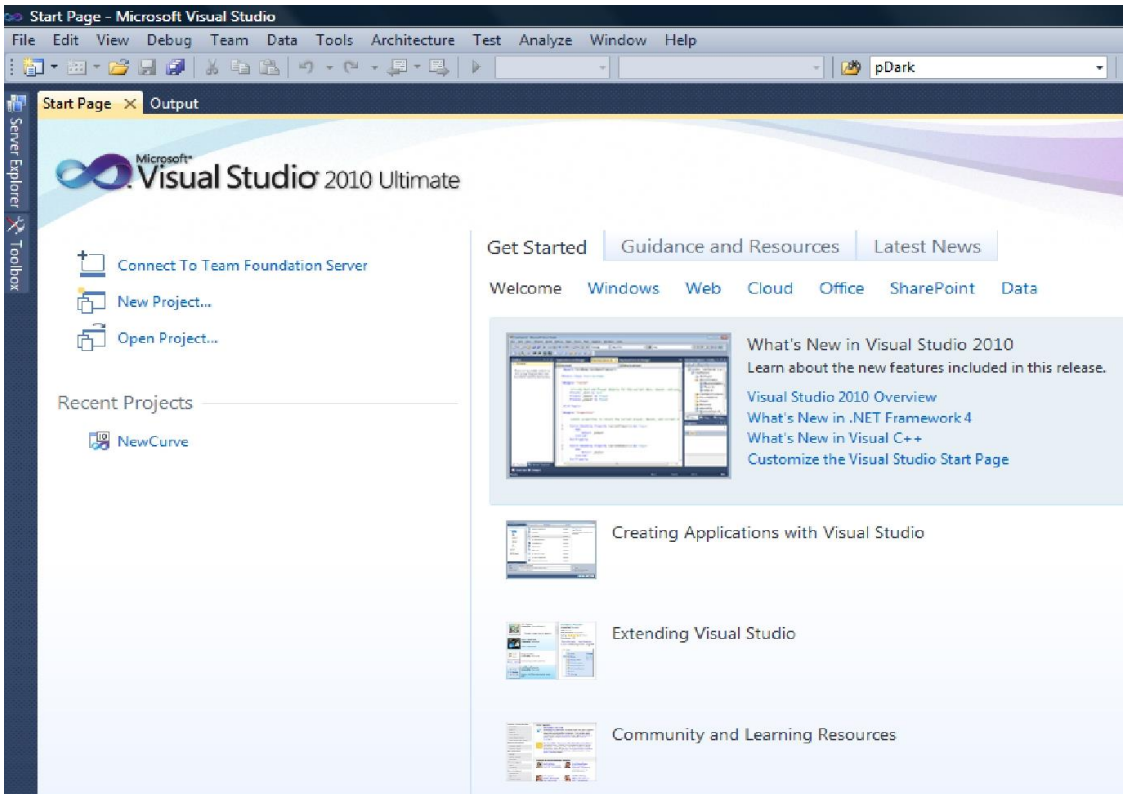
Code1A.cpp: The very first C++ program.

```
#include <iostream>

Using namespace std;

Void main()
{
   cout <<"Welcome to C++...a language we all hate to
        love"<<endl;
   cin.get();
}
```



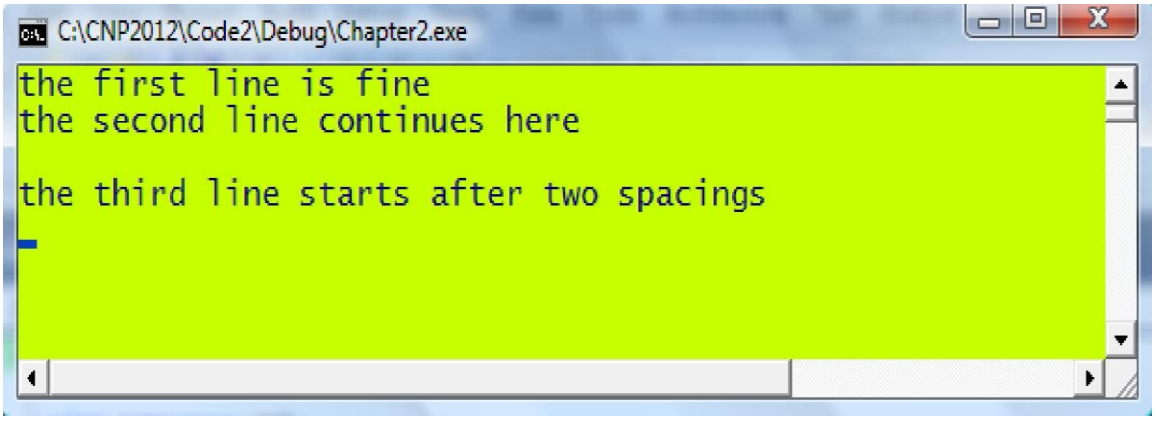
Getting Started with Visual Studio 2010





Code 2A.cpp: endl illustration.

```
#include <iostream>

using namespace std;

void main()
{
     cout << "the first line is fine" << endl;
     cout << "the second line ";
     cout << "continues here";
     cout << endl << endl
          << "the third line starts after two spacings"
          << endl;
     cin.get();
}
```



