Expression and Flow Control

Target

學會使用運算子

學會邏輯判斷

學會使用迴圈

Index

01. Expression

- Definition
- Operator

02. Flow Control

- Logical expressions
- Selection statements
- Loop
- Exit

01

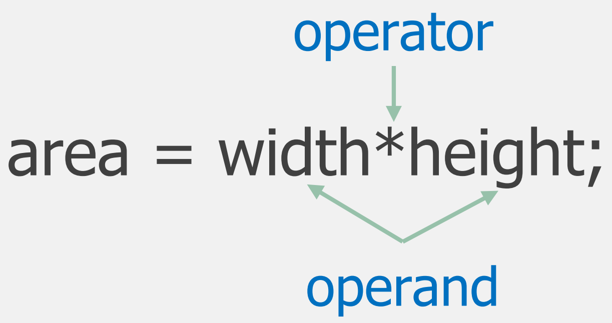
Expression

Expression

Definition

Definition

- Expression is one of C's distinguishing characters
- An expression is a sentence with two or more operands by one or more operators



operator

area = width*height;

operand

```
ratio = TWD / 30;
sum = 5 + 2;
value = (5 + 2) + ratio - area;
```

Expression

Operator

+	Addition	>>	Bit right shift
-	Subtraction	++	Prefix increment
*	Multiplication	--	Prefix decrement
/	Division	>	Greater than
%	Remainder	>=	Greater than or equal
+	Positive	<	Less than
-	Negative	<=	Less than or equal
~	Complement	==	Equality
&	And	!=	Inequality
|	Or	!	Not
^	XOR	&&	Logical And
=	Assignment	||	Logical Or
<<	Bit left shift		

Expression

Operator - Arithmetic operators

Binary

- ➢ An operator acts on two operands
- ➢ C provides 5 binary arithmetic operators
  - + - * / %

Unary

- ➢ It's used primarily to emphasize a numeric constant is positive
- ➢ C provides 2 unary arithmetic operators
  - + and -

+: plus i = +1; -: plus j = -1;

Expression

Operator - Arithmetic operators

Binary

- ➢ When int and float operands are mixed, the result has type float

  9 + 2.5f has the value 11.5, and 6.7f / 2 has the value 3.35

- ➢ The value of i%j is the remainder of i/j

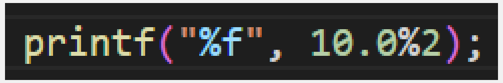
  8%3 is the value 2, 9%3 is the value 0

  Only for integer

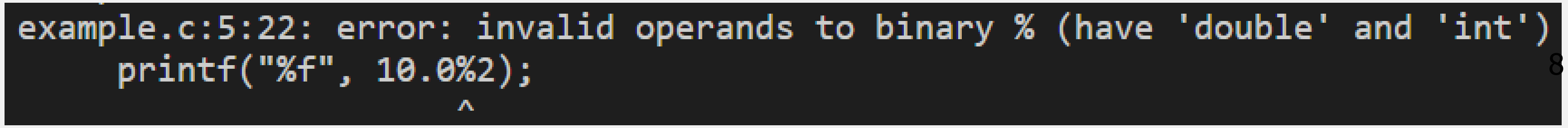
- ➢ The / and % operator require special care

  When both operand are integers, / truncates the result, for example 1 / 2 is 0 not 0.5

  The % operator requires integer operands; if either operand is not integer, the program won't compile



```
printf("%f", 10.0%2);
```



```
example.c:5:22: error: invalid operands to binary % (have 'double' and 'int')
     printf("%f", 10.0%2);
                      ^
```

Expression

Operator - Arithmetic operators

Binary

- The behavior when / and % are used with negative operands is implementation-defined in C89

	8%5	-8%5	8%-5	-8%-5
Quotient:	1	-1 or -2	-1 or -2	1 or 2
Remainder:	3	-3 or 2	3 or -2	-3 or 2

Expression

Operator - Precedence

Operator precedence

- It determines which operator is performed first in an expression with more than one operators

Highest: + - (unary)

* / %

Lowest: + - (binary)

Examples:

x + y * z = x + (y * z)

-x * -y = (-x) * (-y)

+x + y / z = (+x) + (y / z)

Expression

Operator - An Example

Write a program to compute a UPC check digit

- Most goods sold in U.S. and Canadian stores are marked with a University Product code

  First digit: Type of item

  First group of five digits: Manufacturer

  Second group of five digits: Product

  Final digit: Check digit, used to identify an error in the preceding digit

  How to compute check digit?

  Add the first, third, fifth, seventh, ninth, eleventh digits

  Add the second, fourth, sixth, eighth, and tenth digits

  Multiply the first sum by 3 and add it to the second sum

  Subtract 1 from the total

  Compute the remainder when the adjusted total is divided by 10

  Subtract the remainder from 9

Expression

Operator - An Example

If UPC is 0 13800 15173 5

First sum = 0 + 3 + 0 + 1 + 1 + 3 = 8

Second sum = 1 + 8 + 0 +5 + 7 = 21

Multiply the first sum by 3 and add it to second sum = 8 * 3 + 21 = 45

Subtract 1 from the total = 44

Remainder upon dividing by 10 = 4

Subtract the remainder from 9 = 5

```
Enter the first (single) digit: 0
Enter first group of five digits: 13800
Enter second group of five digits: 15173
Check digit: 5
```

Expression

Operator - Assignment

Assignment

- ➢ The variable can be set a value, i.e. assignment "="

```
int height;
float width;
height = 8;
width = 200.15;
```

