“Operators, Expressions, and statements”

Using Bloodshed Dev-C++

Heejin Park

Hanyang University

Introduction

- Introducing Loops
- Some Additional Operators
- Expressions and Statements
- Type Conversions and Type casts
- Function with Arguments
- A Sample Program
- Summary

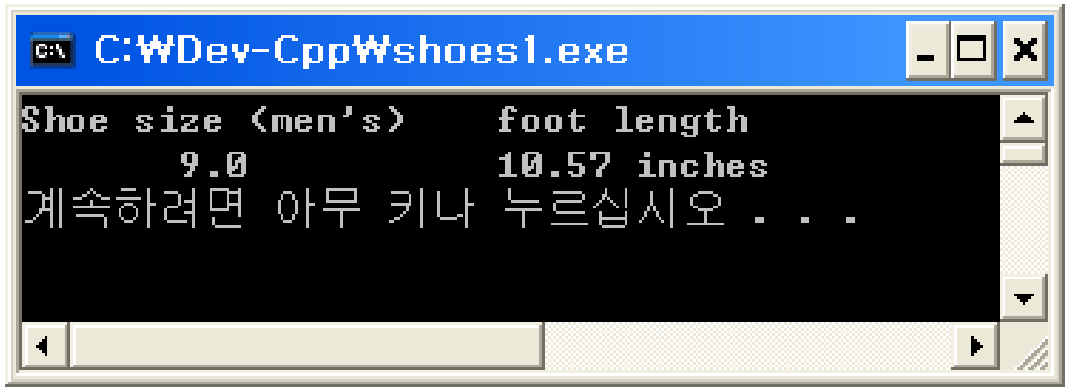
Introducing Loops

The shoes1.c Program

```
#include <stdio.h>
#include <stdlib.h>
#define ADJUST 7.64
#define SCALE 0.325

int main(void){

    double shoe, foot;

    shoe = 9.0;
    foot = SCALE * shoe + ADJUST;

    printf("Shoe size (men's)    foot length\n");
    printf("%10.1f %15.2f inches\n", shoe, foot);

    system("pause");
    return 0;
}
```

Introducing Loops

The shoes1.c Program



Introducing Loops

The shoes2.c Program

```
#include <stdio.h>
#include <stdlib.h>
#define ADJUST 7.64
#define SCALE 0.325

int main(void){

    double shoe, foot;
    printf("Shoe size (men's)    foot length\n");

    shoe = 3.0;

    while (shoe < 18.5){       /* starting the while loop */

        foot = SCALE*shoe + ADJUST; /* start of block */
        printf("%10.1f %15.2f inches\n", shoe, foot);
        shoe = shoe + 1.0;

    }                          /* end of block            */
    printf("If the shoe fits, wear it.\n");

    system("pause");
    return 0;
}
```

Introducing Loops

The shoes2.c Program

```
C:₩Dev-Cpp₩shoes2.exe
Shoe size (men's)    foot length
       3.0            8.62 inches
       4.0            8.94 inches
       5.0            9.27 inches
       6.0            9.59 inches
       7.0            9.91 inches
       8.0           10.24 inches
       9.0           10.57 inches
      10.0           10.89 inches
      11.0           11.22 inches
      12.0           11.54 inches
      13.0           11.87 inches
      14.0           12.19 inches
      15.0           12.52 inches
      16.0           12.84 inches
      17.0           13.16 inches
      18.0           13.49 inches
If the shoe fits, wear it.
계속하려면 아무 키나 누르십시오 . . .
```

Fundamental Operators

Operators

Assignment	=
Addition	+
Subtraction	-
Sign	- and +
Multiplication	*
Division	/

Fundamental Operators

Some Terminology

- **Data Objects**
- a region of data storage that can be used to hold values.
- **Lvalues**
- a name or expression that identifies a particular Data object.
  - A=3; // A is the lvalue
- **Rvalues**
- quantities that can be assigned to modifiable Lvalues.
  - A=3; // 3 is the R value
- **Operands**
- Operands are what operators operate on.

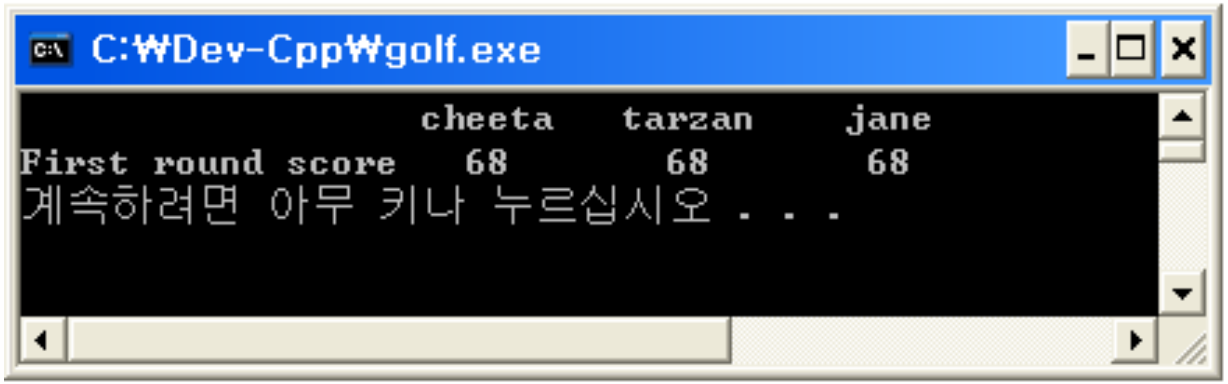
Fundamental Operators

The golf.c Program

```
#include <stdio.h>
#include <stdlib.h>

int main(void){

    int jane, tarzan, cheeta;

    cheeta = tarzan = jane = 68;
    printf("                    cheeta   tarzan    jane\n");
    printf("First round score %4d %8d %8d\n",cheeta,tarzan,jane);

    system("pause");
    return 0;
}
```