Variable/Data Types

Integer An integer number that does not support decimal points.

```
int i,p,q,r;

i=0;  p=1023;  q=-27;  r=0x4B;
```

Floating point A real number that supports decimal points.

```
float q;

double x,y,z;

q=2.75  ;  x=-4.0;  y=23.0794532195770432;
z=5.0705;
```

Character The smallest element in the Roman text system.

```
char  ch,t,u;

ch='r';  t='$';  u='K';
```

String An array of characters.

```
char  str[20];

char  *pqr="Hongkong";

strcpy(str,"Taiwan");
```

Name for a Variable

Any name can be used as long as it abides by the following rules:

- The name must not include a blank character.
- The name must be a single word consisting of alphanumeric characters but the use of any symbol below is illegal:

  , : % + - * / @ ! ^ & = # ~ ‘ “ – { } ( ) < > ? . | \ [ ] $

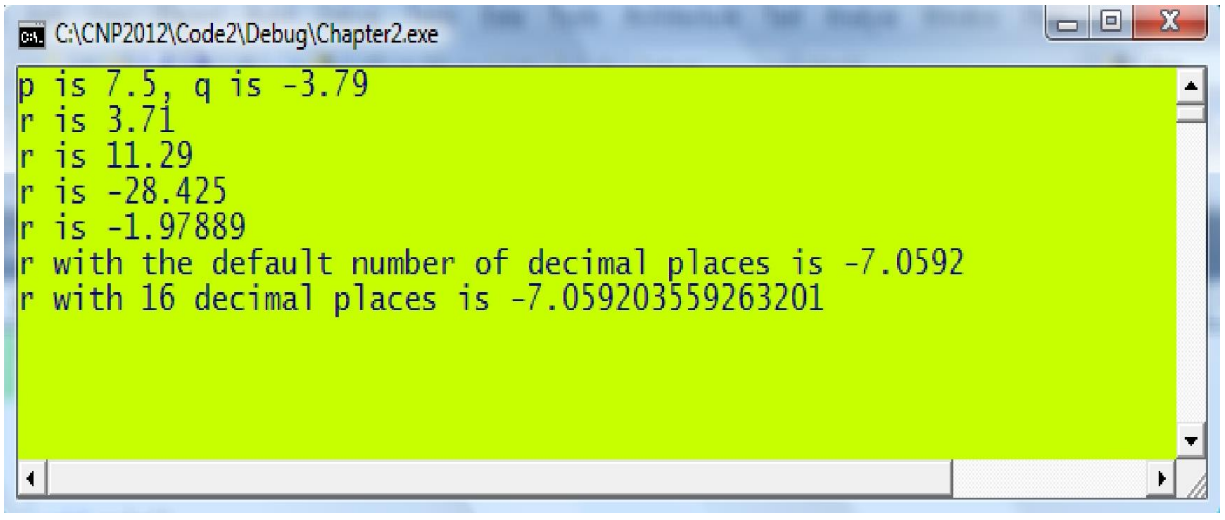
- The name must not be a reserved word in C++. A *reserved word* refers to the directives or commands that are native to the language. They include the followings:

`void`	`for`	`typedef`
`define`	`while`	`pragma`
`include`	`do`	`cout`
`int`	`case`	`cin`
`char`	`if`	`union`
`float`	`else`	`short`
`double`	`struct`	`default`
`break`	`return`	`long`
`unsigned`	`bool`	`switch`

- The name must not exceed 256 characters.
- The name is case-sensitive.
- The name must be different from the names of C++ standard functions, such as `printf`, `scanf`, `sin`, `cos` and `atoi`.