- ➢ Hence, the variable must be declared before assigning a value
- ➢ The variable can be assigned by other variable

```
float area;
area = width *height
```

Expression

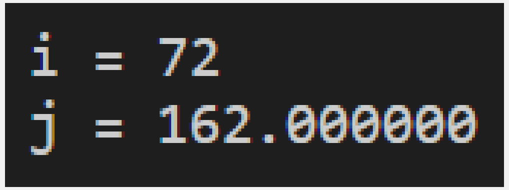
Operator - Assignment

Assignment

- If the types of i and *e* are different, the value of e will be converted to the type of i (*e* is the expression)

i = *e* ;

```
int i;
float j;
i = 72.93f;
j = 162;
```



```
i = 72
j = 162.000000
```

Expression

Operator - Assignment

Side effect

- An operator that alters one of its operands is defined as the side effect
- Several assignments can be chained together

```
int i, j, k;
k = j = i = 99;
k = (j = (i = 99));
```

- Watch out for unexpected results in chained assignments as a result of type conversion

```
int i;
float j;
j = i = 22.343f; -> ?
```

```
j = 22.0
i = 22
```

Expression

Operator - Assignment

Lvalues

- The assignment operator requires a ***lvalue*** as its left operand
- A lvalue represents an object stored in computer memory, not a constant or the result of a computation
- It's illegal to put any other kind of expression on the left side of an assignment expression

```
12 = i;       //Error
i + j = 0;  //Error
-i = j;       //Error
```

- The compiler will produce an error message such as "invalid lvalue in assignment"

Expression

Operator - Assignment

Compound assignment

```
int i = 1;
i = i + 2;
```

Compound assignment operator

- += -= *= %= /=
- i (operator)= (*e*); means i = i (operator) (*e*);

```
int i = 1, j = 2, k = 3;
i += 2;          // i = i + 2
i *= j+k         // i = i * (j+k)
```

Expression

Operator - Increment and Decrement

Increment and decrement operators

- "++" and "--"
  - ++ : adds 1 to its operand
  - -- : subtracts 1 to its operand
- They can be employed as prefix (++i) or postfix (i++) operators
- They have side effects

```
int prefix_i = 1;                              int postfix_i = 1;
printf("prefix_i is %d\n", ++prefix_i);        printf("postfix_i is %d\n", postfix_i++);
printf("prefix_i is %d\n", prefix_i);          printf("postfix_i is %d\n", postfix_i);
```

- "++prefix_i" means "increment prefix_i immediately", while "postfix_i++" means "use the old value of postfix_i for now, but increment it later"
- How much later? The C standard doesn't specify a precise time, but it's safe to assume that the variable will be incremented before the next statement is executed

Expression

Operator - Increment and Decrement

When ++ or -- is used more than once in the same expression, the result can often be hard to understand

```
i = 1;
j = 2;
k = ++i + j++;
```

- The last statement is equivalent to

```
i = i + 1;
k = i + j;
j = j + 1;
```

Expression

Operator - Increment and Decrement

Precedence	Name	Symbol(s)			Associativity
1	Postfix increment	Operand++			Left
	Postfix decrement	Operand--			
2	Prefix increment	++Operand			Right
	Prefix decrement	--Operand			
	Unary plus	+Operand			
	Unary minus	-Operand			
3	Multiplicative	Operand * / % Operand			Left
4	Additive	Operand + - Operand			
5	Assignment	Operand	= *= /= %= += -=	Operand	Right

Expression

Operator - Increment and Decrement

```
x = y += z++-i+--j / -k
x = y += (z++)-i+--j / -k
x = y += (z++)-i+(--j) / -k
x = y += (z++)-i+(--j) / (-k)
x = y += (z++)-i+((--j) / (-k))
x = y += ((z++)-i)+((--j) / (-k))
x = y += (((z++)-i)+((--j) / (-k)))
x = (y += (((z++)-i)+((--j) / (-k))))
```

<table>
<tr><th>Precedence</th><th>Name</th><th colspan="3">Symbol(s)</th></tr>
<tr><td>1</td><td>Postfix increment</td><td colspan="3">Operand++</td></tr>
<tr><td></td><td>Postfix decrement</td><td colspan="3">Operand--</td></tr>
<tr><td>2</td><td>Prefix increment</td><td colspan="3">++Operand</td></tr>
<tr><td></td><td>Prefix decrement</td><td colspan="3">--Operand</td></tr>
<tr><td></td><td>Unary plus</td><td colspan="3">+Operand</td></tr>
<tr><td></td><td>Unary minus</td><td colspan="3">-Operand</td></tr>
<tr><td>3</td><td>Multiplicative</td><td colspan="3">Operand * / % Operand</td></tr>
<tr><td>4</td><td>Additive</td><td colspan="3">Operand + - Operand</td></tr>
<tr><td>5</td><td>Assignment</td><td>Operand</td><td>=
*=
/=
%=
+=
-=</td><td>Operand</td></tr>
</table>

Expression

Operator - Important Concept

Order of subexpression evaluation

- Most expressions have the same value regardless of the order in which their subexpressions are evaluated
- However, this may not be true when a subexpression modifies one of its operands

```
int x = 10, y , z;
z = (y = x + 2) – (x = 1);
printf("x = %d\ty = %d\tz = %d\n", x, y, z);
```

Expression

Operator - Important Concept

Order of subexpression evaluation

- Besides the assignment operators, the only operators that modify their operands are increment and decrement
- When using these operators, be careful that an expression doesn't depend on a particular order of evaluation

```
int x = 2, y = 2, z;        int x = 2, y = 2, z;
z = x * x++;                z = y * x++;
```

