- Characters and Integers
- Floating-Point Variables
- Arithmetic Operators
- Relational and Logical Operators and the Assignment Operator
- Real World Application: Statistical Measures

- The for Statement and the Comma Operator
- The Operators ++ and --
- Real World Application: Printing a Bar Graph

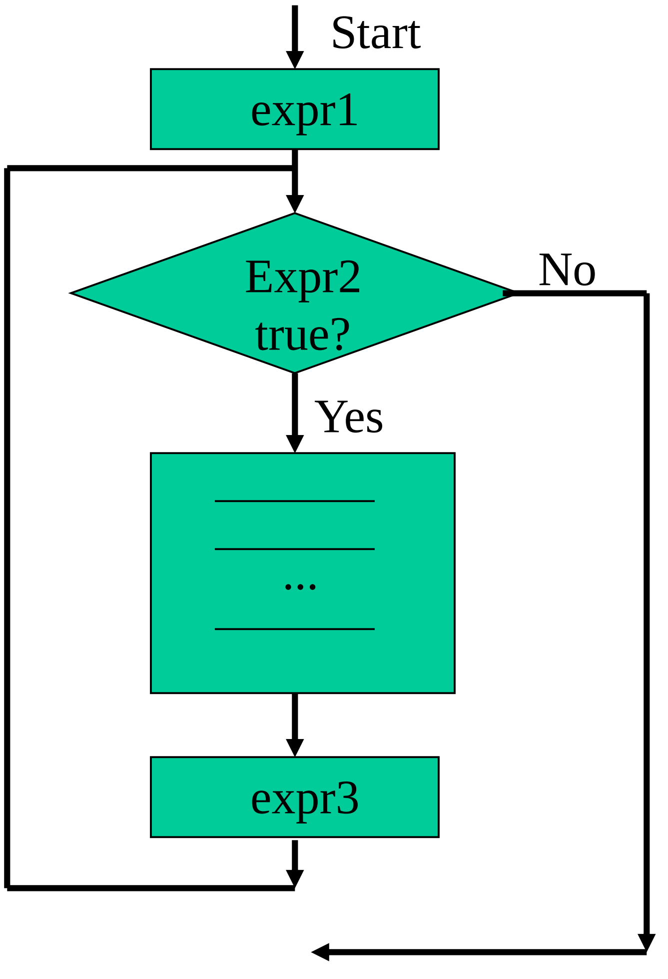
The for Statement and the Comma Operator

```
for (expr1; expr2; expr3)
        action
```

⇕

```
expr1;
while ( expr2 ) {
    action
    expr3;
}
```

```
#include <stdio.h>
main ( ) {
    int i, sum = 4;
    for ( i = 1; i <= 3; i++ )
        sum += i;
    printf ( “sum = %d\n”, sum);
}
```



Loop 1: i = 1, i <= 3, sum = sum + i = 4 + 1 = 5
Loop 2: i = 2, i <= 3, sum = sum + i = 5 + 2 = 7
Loop 3: i = 3, i <= 3, sum = sum + i = 7 + 3 = 10
The output is
sum = 10

```
#include <stdio.h>
main ( ) {
        float x;
        for ( x = 0; x <= 1.0; x += 0.1 )
                printf ( “%3.1f  “, x );
}
```

in VC++, the output is
0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9
where is the missing 1.0?
In this system, it results from changing between decimal and binary.

- It is possible to have multiple initializations in a for loop. The individual assignments are separated by commas.

```
int I, sum;
for ( I = 0, sum = 0; I < 5; I++ )
        sum += I;
printf ( "sum = %d\n", sum );
```

I=0
sum=0
0<5 true
sum =0+0=0
I=1

1<5 true
sum=0+1=1
I=2

2<5 true
sum=1+2=3
I=3

3<5 true
sum+3+3=6
I=4

4<5 true
sum=6+4=10
I=5

5<5 false
print sum= 10

• Any of the cxpr1, expr2, expr3 may be missing. But the two semicolons must always be presented.

```
for ( I = 1; I < 5; ) {
        sum += I;
        I++;
}
```

```
for ( I = 1; I < 5; I++)
        sum += I;
```

If expr2 is missing, the condition is “true”

```
for ( ; ; )
        action
```

is an infinite loop.

Equivalent while and for

```
for (expr1;expr2;expr3)
    action
```

```
expr1;
while (expr2){
    action
    expr3;
}
```

```
for( ;expr; )
    action
```

```
while(expr)
    action
```

↓The Operators ++ and --

++	adds 1
--	subtracts 1

The post-increment operator: when x++; is executed

- The value of the expression x++ is equal to the original value of x.
- The value of the variable x is increased by 1;

The pre-increment operator: when ++x; is executed

- The value of the variable x is increased by 1;
- The value of the expression ++x is 1 more than the original value of x.