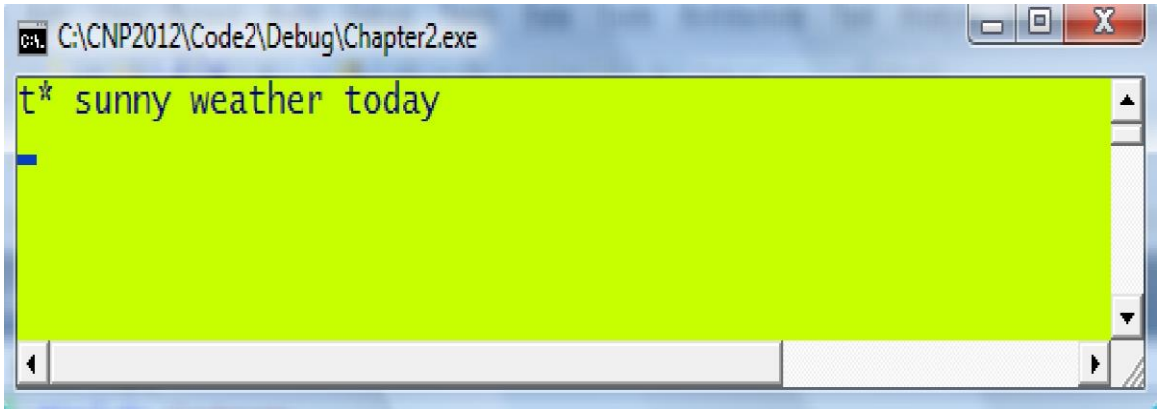
Code 2C.cpp: Variable and Data

```
#include <iostream>
#include <iomanip>

using namespace std;

void main()
{
     double p,q,r;
     p=7.5; q=-3.79;
     cout << "p is " << p << ", q is " << q << endl;
     r=p+q;
     cout << "r is " << r << endl;
     r=p-q;
     cout << "r is " << r << endl;
     r=p*q;
     cout << "r is " << r << endl;
     r=p/q;
     cout << "r is " << r << endl;
     r=-7.059203559263201472 84;
     cout << "r with the default number of decimal
             places is " << r << endl;
     cout << "r with 16 decimal places is "<<
             setprecision(16) << r << endl;
     cin.get();
}
```



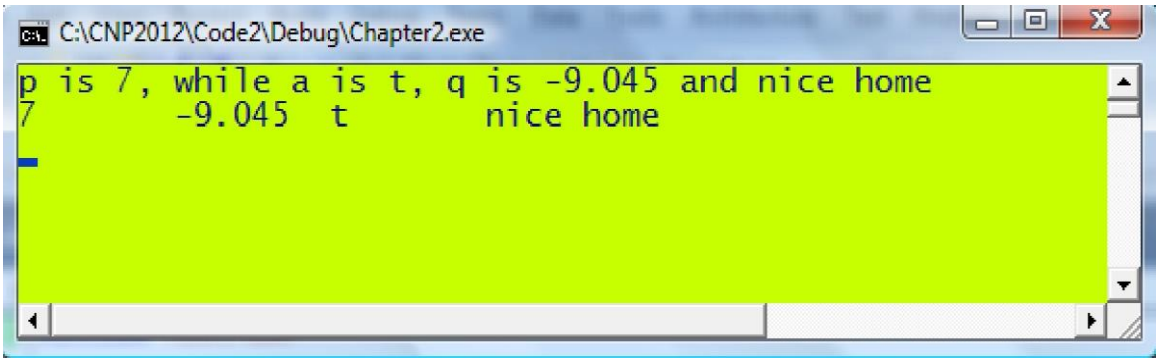
Code 2D.cpp: Character and String

```
#include <iostream>

using namespace std;

void main()
{
     char p,q;
     p='t'; q='*';
     char r[30];
     strcpy(r," sunny weather today");
     cout << p << q << r << endl;
     cin.get();
}
```



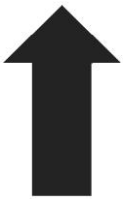
Code 2E.cpp: Multiple data types in cout.

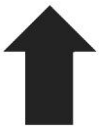
```
#include <iostream>

using namespace std;

void main()
{
     int p=7;
     double q=-9.045;
     char a='t';
     char b[20]="nice  home";
     cout << "p is " << p << ", while a is " << a;
     cout  << ", q is " << q << " and " << b <<
          endl;
     cout << p << "\t"<< q <<"\t"<<a<< "\t" << b << endl;
     cin.get();
}
```



Display using `printf()`

`printf`(*formatted string,variables*)





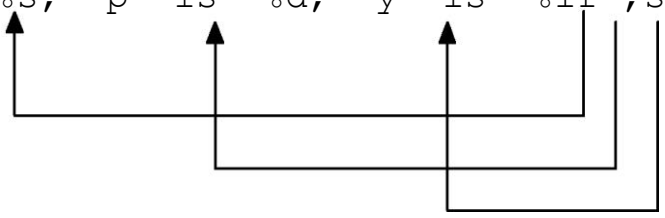
Variables to match the identifiers

The formatted string which includes identifiers for displaying output

Table 2.2. Identifiers in formatting data in printf().