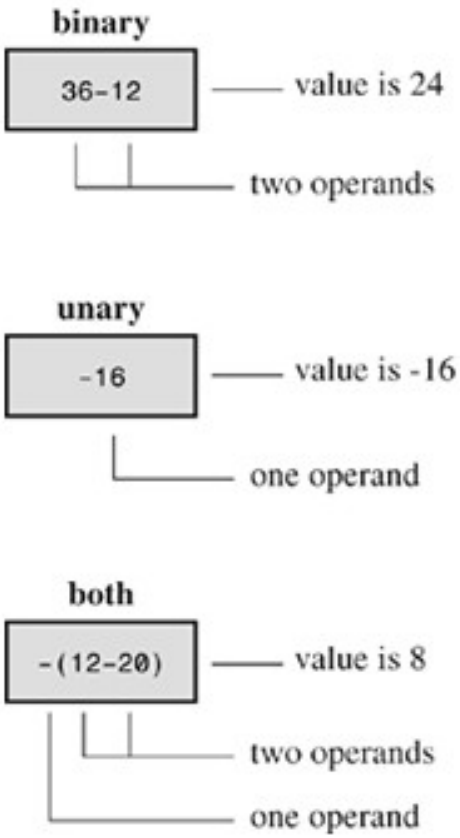
Fundamental Operators

The golf.c Program



Fundamental Operators

- Unary and binary operators



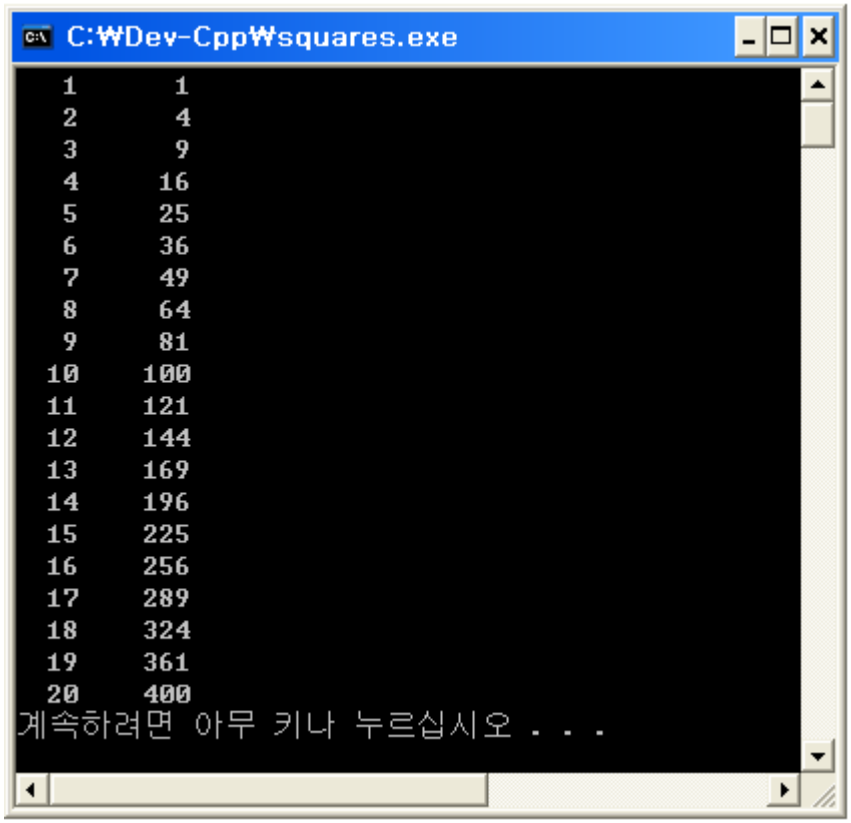
Fundamental Operators

The squares.c Program

```
#include <stdio.h>
#include <stdlib.h>

int main(void){

    int num = 1;
    while (num < 21){

        printf("%4d %6d\n", num, num * num);
        num = num + 1;
    }

    system("pause");
    return 0;
}
```

Fundamental Operators

The squares.c Program



Fundamental Operators

The wheat.c Program(1/2)

```
#include <stdio.h>
#include <stdlib.h>
#define SQUARES 64     /* squares on a checkerboard   */
#define CROP 1E15      /* US wheat crop in grains     */

int main(void){

    double current, total;
    int count = 1;

    printf("square     grains       total     ");
    printf("fraction of \n");
    printf("           added        grains      ");
    printf("US total\n");

    total = current = 1.0; /* start with one grain   */
    printf("%4d %13.2e %12.2e %12.2e\n", count, current,
           total, total/CROP);
```

Fundamental Operators

The wheat.c Program(2/2)

```
    while (count < SQUARES){

        count = count + 1;
        current = 2.0 * current; /* double grains on next square */
        total = total + current;      /* update total */
        printf("%4d %13.2e %12.2e %12.2e\n", count, current,
        total, total/CROP);
    }

    printf("That's all.\n");

    system("pause");
    return 0;
}
```

Fundamental Operators

The wheat.c Program

- **Exponential Growth**

C:₩Dev-Cpp₩wheat.exe

square	grains added	total grains	fraction of US total
1	1.00e+000	1.00e+000	1.00e-015
2	2.00e+000	3.00e+000	3.00e-015
3	4.00e+000	7.00e+000	7.00e-015
4	8.00e+000	1.50e+001	1.50e-014
5	1.60e+001	3.10e+001	3.10e-014
6	3.20e+001	6.30e+001	6.30e-014
7	6.40e+001	1.27e+002	1.27e-013
8	1.28e+002	2.55e+002	2.55e-013
9	2.56e+002	5.11e+002	5.11e-013
10	5.12e+002	1.02e+003	1.02e-012
11	1.02e+003	2.05e+003	2.05e-012
12	2.05e+003	4.10e+003	4.09e-012
13	4.10e+003	8.19e+003	8.19e-012
14	8.19e+003	1.64e+004	1.64e-011
15	1.64e+004	3.28e+004	3.28e-011