- It's natural to assume that z is assigned 4. However, z could just as well as assigned 6 instead

Expression

Operator - Examples

Show the output produced by each of the following program fragments. Assume that i and j are int variables

(a)

```
i = 1;
printf("%d ", i++ - 1);
printf("%d", i);
```

(b)

```
i = 10, j = 5;
printf("%d ", i++ - ++j);
printf("%d %d", i, j);
```

(c)

```
i = 7, j = 8;
printf("%d ", i++ - --j);
printf("%d %d", i, j);
```

(a)

```
0 2
```

(b)

```
4 11 6
```

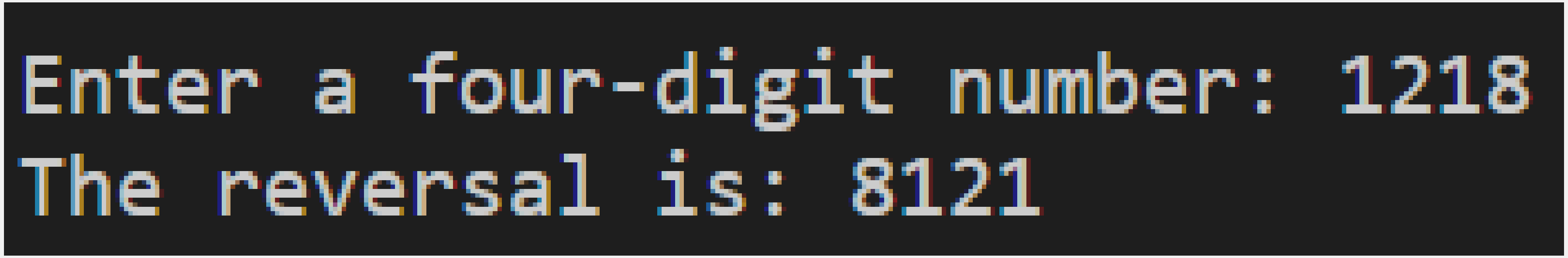
(c)

```
0 8 7
```

Expression

Operator - Examples

Write a program to reverse a four-digit number by using %d conversion specification



```
Enter a four-digit number: 1218
The reversal is: 8121
```

02

Flow Control

Flow Control

Logical expression

Excluding ***return*** and ***expression*** statements, most of remaining statements could be divided into the following types:

- Select: if and switch
- Iteration: for, while, and do
- Jump: break and continue

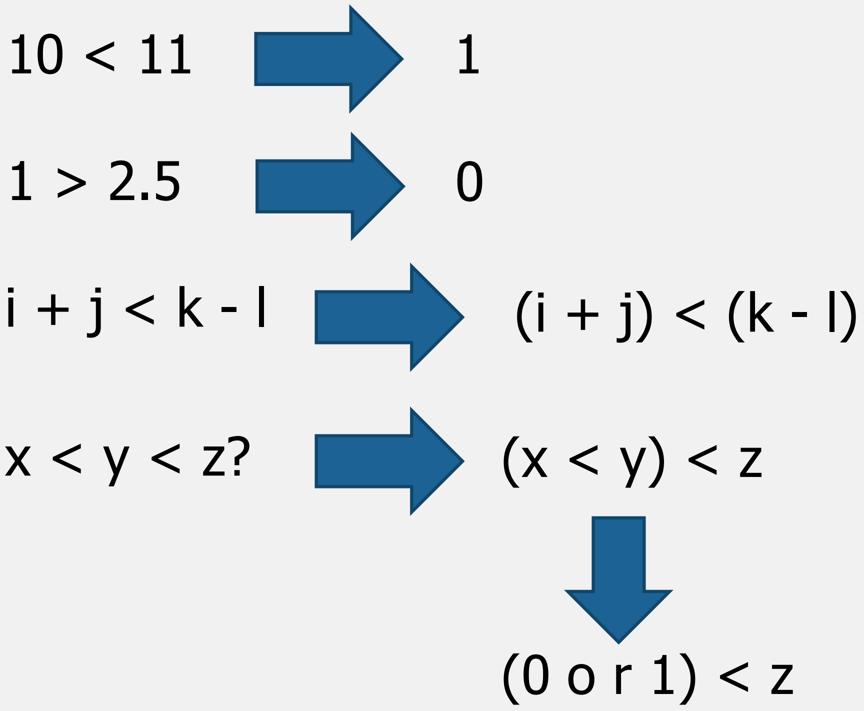
Logical expressions is built from

- Relational operators (< , <= , > , and >=)
- Equality operators (== and !=)
- Logical operators (&&, ||, and !)