Identifier	*Variable Type/Purpose*
`%c`	Character
`%s`	String
`%d`	Integer
`%x`	Integer in hexadecimal form (small case)
`%X`	Integer in hexadecimal form (upper case)
`%o`	Integer in octal form
`%f`	Floating point
`%lf`	Double floating point
`\n`	New line
`\t`	Tab

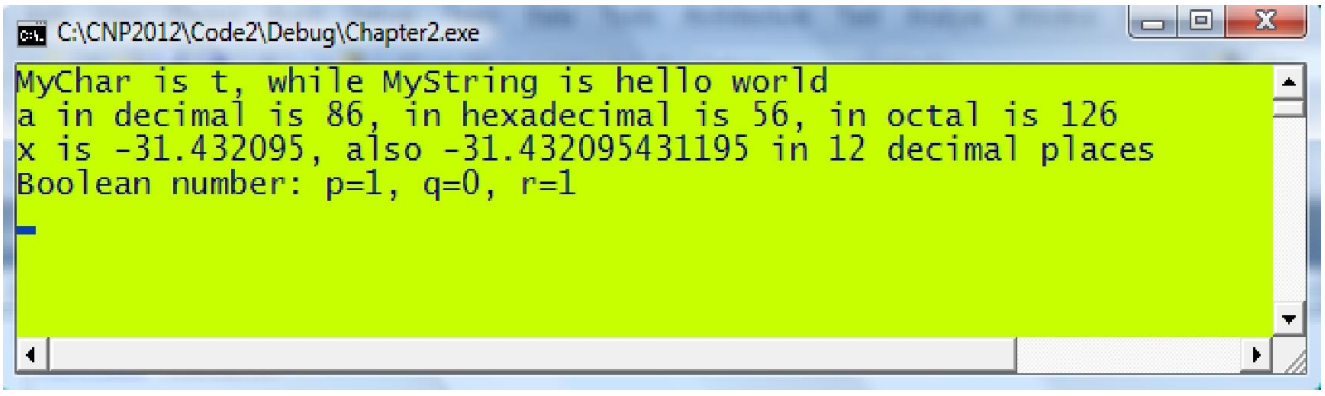
```
char *str="The
results:"; int p=-7;
double y=4.052;
printf("%s,  p  is  %d,  y  is  %lf",str,p,y);
```



```
The results, p is -7, y is 4.052
```

Code2F.cpp: printf() display.

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

void main()
{
      int a=86;
      double
      x=31.4320954311953;
      bool p,q,r;
      char MyChar='t';
      char MyString[]="hello world";
      printf("MyChar is %c,while MyString is %s\n", MyChar,
      MyString);
      printf("a in decimal is %d, in hexadecimal is %x, in octal
            is %o\n",a,a,a);
      printf("x is %lf, also %.12lf in 12 decimal places\n",x,x);
      p=1;  q=0;  r=4;
      printf("Boolean number: p=%d, q=%d, r=%d\n",p,q,r);
      cin.get();
}
```



Typecasting

If a is an `int` and y is a `double` then

```
y=sin(a)
```
will produce an error

Correct way:

```
y=sin((double)a)
```

Mathematical Operators

Table 2.2. Mathematical operators.

Operator	*Meaning*	*Example*
+	Addition	8+5 returns 13.
-	Subtraction	8-5 returns 3.
/	Divide	8/5 returns 1.600000.
*	Multiply	8*5 returns 40.
%	Remainder	8%5 returns 3 which is the remainder when 8 is divided by 5.

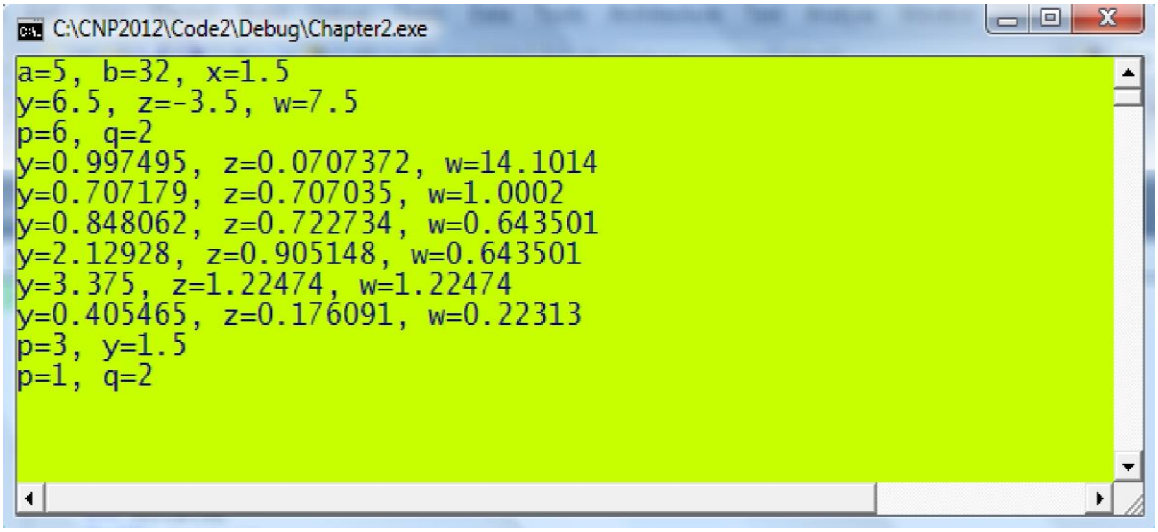
Table 2.3. Built-in mathematical functions in the standard C++ library.

Function	*Description*	*Example*
`sin(`*x*`)`	returns the sine of *x*.	`sin`(2.0) returns 0.909297.
`cos(`*x*`)`	returns the cosine of *x*.	`cos`(2.0) returns 0.416147.
`tan(`*x*`)`	returns the tangent of *x*.	`tan`(2.0) returns -2.185040.
`asin(`*x*`)`	returns the arc sine of *x*, or $\sin^{-1} x$.	`asin`(0.7) returns 0.775397.
`acos(`*x*`)`	returns the arc cosine of *x*, or $\cos^{-1} x$.	`acos`(0.7) returns 0.795398.
`atan(`*x*`)`	returns the arc tangent of *x*, or $\tan^{-1} x$.	`atan`(0.7) returns 0.6107260.
`atoi(`*string*`)`	Converts and returns *string* into an integer.	`atoi("352")` returns 352.
`atof(`*string*`)`	Converts and returns *string* into a double.	`atof("3.52")` returns 3.52.
`abs(`*a*`)`	returns the integer absolute value of *a*, or a.	`abs`(-2) returns 2.
`sinh(`*x*`)`	returns the sine hyperbolic of *x*.	`sinh`(2.0) returns 3.626860.
`cosh(`*x*`)`	returns the cosine hyperbolic of *x*.	`cosh`(2.0) returns 3.762195.
`tanh(`*x*`)`	returns the tangent hyperbolic of *x*.	`tanh(2.0)` returns 0.964028.
`fabs(`*x*`)`	returns the absolute value of *x*, or x.	`fabs(-2.75)` returns 2.75.
`exp(`*x*`)`	returns the exponent of *x* or e^x.	`exp(-2.75)` returns 0.063928.
`log(`*x*`)`	returns the logarithm value of *x*, or $\log x$.	`log(4.0)` returns 0.60206.
`pow(`*x*`,`*y*`)`	returns x^y.	`pow(2,3)` returns 8.
`ceil(`*x*`)`	returns the next integer after *x*.	`ceil`(2.75) returns 3.
`floor(`*x*`)`	returns the previous integer before *x*.	`floor`(2.75) returns 2.

Mathematics	vs.	C++
$\sqrt{x}$		sqrt(x)
x^2		pow(x,2)
$x^{2/5}$		pow(x,2/5)
$\lvert x \rvert$		fabs(x) if *x* is a float or double
$\lvert a \rvert$		abs(a) if *a* is an integer
$\sin x$		sin(x)
$\sin^{-1} x$		asin(x)
$\sinh x$		sinh(x)
e^x		exp(x)
$\log x$		log(x)
$\lfloor a \rfloor$		floor(a)
$\lceil a \rceil$		ceil(a)

Code 2G: Built-in mathematical functions.

```
#define <iostream>
#define PI 3.142

using namespace std;

void main()
{
	int a,b,p,q;
	double x,y,z,w;

	a=5;  b=32;  x=1.5;
	cout <<"a="<< a << ",b="<< b <<", x=" << x << endl;
	y=x+5; z=x-5;  w=x*5;
	cout << "y=" << y << ", z=" << z << ", w=" << w << endl;
	p=b/a; q=b%a;
	cout << "p=" << p << ", q=" << q << endl;
	y=sin(x); z=cos(x);  w=tan(x);
	cout  << "y=" << y << ",  z=" << z << ", w=" << w << endl;
	y=sin(PI/4);  z=cos(PI/4);  w=tan(PI/4);
	cout  <<  "y="  <<  y  <<  ",  z="  <<  z  <<  ",  w="  <<  w  <<  endl;
	y=asin(x/2);  z=acos(x/2);  w=atan(x/2);
	cout  <<  "y="  <<  y  <<  ",  z="  <<  z  <<  ",  w="  <<  w  <<  endl;
	y=sinh(x);  z=cosh(x);  z=tanh(x);
	cout  <<  "y="  <<  y  <<  ",  z="  <<  z  <<  ",  w="  <<  w  <<  endl;
	y=pow(x,3);  z=pow(x,0.5);  w=sqrt(x);
	cout  <<  "y="  <<  y  <<  ",  z="  <<  z  <<  ",  w="  <<  w  <<  endl;
	y=log(x);  z=log10(x);  w=exp(-x);
	cout  <<  "y="  <<  y  <<  ",  z="  <<  z  <<  ",  w="  <<  w  <<  endl;
	p=abs(-3);  y=fabs(-x);
	cout  <<  "p="  <<  p  <<  ",  y="  <<  y  <<  endl;
	p=floor(x);  q=ceil(x);
	cout  <<  "p="  <<  p  <<  ",  q="  <<  q  <<  endl;
	cin.get();
}
```



User-Defined Function

The priority goes from highest to lowest in the order according to:

- Parentheses
- Function
- Index
- * and /
- + and –

For example, `(1-3*x*sin(x+y))/(pow(x,2)+3*exp(x))` is an input string that represents

$$\frac{1-3x\sin(x+y)}{x^2+3e^x}$$

Mathematical equations can be created as functions in C++ using the `#define` directive in the pre-processing area of the program. For example, the equation given by

$$\frac{1-2(2x-1)}{1+3x^2}$$

is created through `#define f(x) (1-(2*x-1)/(1+3*x*x))`

From this definition, `f(x)` is a global function that can be used anywhere inside the

program. For example, `f(0)` returns 2 while `f(-1)`returns 1.75. Similarly, a 2-variable equation such as

$$g(x,y)=\frac{1+xy}{1+3xy}$$

is defined as