…Run until square value is 64

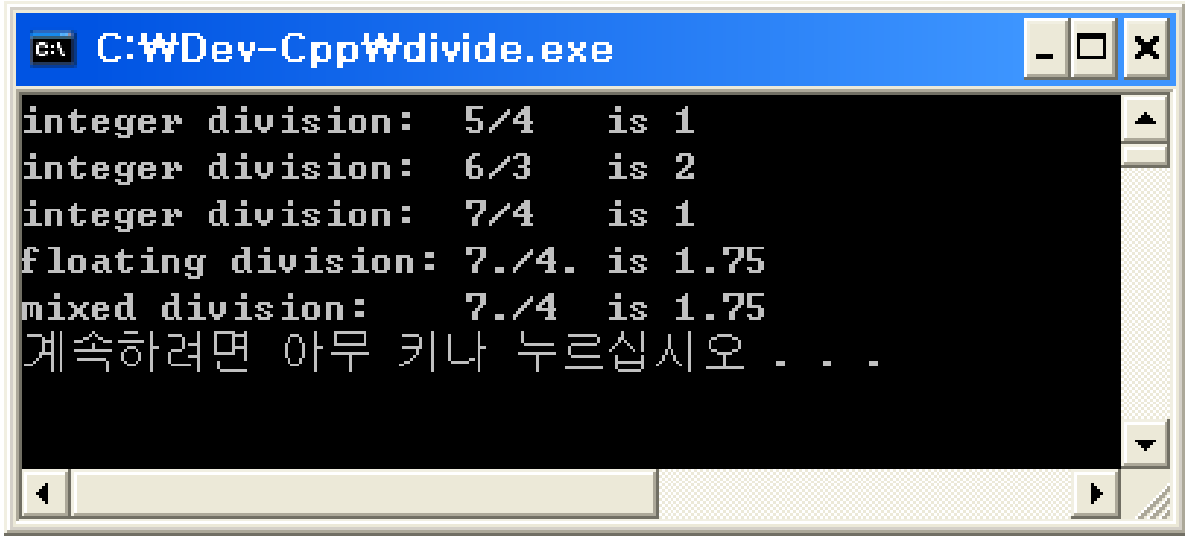
Fundamental Operators

The divide.c Program

```
#include <stdio.h>
#include <stdlib.h>

int main(void){

     printf("integer division:  5/4   is %d \n", 5/4);
     printf("integer division:  6/3   is %d \n", 6/3);
     printf("integer division:  7/4   is %d \n", 7/4);
     printf("floating division: 7./4. is %1.2f \n", 7./4.);
     printf("mixed division:    7./4  is %1.2f \n", 7./4);

     system("pause");
     return 0;
}
```

Fundamental Operators

The divide.c Program

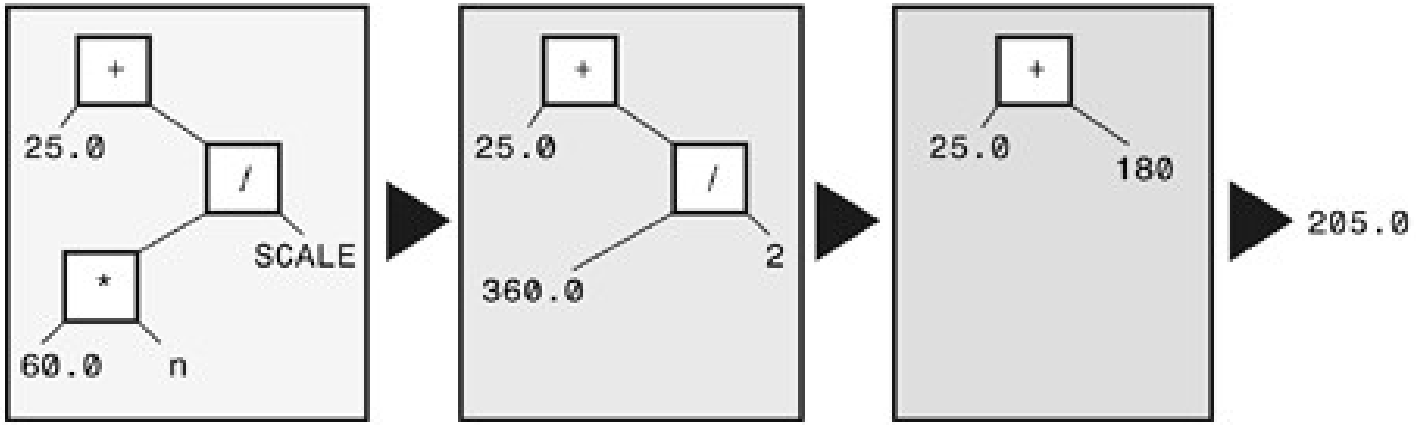


Fundamental Operators

Operator Precedence

- Expression trees showing operators, operands, and order of evaluation.

```
SCALE=2;
n=6;
butter=25.0+60.0*n/ SCALE;
```



Fundamental Operators

Operator Precedence

- Operators in Order of Decreasing Precedence

Operator	Associativity
()	Left to right
+ - (unary)	Right to left
*** /**	Left to right
+ - (binary)	Left to right
=	Right to left

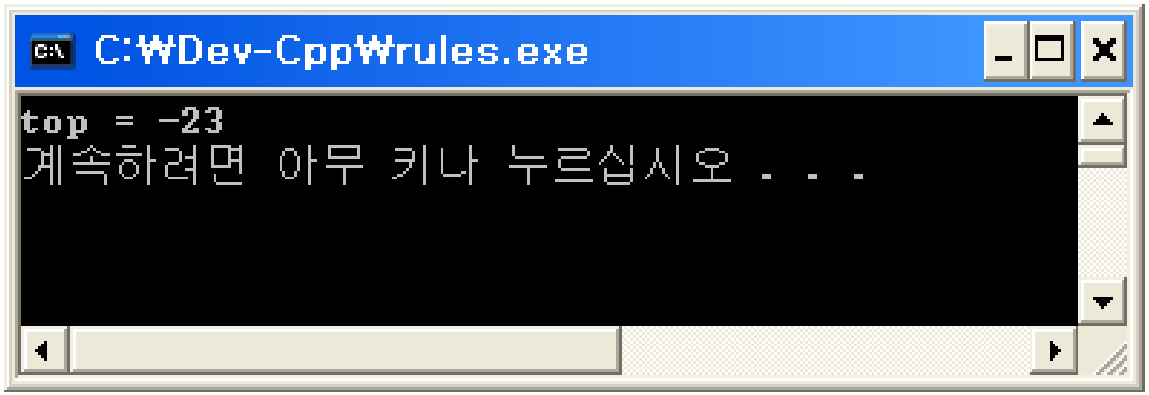
Fundamental Operators

The rules.c Program

```
#include <stdio.h>
#include <stdlib.h>

int main(void){

    int top, score;

    top = score = -(2 + 5) * 6 + (4 + 3 * (2 + 3));

    printf("top = %d \n", top);

    system("pause");
    return 0;
}
```