Flow Control

Logical expression - Relational operators

The relational operators can be used to compare two operands with mixed types



10 < 11 → 1

1 > 2.5 → 0

i + j < k - l → (i + j) < (k - l)

x < y < z? → (x < y) < z → (0 or 1) < z

Symbol	Meaning
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to

Flow Control

Logical expression - Equality operators

The equality operators have lower precedence than the relation operators

Symbol	Meaning
==	Equal to
!=	Not equal to

i < j == j < k → (i < j) == (j < k)

(i >= j) + (i == j) → either 0, 1, or 2

Flow Control

Logical expression - Logical operators

The logical operators generate either 0 or 1

- The non-zero operand will be regarded as the true value and the zero one as false value
- The precedence of "&&" and "||" is lower than relation and equality operators

Symbol	Meaning
!	Logical "negative" (unary)
&&	Logical "and"
||	Logical "or"

Flow Control

Selection statements - If and else

if (*expression*) *statement*

- The parentheses around the expression are mandatory
- The word "then" is unnecessary in C
- When *if* statement is performed, the expression is evaluated and the statement is executed if the value after evaluating expression is non-zero

```
int x = y = 1;
if ( x == y)
   printf("Haha\n");
```

```
int x = y = 1;
if ( x = y)
   printf("Haha\n");
```

Flow Control

Selection statements - If and else

if (*expression*) *statement*

- How to design a if statement that will test whether a variable falls within a range of values?

```
if ( 0 <= x && x <= n)
```

- How to design a if statement that will test whether a variable is out of a range of values?

```
if ( x < 0 || n < x)
```

Flow Control

Selection statements - If and else

if (*expression*) *statement*

Compound statements

- ➢ The statement in the *if* template is singular, not plural

How to control two or more statements?

- ➢ Using {}

```
if ( x < y)
{
        x = y;
        printf("哭阿");
}
```

Flow Control

Selection statements - If and else

if (*expression*) *statement* else *statement*

The statement after the word else will be executed if the expression is not success

```
if ( x < y)
        x = y;
else
        y = x;
```

Flow Control

Selection statements - If and else

if (*expression*) *statement* else *statement*

There are no restrictions on what kind of statements can appear inside an *if* statement

```
if ( x < y)
  if (z < x)
     min = z;
  else
     min = x;
else
  if (z < j)
     min = z;
  else
     min = j;
```

Flow Control

Selection statements - If and else

if (*expression*) *statement* else *statement*

Add braces for easy modification and read

```
if ( x < y)
  if (z < x)
     min = z;
  else
     min = x;
else
  if (z < j)
     min = z;
  else
     min = j;
```

```
if ( x < y) {
  if (z < x)
     min = z;
  else
     min = x;
} else {
   if (z < j)
     min = z;
   else
     min = j;
}
```

```
if ( x < y) {
  if (z < x){
     min = z;
  } else {
     min = x;
  }
} else {
   if (z < j) {
     min = z;
   } else {
     min = j;
   }
}
```

Flow Control

Selection statements - If and else

if (*expression*) *statement* else if *statement* else *statement*

It is often to test a series of conditions, stopping as soon as one of them is true

```
if ( m < n)
  printf("m is less than n\n");
else
  if (m == n)
     printf("m is equal to n\n");
  else
     printf("m is greater than n\n");
```

```
if ( m < n)
  printf("m is less than n\n");
else if (m == n)
  printf("m is equal to n\n");
else
  printf("m is greater than n\n");
```

Flow Control

Selection statements - If and else

if (*expression*) *statement* else if *statement* else *statement*

```
if (expression)
    statement
else if (expression)
   statement
…
else if (expression)
   statement
else
   statement
```

Flow Control

Selection statements - If and else

Write a program that inputs a trade price and output a commission price

Trade price	*Commission rate*
Under $500	$20 + 1.5%
$500 ~ $1000	$30 + 0.93%
$1001 ~ $2000	$50 + 0.76%
$2001 ~ $3500	$70 + 0.55%
$3501 ~ $6500	$100 + 0.33%
Over 6500	$150 + 0.13%

The minimum Commission is 23

```
Enter price of trade: 2000
Commission: 65.2
```

Flow Control

Selection statements - If and else

Dangling else problem

```
if ( x != 0)
  if (y != 0)
     result = y/x;
else
  printf("Error the x is equal to 0\n");
```

```
if ( x != 0)
  if (y != 0)
     result = y/x;
  else
     printf("Error the x is equal to 0\n");
```

Flow Control

Selection statements - Conditional Expressions

C also provides an operator to allows an expression to execute one of two values depending on the value of a condition

The conditional expression contains two symbols, " ? " and " : "

expression 1 *?* *expression* 2 : *expression* 3

```
int x, y, z;
x = 1;
y = 2;
if (x > y) z = x;
else z = y;
if (x > 0) z= x + y;
else z = 0 + y;
```

```
int x, y, z;
x = 1;
y = 2;
z = x>y? x : y;
z = (x >=0? x : 0) + y;
```

Flow Control

Selection statements - Switch

Switch statement

```
switch (expression)
{
    case constant-expression: statements
    …
    case constant-expression: statements
    default: statements
}
```

- Controlling expression
  - The word switch must be followed by an integer expression in parentheses
  - The characters are also treated as integer
  - Floating-point and string don't qualify
- Case label
  - case constant-expression:
    - It is like an ordinary expression except that it can't contain variables or function calls
- Statements
  - No braces are required around the statements

Flow Control

Selection statements - Switch

Cascaded *if* Statement

```
if (grade == 3)
    printf("Very good\n");
else if (grade == 2)
    printf("Good\n");
else if (grade == 1)
    printf("Average\n");
else if (grade == 0)
    printf("Failing\n");
else
    printf("Illegal grade\n");
```

Switch statement

```
switch (grade)
{
    case 3:
           printf("Very good\n");
           break;
    case 2:
           printf("Good\n");
           break;
     case 1:
           printf("Average\n");
           break;
     case 0:
           printf("Failing\n");
           break;
     default:
           printf("Illegal grade\n");
           break;
}
```

Flow Control

Selection statements - Switch

The role of the break

- It causes the program to "break" out of the switch statement

```
switch (grade)
{
    case 3:
          printf("Very good\n");
    case 2:
          printf("Good\n");
    case 1:
          printf("Average\n");
    case 0:
          printf("Failing\n");
    default:
          printf("Illegal grade\n");
}
```

If the value of grade is 2, the message printed is?