```
#define  g(x,y)  ((1-x*y)/(1+3*x*y))
```

Again, in order not to confuse the compiler it is important to provide typecasting so that the function type is specified correctly. For example,

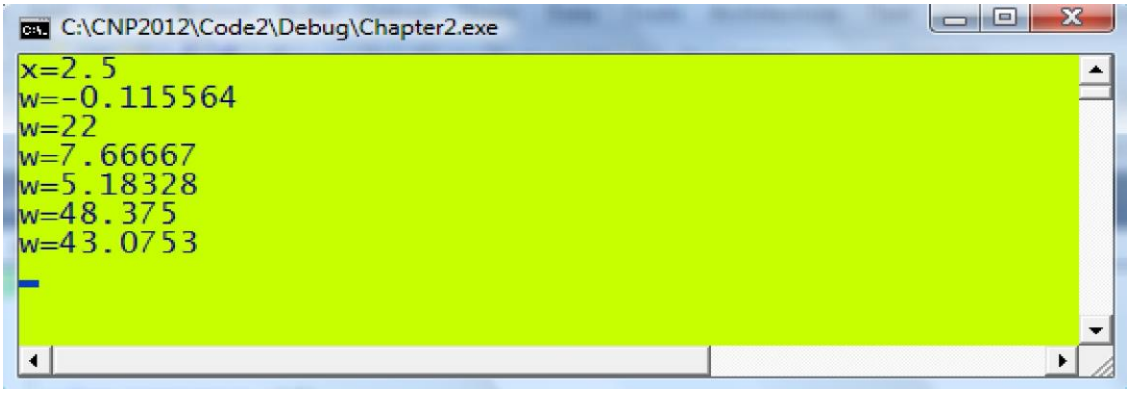
```
#define  f(x)  (1+2*x)
```

will cause `f(x)` to be declared as an `int` automatically. Any further extension of the program which pits `f(x)` into another function of type `double` will definitely create an error because of the mismatched data types. Therefore, the safe way of writing is

```
#define  f(x)  ((double)(1+2*x))
```

Code 2H: User-defined functions

```
#define <iostream>
#define f(s) ((double)1-2*s+3*s*s)
#define g(u,v) (1+5*sin(v*u))

using namespace std;

void main()
{
      double  x=2.5,w;
      cout << "x=" << x << endl;
      w=(1-3*sin(x))/(2+5*cos(3*x-1));
      cout << "w=" << w << endl;
      w=f(3.0);
      cout << "w=" << w << endl;
      w=(1+f(3.0))/(3*f(0.0));
      cout << "w=" << w << endl;
      w=g(7.0,3.0);
      cout << "w=" << w << endl;
      w=1+3*f(x)-2*pow(x,2.0)+pow(x,3.0);
      cout << "w=" << w << endl;
      w=3*f(x)-(1+cos(w)/(3*exp(x))+log(x)/5);
      cout << "w=" << w << endl;
      cin.get();
}
```



String Conversion to `int` and `double`

atoi()	Converts a string into an integer.
return type	`int`
Arguments	*`string`*
Prototype	`iostream.h`

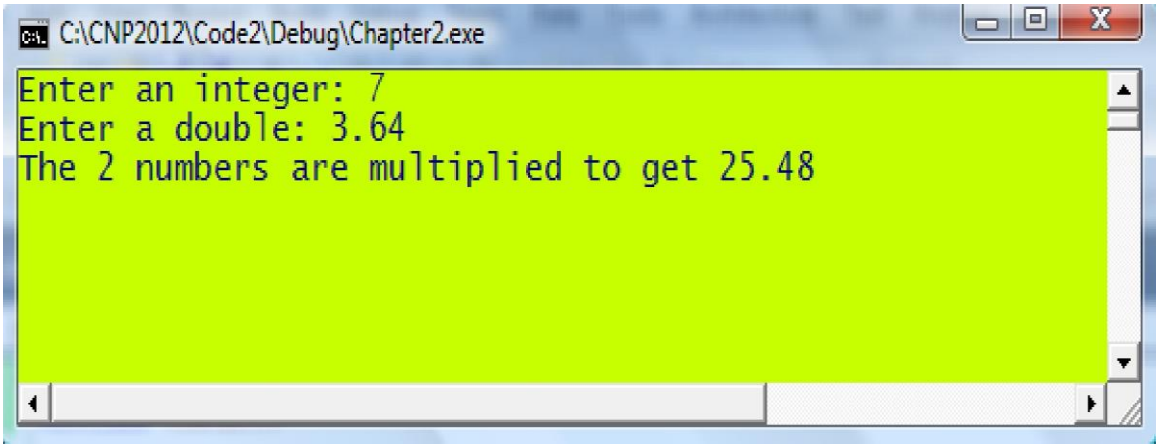
Example	`int w;`
	`w=atoi("-45");`
	// assigns w with the integer value of -45

atof()	Converts a string into a float or double.
Return type	`double/float`
Arguments	*`string`*
Prototype	`iostream.h`

Example	`double t;`
	`t=atof("-5.067");`
	`// assigns t with the double value of -5.067.`

Code2I.cpp: String conversions.

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

void main()
{
      int p; double q,r;
      char str[10];
      cout << "Enter an integer:”;
      cin >> str;
      p=atoi(str);
      cout << "Enter a double: ";
      cin >> str;
      q=atof(str); r=(double)p*q;
      cout << "The 2 numbers are multiplied to get "
           << r << endl; cin.get();
}
```