Fundamental Operators

The rules.c Program



Some Additional Operators

The `sizeof` Operator and the `size_t` Type

- **`sizeof`** returns a value of type **`size_t`**
- Operator returns the size, in bytes, of its operand.
- The operand can be a specific data object or it can be a type.

- **Additional Operators**

Modulus	**%**
Increment and Decrement	**++ and --**

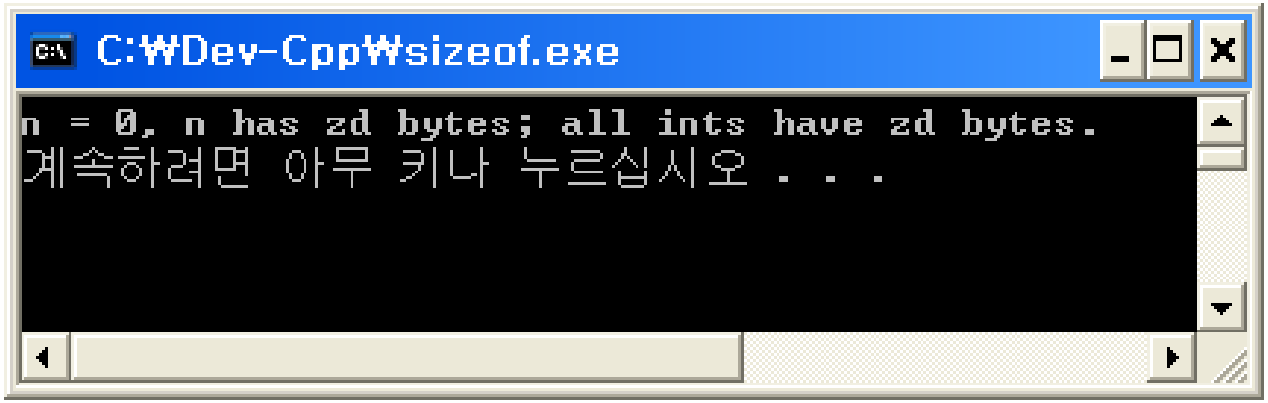
Some Additional Operators

The sizeof.c Program

```
#include <stdio.h>
#include <stdlib.h>

// uses C99 %z modifier -- try %u or %lu if you lack %zd

int main(void){

    int n = 0;
    size_t intsize;

    intsize = sizeof (int);
    printf("n = %d, n has %zd bytes; all ints have %zd bytes.\n", n, sizeof n, intsize );

    system("pause");
    return 0;
}
```

Some Additional Operators

The sizeof.c Program



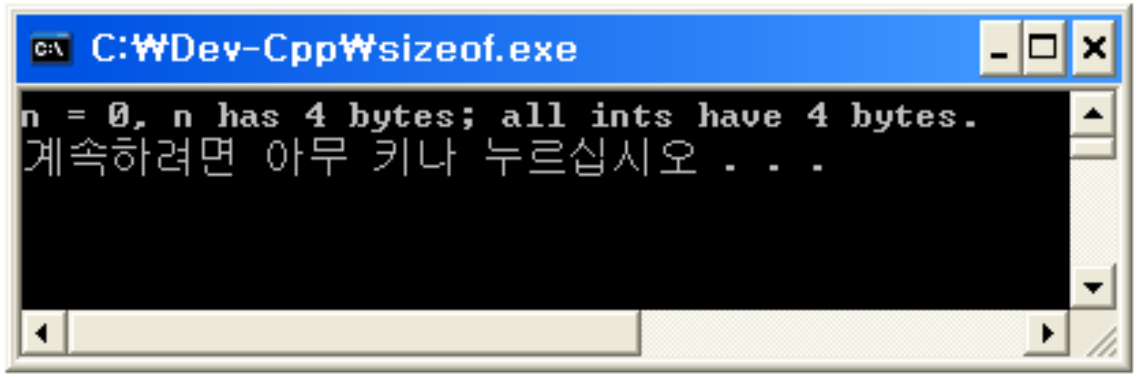
Some Additional Operators

The sizeof.c Program

```
#include <stdio.h>
#include <stdlib.h>

// uses C99 %z modifier -- try %u or %lu if you lack %zd

int main(void){

    int n = 0;
    size_t intsize;

    intsize = sizeof (int);
    printf("n = %u, n has %u bytes; all ints have %u bytes.\n", n, sizeof n, intsize );

    system("pause");
    return 0;
}
```

Some Additional Operators

The sizeof.c Program



Some Additional Operators

Additional Operators

Modulus	%
Increment and Decrement	++ and --

Some Additional Operators

The min_sec.c Program

```
#include <stdio.h>
#include <stdlib.h>
#define SEC_PER_MIN 60            // seconds in a minute

int main(void){

    int sec, min, left;

    printf("Convert seconds to minutes and seconds!\n");
    printf("Enter the number of seconds (<=0 to quit):\n");
    scanf("%d", &sec);            // read number of seconds

    while (sec > 0){

        min = sec / SEC_PER_MIN;  // truncated number of minutes
        left = sec % SEC_PER_MIN; // number of seconds left over

        printf("%d seconds is %d minutes, %d seconds.\n", sec,
                min, left);
        printf("Enter next value (<=0 to quit):\n");
        scanf("%d", &sec);
    }

    printf("Done!\n");

    system("pause");
    return 0;
}
```