The post-decrement x--

- The value of the expression x-- is equal to the original value of x.
- The value of the variable x is decreased by 1;

The predecrement operator --x

- The value of the variable x is decreased by 1;
- The value of the expression --x is 1 less than the original value of x.

```
#include <stdio.h>
main ( ) {
        int x = 5, y;
        y = x++;
        printf ( “x = %d, y = %d\t”, x, y ); /* y = 5, x = 6 */

        y = ++x;
        printf ( “x = %d, y = %d\n”, x, y); /* y = 7, x = 7 */

        y = x--;
        printf ( “x = %d, y = %d\t”, x, y ); /* y = 7, x = 6 * /

        y = --x;
        printf ( “x = %d, y = %d\n”, x, y ); /* y = 5, x = 5 */
```

```
y=5; x=5;

y += x++;
printf ( “x = %d, y = %d\t”, x, y ); /* x = 6, y = y + x++ = 5 + 5 = 10 */

y += ++x;
printf ( “x = %d, y = %d\n”, x, y ); /* x = 7, y = y + ++x = 10 + 7 = 17 */

if (--y == 16 && !( x++ <= 7 ) ) /*--y == 16 is true, x++ <= 7 is true */

printf ( “--y == 16 && !( x++ <= 7 ) is true.\n”);
else
printf ( “--y == 16 && !( x++ <= 7 ) is false.\n”);
}
```

The output is

x = 6, y = 5 x = 7, y = 7

x = 6, y = 7 x = 5, y = 5

x = 6, y = 10 x = 7, y = 17

--y == 16 && !(x++ <= 7) is false.

I += I++; /*danger*/

I=I+I++

increase I by 1 and then add I to I

or increase I by I then add 1?

Uncertain result.

Real World Application: Printing a Bar Graph

- ***Problem***

Printing a bar graph that shows the volume of large lakes of the world. The input data are the names of the lakes and the volumes of the lakes in hundreds of cubic miles.

- ***Sample Input/Output***

Input
Baikal
58
Superior
54
Tanganyika
45
Nyasa
38
Michigan
26
Huron
21

Output

Baikal

****

Superior

****

Tanganyika

****

Nyasa

****

Michigan

****

Huron

****

- ***Solution***

Read data from input data file (call graph-in.txt) and then use the number read from data file to do the for loop to print the number of star ‘*’.

- ***C Implementation***

```
#include <stdio.h>
main ( ) {
    char c;
    int i, vol;
    FILE *fin;
    fin = fopen( “graph-in.txt”, “r”);
    if ( fin != NULL ) {
        while ( fscanf (fin,  “  %c”, &c) != EOF ) { /*note the blank %c*/
            do { /* print the name */
                printf( "%c", c );
                fscanf (fin, "%c", &c );
            }while ( c != '\n');
            printf ( "\n");
            vol = 0;
            fscanf ( fin, "%d", &vol );
            //printf ( "vol = %d\n", vol);
            for ( i = 1; i <= vol; i++)
                printf ("*");
            printf ( "\n\n");}}}
```

while (fscanf (fin, “ %c”, &c) != EOF)

- Difference between “%c” and “ %c”
- “%c”: reads a character including \n
- “ %c”: skips white space including blanks, tabs, \n until meets the character other than white space.

fscanf()

- fscanf( ) – return EOF if it encounters end of file before any conversion; otherwise, it return the number of successful conversions that were stored.

```
int rval;
char c;
float f1;
rval = fscanf(fin, " %c%f", &c, &f1 );
while( rval != EOF) {
        if ( rval == 2 ) {
                …
        }
        rval = fscanf(fin, " %c%f", &c, &f1 );
}
```

```
        }
    }
}
Method 2:
#include <stdio.h>
main ( ) {
    char name[ 20 ];     /*store name of the lake */
    int vol;             /*volume in hundreds of cubic miles of lake */
    int I;
    FILE *fin, *fout;
    fin = fopen ( “graph-in.txt”, “r”);
    fout = fopen (“graph-out.txt”, “w”);
    if ( fin != NULL ) {
        while ( fscanf ( fin, "%s%d", name, &vol ) != EOF ) {
            fprintf ( fout, "%s\n", name);
            for ( i = 1; i <= vol; i++)
                fprintf (fout, "*");
            fprintf ( fout, “\n\n”);
        }
```

```
}
else
      printf ("No input file exists");
}
```

♦ Common Programming Errors

• Place one character between single quotation marks except for the specially denoted characters ‘\n’, ‘\t’. For example,

```
char c = ‘0’;  /* right */
c = ‘a’;   /* right */
```

‘abcd’ is wrong.

• It is a logical error to mismatch a format descriptor and its corresponding argument in scanf. For example, it is an error to write

```
int i;
scanf (“%c”, &i );      /* Logical error */
```

• The logical and operator is && rather than &. The unary operator & is the address operator, and the binary operator & is the bitwise and operator.

• It is illegal to write x % y if either x or y is float, double, or long double. The operands of % must be integers.

• The three expressions in a for statement are separated by semicolons, not commas or some other symbol. It is an error to write

```
for (expression1, expression2, expression3) {  /* Error */
}
```

• Do not place a semicolon between for (-)and the body of the for loop. The code

```
for (I = 0; I < 10; I++);
        printf ( "I = %d\n", I );
```

is syntactically correct but logically erroneous, assuming that the programmer intends the statement

```
printf ("I = %d\n", I );
```

to be the body of the for loop.