Introduction to C

Control Flow

Instructor: Yin Lou

01/26/2011

Statements

Statements

<statement> := <expression>;

```
x = 0;
++i;
printf("%d", x);
```

Blocks

<block> := {<statements>}

```
{
x = 0;
++i;
printf("%d", x);
}
```

Blocks

<block> := {<statements>}

```
{
x = 0;
++i;
printf("%d", x);
}
```

- A block is syntactically equivalent to a single statement.

Blocks

<block> := {<statements>}

```
{
x = 0;
++i;
printf("%d", x);
}
```

- A block is syntactically equivalent to a single statement.
  - if, else, while, for
  - Variables can be declared inside *any* block.
  - There is no semicolon after the right brace that ends a block.

Example

```
int x = 0;
{
    int x = 5;
    printf("Inside: x = %d\n", x);
}
printf("Outside: x = %d\n", x);
```

Example

```
int x = 0;
{
    int x = 5;
    printf("Inside: x = %d\n", x);
}
printf("Outside: x = %d\n", x);
```

Inside: x = 5
Outside: x = 0

if Statement

if (<condition>) <statement>

```
// single statment
if (2 < 5)
    printf("2 is less than 5.\n");

// block
if (2 < 5)
{
    printf("I'll always print this line.\n");
    printf("because 2 is always less than 5!\n");
}
```

if-else Statement

if (<condition>) <statement1> else <statement2>

```
if (x < 0)
{
    printf("%d is negative.\n", x);
}
else
{
    printf("%d is non-negative.\n", x);
}
```

else-if Statement

```
if (a < 5)
    printf("a < 5\n");
else
{
    if (a < 8)
        printf("5 <= a < 8\n");
    else
        printf("a >= 8\n");
}
```

```
if (a < 5)
    printf("a < 5\n");
else if (a < 8)
    printf("5 <= a < 8\n");
else
    printf("a >= 8\n");
```

if-else Statement Pitfalls

```
if (a > 70)
    if (a > 80)
        printf("grade = B\n");
else
    printf("grade < B\n");
    printf("Fail.\n");
printf("Done.\n");
```

```
if (a > 70)
{
    if (a > 80)
    {
        printf("grade = B\n");
    }
    else
    {
        printf("grade < B\n");
    }
}
printf("Fail.\n");
printf("Done.\n");
```

Relational Operators

C has the following relational operators

a == b	true iff a equals b
a != b	true iff a does not equal b
a < b	true iff a is less than b
a > b	true iff a is greater than b
a <= b	true iff a is less than or equal to b
a >= b	true iff a is greater than or equal to b
a && b	true iff a is true and b is true
a || b	true iff a is true or b is true
!a	true iff a is false

Booleans in C

- C DOES NOT have a boolean type.
- Instead, conditional operators evaluate to integers (int)
  - 0 indicates false. Non-zero value is true.
  - if (<condition>) checks whether the condition is non-zero.

Booleans in C

- C DOES NOT have a boolean type.
- Instead, conditional operators evaluate to integers (int)
  - 0 indicates false. Non-zero value is true.
  - if (<condition>) checks whether the condition is non-zero.
  - Programmer must be very careful to this point!

Booleans in C

- C DOES NOT have a boolean type.
- Instead, conditional operators evaluate to integers (int)
  - 0 indicates false. Non-zero value is true.
  - if (<condition>) checks whether the condition is non-zero.
  - Programmer must be very careful to this point!

Examples

```
if (3)
    printf("True.\n");

if (!3)
    // unreachable code

if (a = 5)
    // always true, potential bug (a == 5)

int a = (5 == 5); // a = 1
```

Conditional expressions

<condition> ? <expression1> : <expression2>

```
grade = (score >= 70 ? 'S' : 'U');

printf("You have %d item%s.\n", n, n == 1 ? "" : "s");
```

Conditional expressions

<condition> ? <expression1> : <expression2>

```
grade = (score >= 70 ? 'S' : 'U');

printf("You have %d item%s.\n", n, n == 1 ? "" : "s");
```

Conditional expression often leads to succinct code.

switch Statement

A common form of if statement

```
if (x == a)
    statement1;
else if (x == b)
    statement2;
...
else
    statement0;
```

switch Statement

A common form of if statement

```
if (x == a)
    statement1;
else if (x == b)
    statement2;
...
else
    statement0;
```

switch statement

```
switch (x)
{
    case a: statement1; break;
    case b: statement2; break;
    ...
    default: statement0;
}
```

More on switch Statement

Fall-through property

```
int month = 2;

switch (month)
{
    case 1:
        printf("Jan.\n");
        break;
    case 2:
        printf("Feb.\n");
    case 3:
        printf("Mar.\n");
    default:
        printf("Another month.\n");
}
```

More on switch Statement

Fall-through property

```
int month = 2;

switch (month)
{
    case 1:
        printf("Jan.\n");
        break;
    case 2:
        printf("Feb.\n");
    case 3:
        printf("Mar.\n");
    default:
        printf("Another month.\n");
}
```

Feb.
Mar.
Another month.

More on switch Statement

Fall-through property

```
int month = 2;
int days;

switch (month)
{
    case 2:
        days = 28;
        break;
    case 9:
    case 4:
    case 6:
    case 11:
        days = 30;
        break;
    default:
        days = 31;
}
```

More on switch Statement

Fall-through property

```
int month = 2;
int days;

switch (month)
{
    case 2:
        days = 28;
        break;
    case 9:
    case 4:
    case 6:
    case 11:
        days = 30;
        break;
    default:
        days = 31;
}
```

It's always recommended to have default, though it's optional.

while Loop

- while (<condition>) <statement>

while Loop

- while (<condition>) <statement>
  - If the condition is initially false, the statement is never executed.

while Loop

- while (<condition>) <statement>
  - If the condition is initially false, the statement is never executed.
- do <statement> while (<condition>);

while Loop

- while (<condition>) <statement>
  - If the condition is initially false, the statement is never executed.
- do <statement> while (<condition>);
  - The statement is executed at least one.

for Loop

for (<exp1>; <exp2>; <exp3>) <statement>

```
exp1;
while (exp2)
{
    statement
    exp3;
}

for (i = 0; i < n; ++i)
{
    // do something
}
```

Infinite Loop

```
while (1)
{
    // do something
}
```

```
for (;;)
{
    // do something
}
```

Infinite Loop

```
while (1)
{
    // do something
}
```

```
for (;;)
{
    // do something
}
```

Both are okay, but for may lead to fewer machine code on some platform, which means it is slightly more efficient.

break and continue

break

```
int n = 10;
while (1)
{
    if (!n)
    {
        break;
    }
    --n;
}
```

continue

```
int i;
for (i = 0; i < 10; ++i)
{
    if (i == 0)
    {
        continue;
    }
    printf("%d\n", i);
}
```

Common Pitfalls

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

```
int n = 10;
while (n = 1)
{
    printf("%d\n", n);
    --n;
}
```

How to Avoid Bugs?

Always use {}

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

Put literals on the left

```
int n = 10;
while (1 == n)
{ // 1 = n, compilation error
    printf("%d\n", n);
    --n;
}
```