Low-level Arithmetic

Table 2.4. Integer representation from 0 to 15.

Decimal	*Binary*	*Octal*	*Hexadecimal*
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

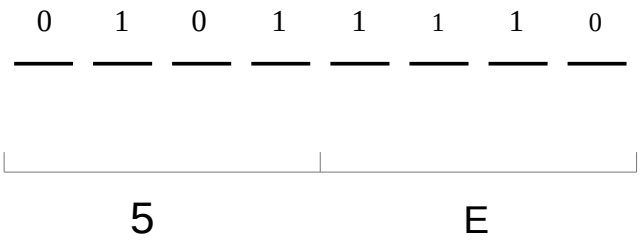


Figure 2.9. Binary representation.

Table 2.5. Mathematical operators for bit-level arithmetic.

Operator	*Meaning*	*Example with p=5E and q=D5 (in hexadecimals)*
|	OR	p | q returns DF.
&	AND	p & q returns 54.
^	XOR	p ^ q returns 8B.
>>	Right shift	p >> 3 returns 0B.
<<	Left shift	p << 3 returns F0.

Table 2.6. AND, OR and XOR tables.

AND	0	1
0	0	0
1	0	1

OR	0	1
0	0	1
1	1	1

XOR	0	1
0	0	1
1	1	0

The AND, OR and XOR operations are illustrated in Figure 2.9 through an example using the hexadecimal numbers 5E and D5 as their inputs.

Figure 2.9. The AND, OR and XOR operations on 5E and D5.

Bit Shifting

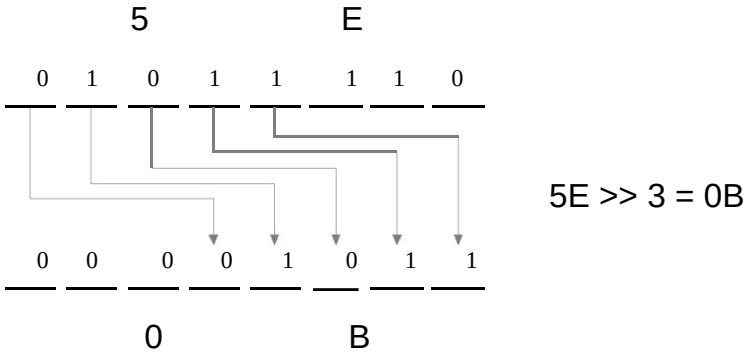


Figure 2.10. Right shifting on 5E.