Some Additional Operators

The min_sec.c Program

```
C:₩Dev-Cpp₩min_sec.exe
154 seconds is 2 minutes, 34 seconds.
Enter next value (<=0 to quit):
567
567 seconds is 9 minutes, 27 seconds.
Enter next value (<=0 to quit):
0
Done!
계속하려면 아무 키나 누르십시오 . . .
```

Some Additional Operators

The add_one.c Program

```
#include <stdio.h>
#include <stdlib.h>

int main(void){

    int ultra = 0, super = 0;

    while (super < 5){

        super++;
        ++ultra;

        printf("super = %d, ultra = %d \n", super, ultra);
    }

    system("pause");
    return 0;
}
```

Some Additional Operators

The add_one.c Program

```
C:₩Dev-Cpp₩add_one.exe
super = 1, ultra = 1
super = 2, ultra = 2
super = 3, ultra = 3
super = 4, ultra = 4
super = 5, ultra = 5
계속하려면 아무 키나 누르십시오 . . .
```

Some Additional Operators

Through the loop once

while loop

```
shoe = 2.0;
while (++shoe < 18.5)
{
  foot=SCALE*shoe + ADJUST;
  printf("------", shoe, foot);
}
```

1. increment shoe to 3
2. evaluate test (true)
3. do these statements
4. return to beginning of loop

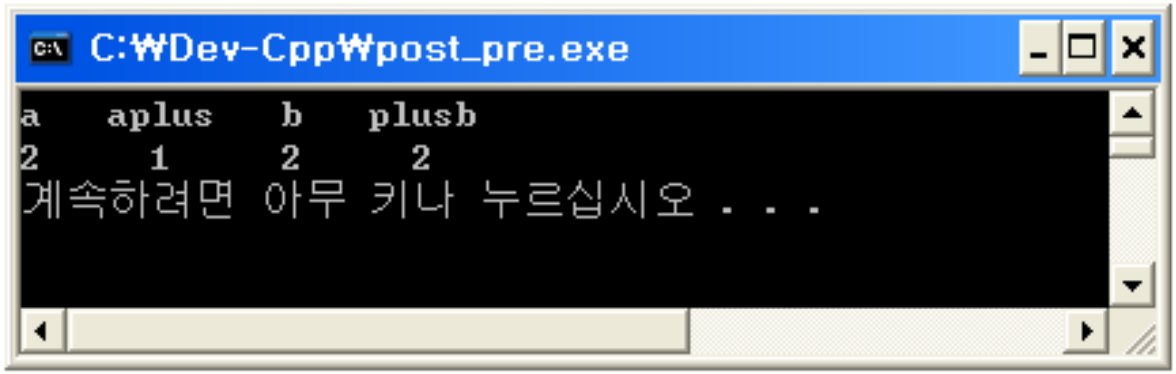
Some Additional Operators

The post_pre.c Program

```
#include <stdio.h>
#include <stdlib.h>

int main(void){

    int a = 1, b = 1;
    int aplus, plusb;

    aplus = a++;        /* postfix */
    plusb = ++b;        /* prefix  */
    printf("a   aplus   b   plusb \n");
    printf("%1d %5d %5d %5d\n", a, aplus, b, plusb);

    system("pause");
    return 0;
}
```

Some Additional Operators

The post_pre.c Program



Some Additional Operators

Prefix and postfix

- **Prefix**

```
q = 2*++a;
```

- Fist, increment a by 1;
- Then, multiply a by 2 and assign to q

- **Postfix**

```
q = 2*a++;
```

- Fist, multiply a 2, assign to q
- then, increment a by 1

Some Additional Operators

The bottles.c Program

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 100

int main(void){

    int count = MAX + 1;
    while (--count > 0) {

        printf("%d bottles of spring water on the wall, "
               "%d bottles of spring water!\n", count, count);
        printf("Take one down and pass it around,\n");
        printf("%d bottles of spring water!\n\n", count - 1);
    }

    system("pause");
    return 0;

}
```

Some Additional Operators

The bottles.c Program

```
C:₩Dev-Cpp₩bottles.exe
Take one down and pass it around,
5 bottles of spring water!

5 bottles of spring water on the wall, 5 bottles of spring water!
Take one down and pass it around,
4 bottles of spring water!

4 bottles of spring water on the wall, 4 bottles of spring water!
Take one down and pass it around,
3 bottles of spring water!

3 bottles of spring water on the wall, 3 bottles of spring water!
Take one down and pass it around,
2 bottles of spring water!

2 bottles of spring water on the wall, 2 bottles of spring water!
Take one down and pass it around,
1 bottles of spring water!

1 bottles of spring water on the wall, 1 bottles of spring water!
Take one down and pass it around,
0 bottles of spring water!

계속하려면 아무 키나 누르십시오 . . .
```

Some Additional Operators

Quiz

- squares.c program

```
while (num < 21){
    printf("%4d %6d\n", num, num * num);
    num = num + 1;
}
```

```
while (num < 21){
    printf("%10d %10d\n", num, num*num++);
}
```

Some Additional Operators

Quiz

- Yet another troublesome case is this:
- Guess the result.