Flow Control

Selection statements - Switch

Programmer sometimes put several case labels on the same line

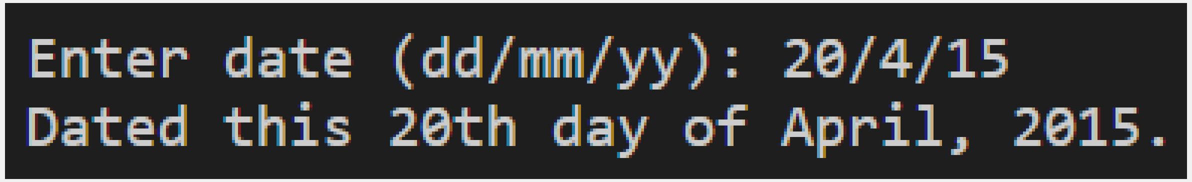
```
switch (grade)
{
    case 3:
    case 2:
    case 1:
          printf("Passing\n");
          break;
    case 0:
          printf("Failing\n");
          break;
    default:
          printf("Illegal grade\n");
          break;
}
```

```
switch (grade)
{
    case 3: case 2: case 1:
          printf("Passing\n");
          break;
    case 0:
          printf("Failing\n");
          break;
    default:
          printf("Illegal grade\n");
          break;
}
```

Flow Control

Selection statements - Switch

Write a program to display dates in the following formatting



```
Enter date (dd/mm/yy): 20/4/15
Dated this 20th day of April, 2015.
```

Flow Control

Loop

Loop

- It is used to repeat a block of code until completing the specified condition
- Every loop has a controlling expression and loop body

loop (*controlling expression*)

loop body

- Three types:
  - while
  - do…while
  - for

Flow Control

Loop - while

The *while* statement is the simplest and most fundamental

```
while (expression)
        statement
```

Example

```
while (x < n)  /*controlling expression*/
x = x * 2;     /*loop body*/
```

if n = 10, how many iteration does the loop body execute?

Flow Control

Loop - while

A trace of the loop when n = 10

```
while (x < n)
  x = x * 2;
```

x = 1;	x is now 1.
Is x < n?	Yes; continue.
x = x * 2;	x is now 2.
Is x < n?	Yes; continue.
x = x * 2;	x is now 4.
Is x < n?	Yes; continue.
x = x * 2;	x is now 8.
Is x < n?	Yes; continue.
x = x * 2;	x is now 16.
Is x < n?	No; exit from loop.

Flow Control

Loop - while

Compound statement

```
while (expression)
{
        statements
}
```

Example

```
while (x > 0)
{
        printf("T minus %d and counting\n", x)
        x--;
}
```

Flow Control

Loop - while

The *while* statement

- ➢ The controlling expression is false when a while loop terminates
- ➢ A while statement is often written in a variety of ways

```
while (x > 0)
{
   printf("T minus %d and counting\n", x);
   x--;
}
```

```
while (x > 0)
{
   printf("T minus %d and counting\n", x--);
}
```

Flow Control

Loop - while

Infinite loop

- A *while* statement didn't terminate if the controlling expression is a nonzero value

```
while (1)
{
   ...
}
```

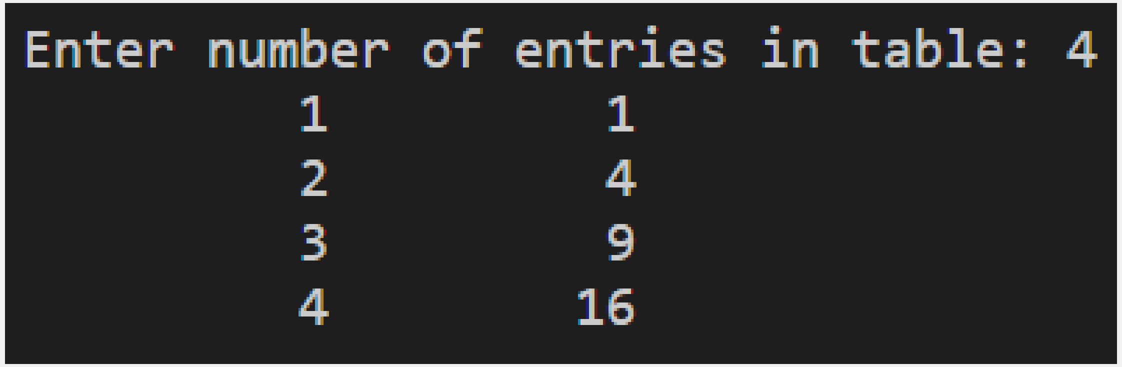
A *while* statement of this form will execute forever unless its body contains a statement that transfers control out of the loop (such as break, goto, return) or call a function that causes the program to terminate

