MONTANA
STATE UNIVERSITY

Programming with C I

Fangtian Zhong
CSCI 112

Gianforte School of Computing
Norm Asbjornson College of Engineering
E-mail: fangtian.zhong@montana.edu

2025.02.10

Compound assignment

Operator	Definition
+	addition
-	subtraction
*	multiplication
/	division
%	remainder

- **Can do these too:**

  +=

  -=

  ***=**

  /=

  %=

Increment and Decrement Operators

- counter = counter + 1

  count += 1

  counter++

  ++counter

- counter = counter - 1

  count -= 1

  counter--

  --counter

while Statement Syntax

```
while (loop repetition condition)
        statement;

/* display N asterisks. */
count_star = 0;
while (count_star < N) {
        printf(“*”);
        count_star = count_star + 1;
}
```

while Statement Syntax

```
while (loop repetition condition)
        statement;

/* display N asterisks. */
count_star = 0;
while (count_star < N) {
        printf("*");
        count_star += 1;
}
```

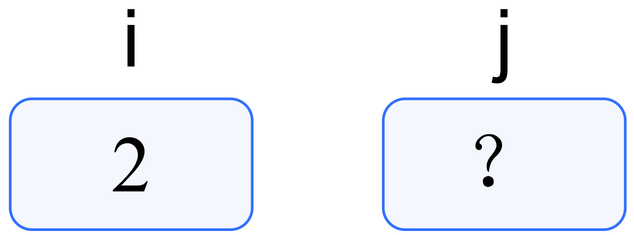
Increment and Decrement Operators

side effect

- – a change in the value of a variable as a result of carrying out an operation

Increment and Decrement Operators

Before..

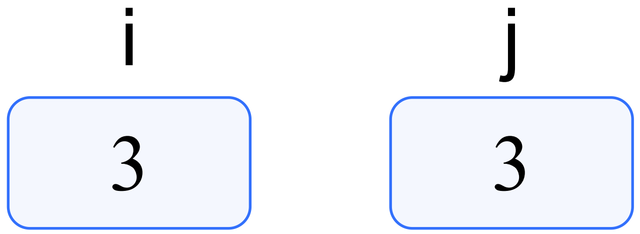


Increments...

j = ++i;

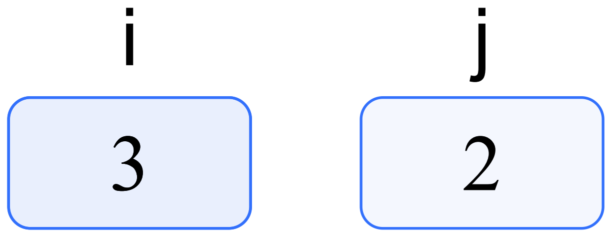
prefix:
Increment i and then use it.

After...



j = i++;

postfix:
Use i and then increment it..



The for Statement Syntax

```
for (*initialization expression;*
          *loop repetition condition;*
          *update expression*)
      statement;

/* Display N asterisks. */
for (count_star = 0;
       count_star < N;
       count_star += 1)
    printf(“*”);
```

do-while Syntax

```
do
        statement;
while (loop repetition condition);

/* Find first even number input */
do
        status = scanf(“%d”, &num);
while (status > 0 && (num % 2) != 0);
```

We will talk more about the output of scanf next time.

do-while Statement

- For conditions where we know that a loop must execute at least one time.

1. Get a *data value*
2. If *data value* isn't in the acceptable range, go back to step 1.

Computing a Sum or Product in a Loop

- **accumulator**
  - a variable used to store a value being computed in increments during the execution of a loop

Computing Factorial

logical complement (negation)

- loop body executes for decreasing value of **i** from **n** through 2
- each value of **i** is incorporated in the accumulating product
- loop exit occurs when **i** is 1

Figure Function to Compute Factorial

```
/*
 * Computes n!
 * Pre: n is greater than or equal to zero
 */
int
factorial(int n)
{
      int i,                    /* local variables */
          product;              /* accumulator for product computation */

      product = 1;
      / * Computes the product n × (n-1) × (n-2) × . . .  ×2 × 1
*/
      for (i = n; i > 1; --i) {
            product = product * i;
      }

      /* Returns function result */
      return (product);
}
```

Table Compound Assignment Operators

Statement with Simple Assignment Operator	Equivalent Statement with Compound Assignment Operator
count_emp = count_emp +1;	count_emp += 1;
time = time - 1;	time -= 1;
total_time = total_time + times;	total time += time;
product = product * item;	prouct *= item;
n = n* (x + 1);	n *= x + 1;

Loop Control Components

- initialization of the loop control variable
- test of the loop repetition condition
- change (update) of the loop control variable

- the for loop supplies a designated place for each of these three components

THE END

Fangtian Zhong
CSCI 112

Gianforte School of Computing
Norm Asbjornson College of Engineering
E-mail: fangtian.zhong@montana.edu

2025.02.10