```
int n = 3;
int y;
    printf("y value: %d", y = n++ + n++);
    printf("n value: %d", n);
```

Expressions and Statements

Expressions

- An expression consists of a combination of operators and operands.
- Every expression has a value.

Expression	Value
`-4 + 6`	2
`c = 3 + 8`	11
`5 > 3`	1
`6 + (c = 3 + 8)`	17

Expressions and Statements

Statements

- The primary building blocks of a program.
- In C, statements are indicated by a semicolon at the end.

This is a statement

```
num = 2;
```

This is just an expression

```
num = 2
```

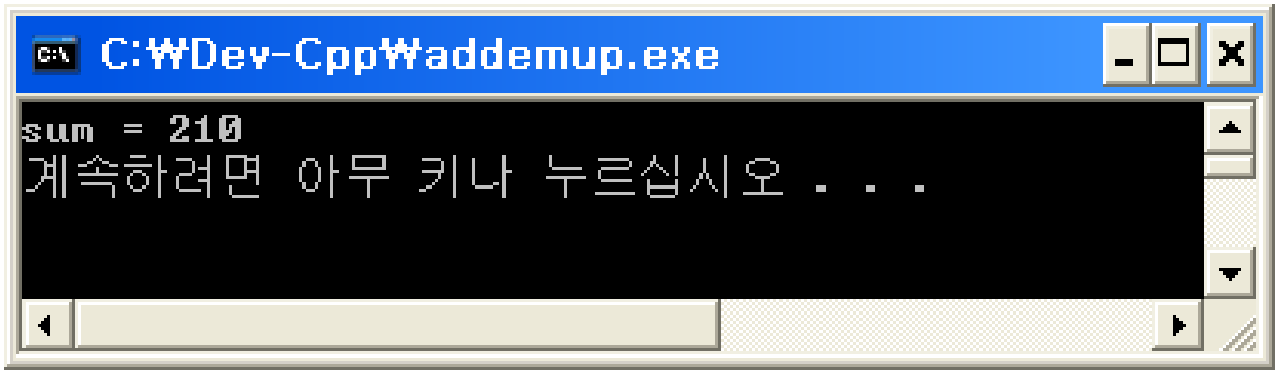
Expressions and Statements

The addemup.c Program

```
#include <stdio.h>
#include <stdlib.h>

int main(void)                    /* finds sum of first 20 integers */
{
    int count, sum;               /* declaration statement          */
    count = 0;                    /* assignment statement           */
    sum = 0;                      /* ditto                          */

    while (count++ < 20)          /* while                          */
        sum = sum + count;        /*     statement                  */
    printf("sum = %d\n", sum);/* function statement                 */

    system("pause");
    return 0;
}
```

Expressions and Statements

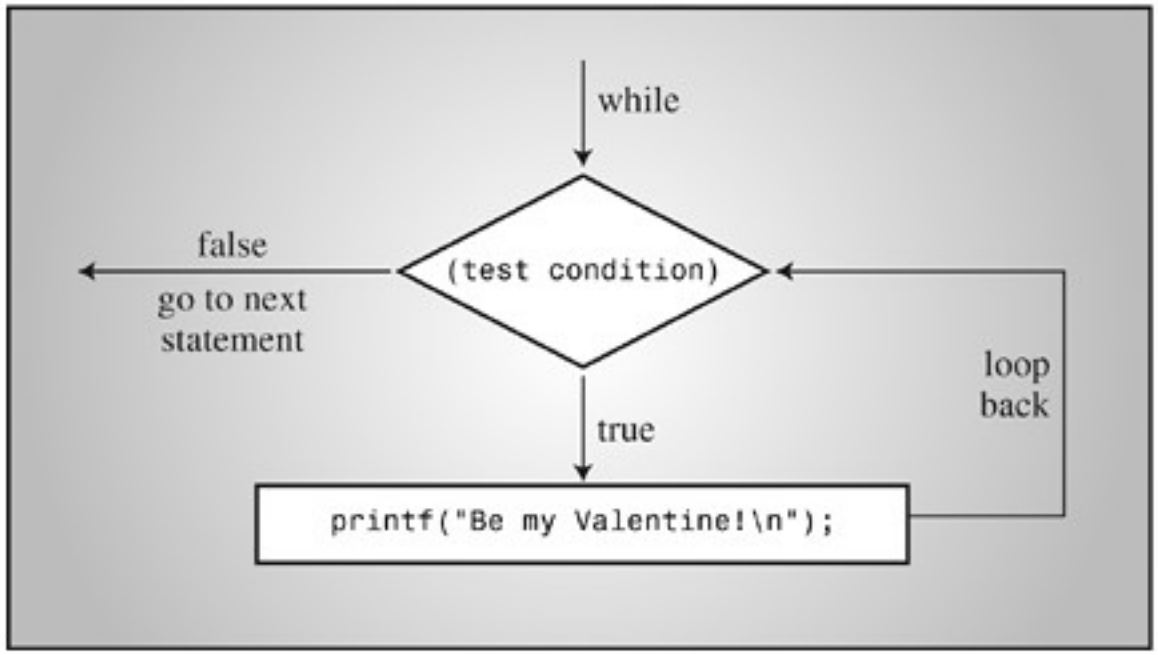
The addemup.c Program



Expressions and Statements

Statements

- Structure of a simple `while` loop



Expressions and Statements

Compound Statements (Blocks)

- Two or more statements grouped together by enclosing them in braces.
- Compare the following program fragments.

```
index = 0;

while (index++ < 10)
sam = 10 * index + 2;
printf("sam = %d\n", sam);
```

```
index = 0;
while (index++ < 10)
{
        sam = 10 * index + 2;
        printf("sam = %d\n", sam);
}
```

- <fragment 1> <fragment 2>
-

Expressions and Statements

Compound Statements (Blocks)

- **Results**

<Fragment1>