Flow Control

Loop - while

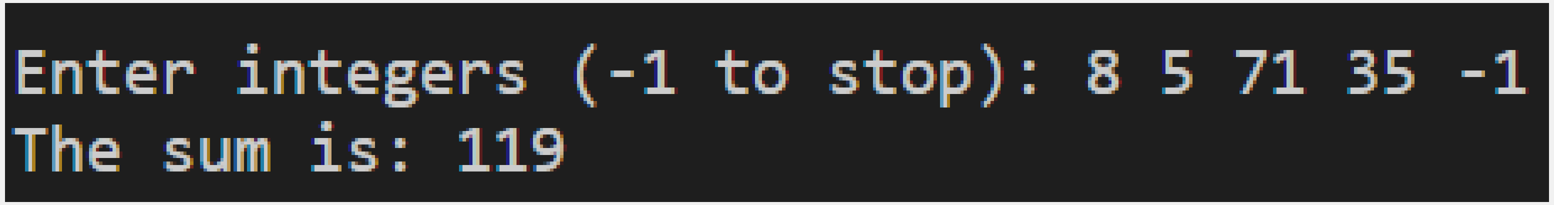
Two examples

- Write a program to print a table of squares



```
Enter number of entries in table: 4
         1         1
         2         4
         3         9
         4        16
```

- Write a program to summary a series of numbers



```
Enter integers (-1 to stop): 8 5 71 35 -1
The sum is: 119
```

Flow Control

Loop - do…while

The general form of *do…while* statement is

```
do {
  statements
} while (expression);
```

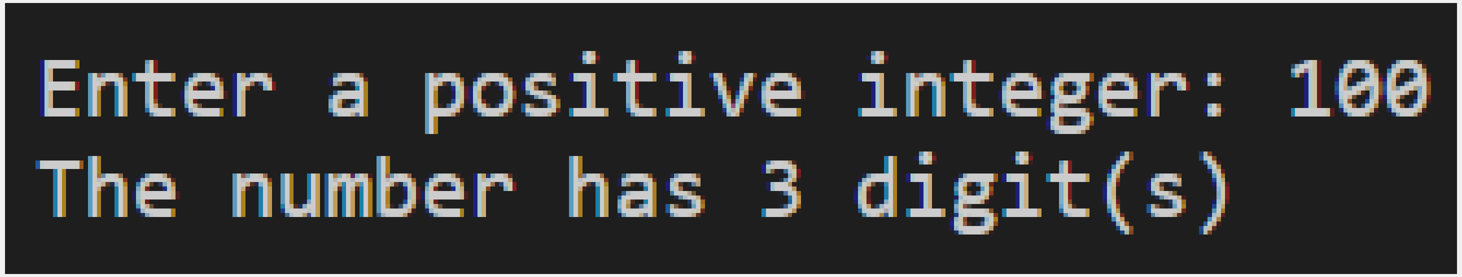
The do statement is essentially a while statement but performing controlling expression after each execution of loop body

```
i = 10;
do {
    printf("T minus %d and counting\n", i);
    --i;
} while (i > 0);
```

Flow Control

Loop - do…while

Write a program to calculate the number of digital in an integer



```
Enter a positive integer: 100
The number has 3 digit(s)
```

Flow Control

Loop - for

The *for* statement is the best way to write many loops

```
for (exp 1; exp 2; exp 3)
{
        statements
}
```

where exp 1 is the initialization, exp 2 is the stop condition, and exp 3 is the update condition

```
for (int x = 0; x < 10; x++)
{
        printf("x = %d", x);
}
```

Flow Control

Loop - for

The *for* statement ⇔ the *while* statement

```
for (exp 1; exp 2; exp 3)
{
        statements
}
```

```
exp 1;
while (exp 2)
{
        statements
        exp 3;
}
```

Flow Control

Loop - for

C allows any or all of the expressions that control a for statement to be omitted

If the first expression is omitted, no initialization is performed before the loop is executed

```
i = 10;
for (; i > 0; --i)
     printf("T minus %d and counting\n", i);
```

If the third expression is omitted, the loop body is responsible for ensuring that value of the second expression eventually becomes false

```
for (i = 10; i > 0;)
     printf("T minus %d and counting\n", i--);
```

Flow Control

Loop - for

When the first and third expressions are both omitted, the resulting loop is nothing more than a while statement in disguise

```
for (; i > 0;)
    printf("T minus %d and counting\n", i--);
```

=

```
while (i > 0)
    printf("T minus %d and counting\n", i--
  );
```

Flow Control

Loop - for

A variable declared by a for statement can't be accessed outside the body of the loop (we say that it's not visible outside the loop)

```
for (int i = 0; i < n; i++) {
    …
    printf("%d", i);  // legal, i is visible inside loop
    …
}
printf("%d", i);   // Error
```

Flow Control

Loop - for

A for statement may declare more than one variable by using the comma operator

exp 1_1, *exp* 1_2, exp 1_3, …

```
for (exp 1_1, exp 1_2, exp 1_3, … ; exp 2; exp 3)
{
        statements
}
```

```
sum = 0
for (x = 1; x <= 10; x++)
{
        sum += x
}
```