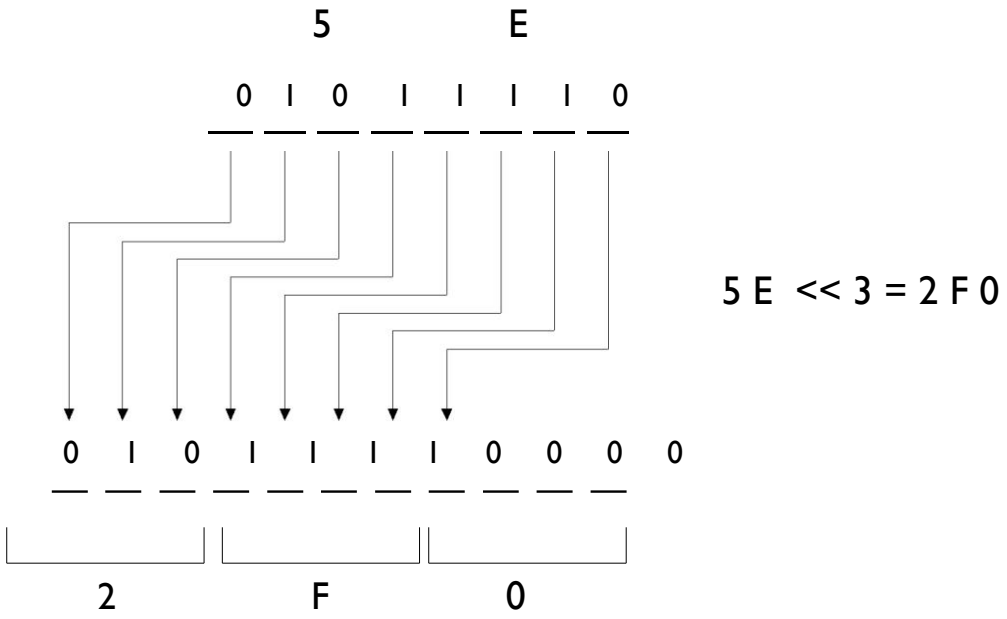
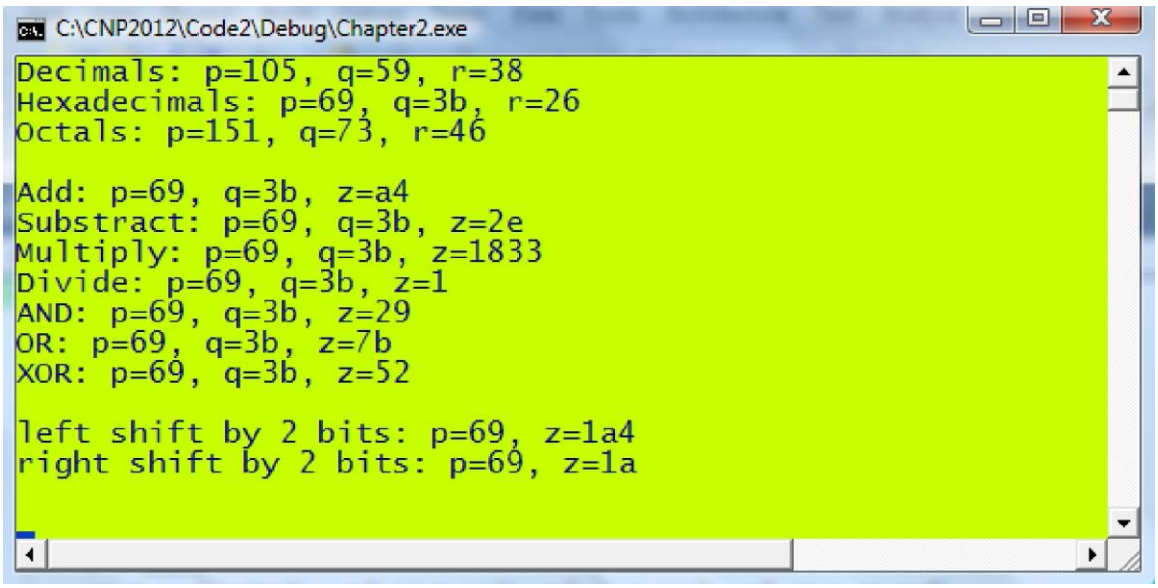


Figure 2.11. Left shifting on 5E.

Code 2J.cpp: Low level operations.

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

using namespace std;

void main()
{
	int p,q,r,z;
	p = 105;
	q = 0x3B;
	r = 0X26;
	cout << "Decimals: p=" << p << ", q=" << q << ", r=" << r << endl;
	printf("Hexadecimals: p=%x, q=%x, r=%x\n", p,q,r);
	printf("Octals: p=%o, q=%o, r=%o\n\n", p,q,r);
	z  =  p+q;
	printf("Add:  p=%x,  q=%x,  z=%x\n",  p,q,z);
	z = p-q;
	printf("Substract: p=%x, q=%x, z=%x\n", p,q,z);
	z = p*q;
	printf("Multiply: p=%x, q=%x, z=%x\n", p,q,z);
	z = p/q;
	printf("Divide: p=%x, q=%x, z=%x\n", p,q,z);
	z = p & q;
	printf("AND: p=%x, q=%x, z=%x\n" ,p,q,z);
	z = p | q;
	printf("OR: p=%x, q=%x, z=%x\n", p,q,z);
	z  =  p  ^  q;
	printf("XOR:  p=%x,  q=%x,  z=%x\n\n",  p,q,z);
	z  =  p  <<  2;
	printf("left  shift  by  2  bits:  p=%x,  z=%x\n",  p,z);
	z = p >> 2;
	printf("right shift by 2 bits: p=%x, z=%x\n\n", p,z);
	getch();
}
```



MAIN REFERENCE:

Shaharuddin Salleh (2012), C++ Numerical Programming.