```
C:₩Dev-Cpp₩addemup.exe
sam = 102
계속하려면 아무 키나 누르십시오 . . .
```

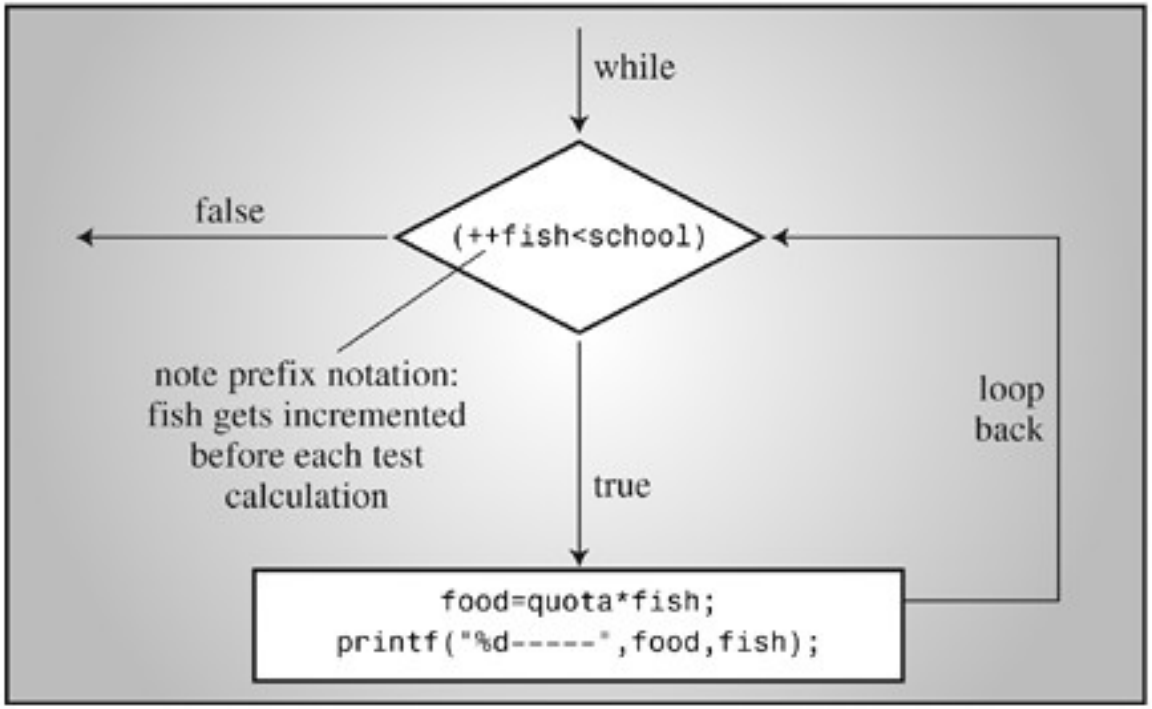
<Fragment2>

```
C:₩Dev-Cpp₩addemup.exe
sam = 12
sam = 22
sam = 32
sam = 42
sam = 52
sam = 62
sam = 72
sam = 82
sam = 92
sam = 102
계속하려면 아무 키나 누르십시오 . . .
```

Expressions and Statements

Statements

- A `while` loop with a compound statement.



Type Conversions and type casts

Automatic type conversions

- When you add values having different data types,
- both values are first **converted to the <u>same type</u>**.

- **Type conversions**
- depend on the specified operator and the type of the operand or operators.

The cast operator

- Explicit type conversions.

```
(type) variable
```

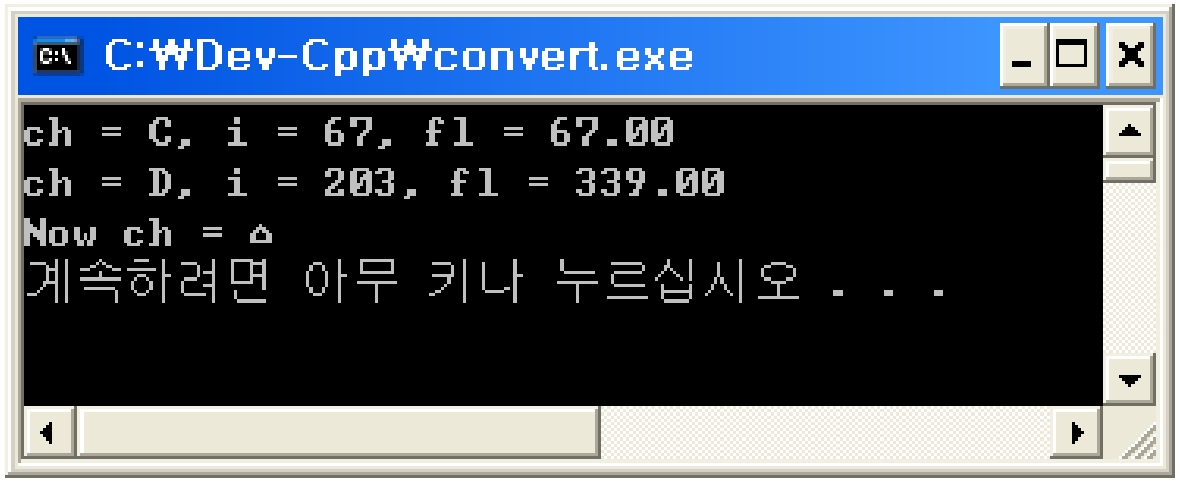
Type Conversions and type casts

The convert.c Program

```
#include <stdio.h>
#include <stdlib.h>

int main(void){

    char ch;
    int i;
    float fl;

    fl = i = ch = 'C';                                  /* line 9  */
    printf("ch = %c, i = %d, fl = %2.2f\n", ch, i, fl); /* line 10 */
    ch = ch + 1;                                        /* line 11 */
    i = fl + 2 * ch;                                    /* line 12 */
    fl = 2.0 * ch + i;                                  /* line 13 */
    printf("ch = %c, i = %d, fl = %2.2f\n", ch, i, fl); /* line 14 */
    ch = 5212205.17;                                    /* line 15 */
    printf("Now ch = %c\n", ch);

    system("pause");
    return 0;
}
```

Type Conversions and type casts

The convert.c Program



Type Conversions and type casts

The Cast Operator

- This is the general form of a cast operator:

```
(type)
```

- Consider the next two code lines, in which mice is an int variable.
- The second line contains two casts to type int.