→

```
for (sum = 0, x = 1; x = 10; x++)
{
        sum += x
}
```

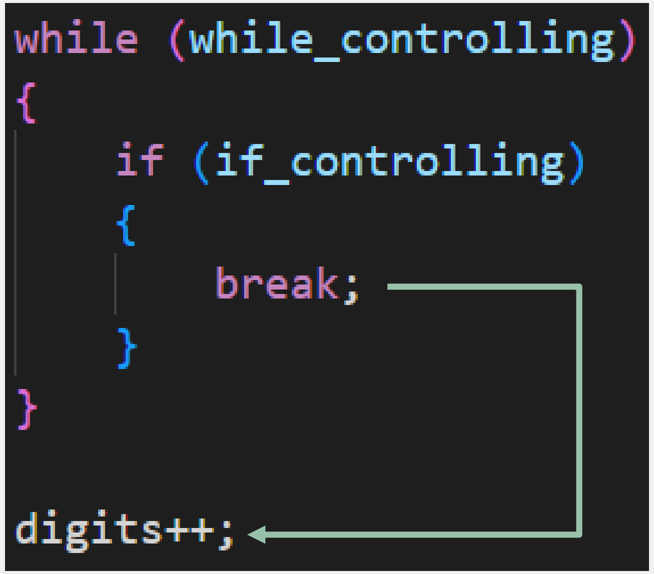
Flow Control

Exit

If we want to exit a loop in the middle, using the following statement

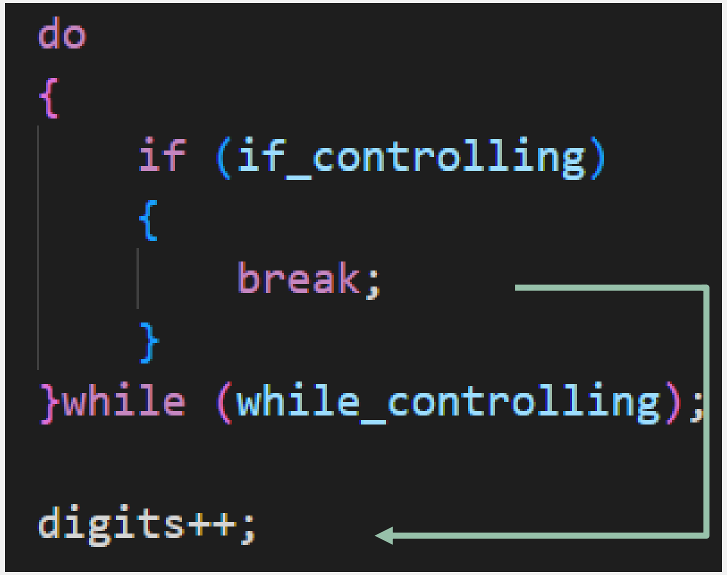
- ➢ break
- ➢ continue

break



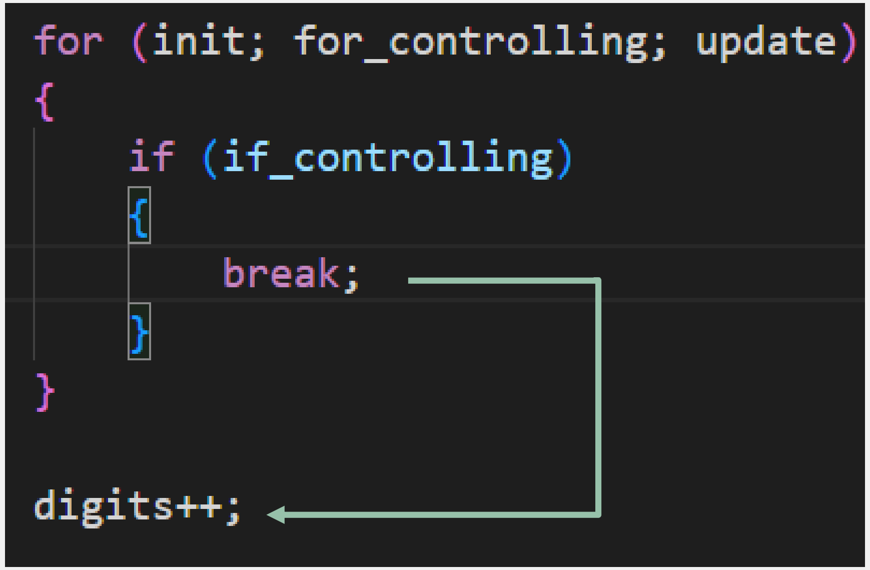
```
while (while_controlling)
{
    if (if_controlling)
    {
        break;
    }
}

digits++;
```



```
do
{
    if (if_controlling)
    {
        break;
    }
}while (while_controlling);

digits++;
```



```
for (init; for_controlling; update)
{
    if (if_controlling)
    {
        break;
    }
}

digits++;
```

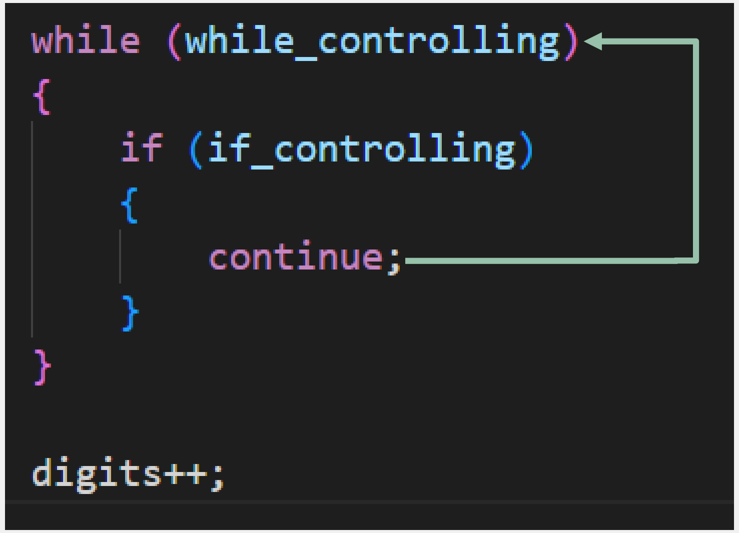
Flow Control

Exit

If we want to exit a loop in the middle, using the following statement

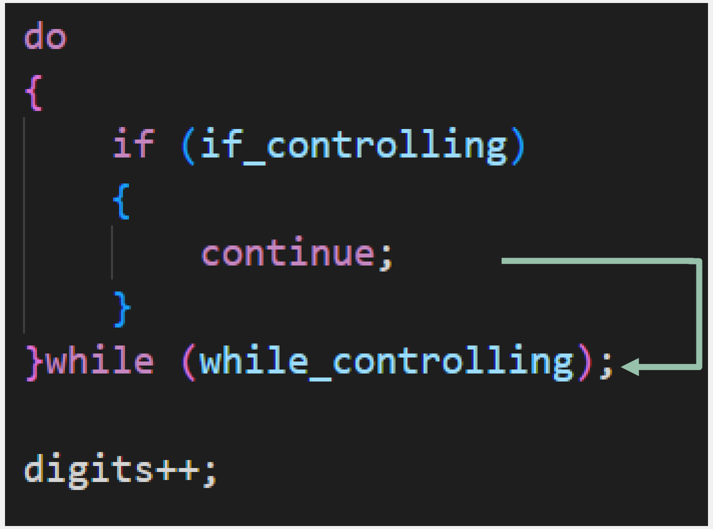
- break
- continue

continue



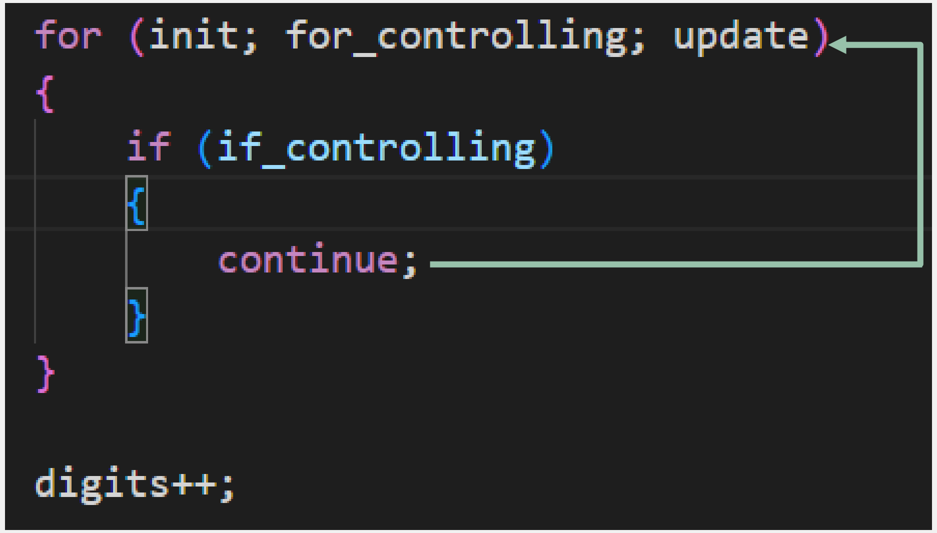
```
while (while_controlling)
{
    if (if_controlling)
    {
        continue;
    }
}

digits++;
```



```
do
{
    if (if_controlling)
    {
        continue;
    }
}while (while_controlling);

digits++;
```



```
for (init; for_controlling; update)
{
    if (if_controlling)
    {
        continue;
    }
}

digits++;
```

Flow Control

Exit

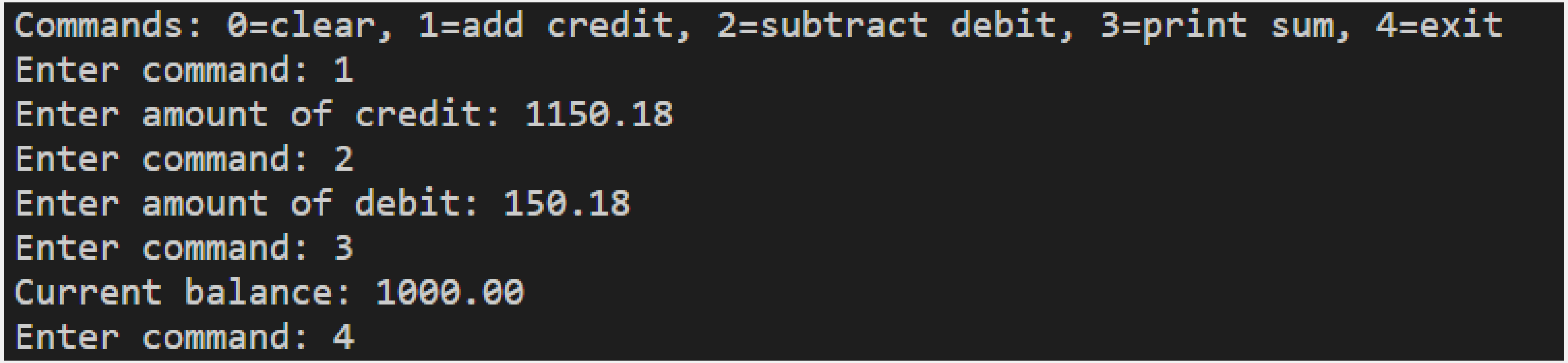
A break statement transfers control out of the innermost enclosing while, do, for, or switch statement

```
while (…)
{
    switch(…)
    {
          …
          break;
          …
    }
}
```

Flow Control

Exit

Write a program to calculate a check-book balance using for and switch statement



```
Commands: 0=clear, 1=add credit, 2=subtract debit, 3=print sum, 4=exit
Enter command: 1
Enter amount of credit: 1150.18
Enter command: 2
Enter amount of debit: 150.18
Enter command: 3
Current balance: 1000.00
Enter command: 4
```