```
mice = 1.6 + 1.7;

mice = (int) 1.6 + (int) 1.7;
```

Function with Arguments

Using function arguments

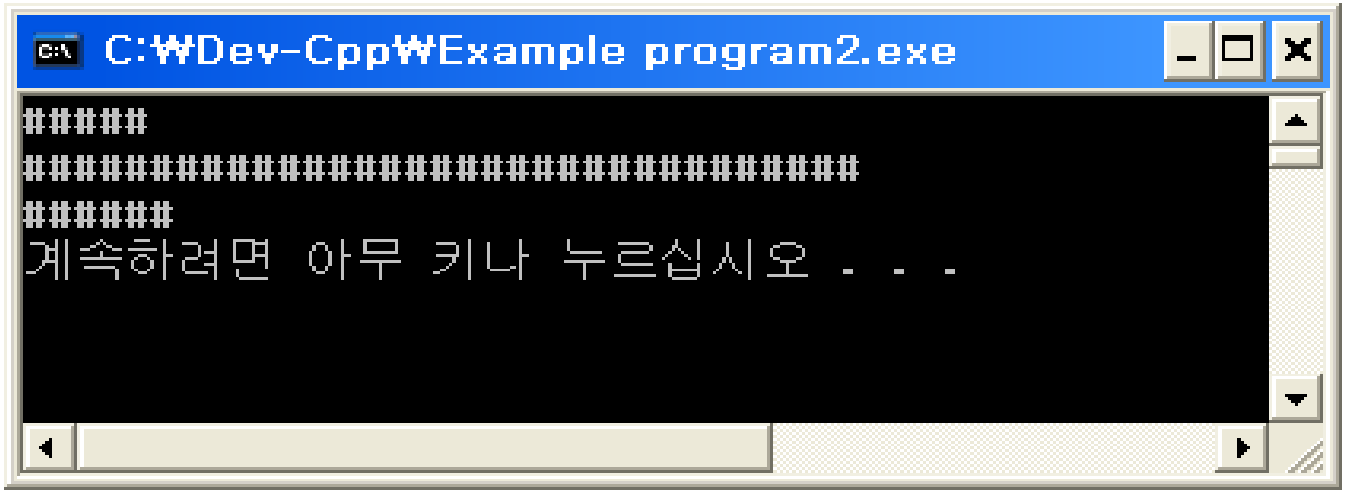
- **Example program**

```
#include <stdio.h>
#include <stdlib.h>

void pound(int n);    /* ANSI prototype              */

int main(void){

    int times = 5;
    char ch = '!';    /* ASCII code is 33            */
    float f = 6.0;

    pound(times);     /* int argument                */
    pound(ch);        /* char automatically -> int   */
    pound((int) f);   /* cast forces f -> int        */

    system("pause");
    return 0;
}
void pound(int n){     /* ANSI-style function header  */

     while (n-- > 0)
           printf("#");

     printf("\n");
}
```

Function with Arguments

Using function arguments

- **Example program**
- **pound( )function:** prints a specified number of pound signs (#).
- also illustrates some points about **type conversion**.



A Sample Program

A Sample Program

- **Example program**

```
C:₩Dev-Cpp₩running.exe
This program converts your time for a metric race
to a time for running a mile and to your average
speed in miles per hour.
Please enter, in kilometers, the distance run.
10.0
Next enter the time in minutes and seconds.
Begin by entering the minutes.
36
Now enter the seconds.
23
You ran 10.00 km (6.21 miles) in 36 min, 23 sec.
That pace corresponds to running a mile in 5 min, 51 sec.
Your average speed was 10.25 mph.
계속하려면 아무 키나 누르십시오 . . .
```

A Sample Program

Using function arguments

- **Source code (1/3)**

```
#include <stdio.h>
#include <stdlib.h>

const int S_PER_M = 60;          // seconds in a minute
const int S_PER_H = 3600;        // seconds in an hour
const double M_PER_K = 0.62137; // miles in a kilometer

int main(void){

    double distk, distm;  // distance run in km and in miles
    double rate;          // average speed in mph
    int min, sec;         // minutes and seconds of running time
    int time;             // running time in seconds only
    double mtime;         // time in seconds for one mile
    int mmin, msec;       // minutes and seconds for one mile
```

A Sample Program

A Sample Program

- **Source code (2/3)**

```
printf("This program converts your time for a metric race\n");
printf("to a time for running a mile and to your average\n");
printf("speed in miles per hour.\n");
printf("Please enter, in kilometers, the distance run.\n");

scanf("%lf", &distk);  // %lf for type double
printf("Next enter the time in minutes and seconds.\n");
printf("Begin by entering the minutes.\n");

scanf("%d", &min);
printf("Now enter the seconds.\n");
scanf("%d", &sec);
```

A Sample Program

A Sample Program

- **Source code (3/3)**

```
// converts time to pure seconds
    time = S_PER_M * min + sec;
// converts kilometers to miles
    distm = M_PER_K * distk;
// miles per sec x sec per hour = mph
    rate = distm / time * S_PER_H;
// time/distance = time per mile
    mtime = (double) time / distm;
    mmin = (int) mtime / S_PER_M; // find whole minutes
    msec = (int) mtime % S_PER_M; // find remaining seconds
    printf("You ran %1.2f km (%1.2f miles) in %d min, %d sec.\n",
         distk, distm, min, sec);
    printf("That pace corresponds to running a mile in %d min, ", mmin);
    printf("%d sec.\nYour average speed was %1.2f mph.\n",msec, rate);

    system("pause");
    return 0;
}
```

Summary(1/4)

Operators

- = - * / % ++ -- (type)
- **Unary operators**
- minus sign and `sizeof`
- **Binary operators**
- addition and the multiplication operators

Expressions

- Combinations of *operators* and *operands*
- **Rules of operator precedence**

Summary(2/4)

Statements

- Complete instructions to the computer and are indicated in C by a terminating semicolon.
- Included within a pair of braces constitute a ***compound statement***, or ***block***.
- **Declaration statements**
- **Assignment statements**
- **Function call statements**
- **Control statements**
  - **`While` loop**

Summary(3/4)

Type Conversions

- **The `char` and `short` types**
- promoted to type `int` whenever they appear in expressions or as function arguments.

- **The `float` type**
- promoted to type `double` when used as a function argument.

- Converted from a larger type to a smaller type
- `long` to `short` or `double` to `float`

- there might be a loss of data.

Summary(4/4)

Define a function

- When you define a function that **takes an argument**
- Declare a ***variable***, or ***formal argument***, in the function definition.
- Then the value passed in a function call
  - assigned to this variable, which can now be used in the function.