Lecture03: Control Flow

9/24/2012

Slides modified from Yin Lou, Cornell CS2022: Introduction to C

Administrative things

- Any problem with submitting assignment #1?
- Assignment #2 is on the course webpage due next week.

Review from Last Week

- Variable declaration
  - Type, variable name
- C Punctuations
  - ; , “ { } (% #
- printf() : write to the screen
  - printf(f**ormat string**, arg1, arg2, …)
  - What is a format string? Format tags whose values are substitute by args.
  - What are format tags?
- scanf() : read from the screen (user)
  - scanf(format string, **&**arg1, &arg2, …)
  - Store values into locations pointed by args

main.c: Variables & Punctuations

```
#include <stdio.h>

int main()
{
    int a, b, c;
    a = 10;
    b = 20;
    c = a * b;
    printf("a = %d b = %d c = %d\n", a, b, c);
    return 0;
}
```

More on printf()

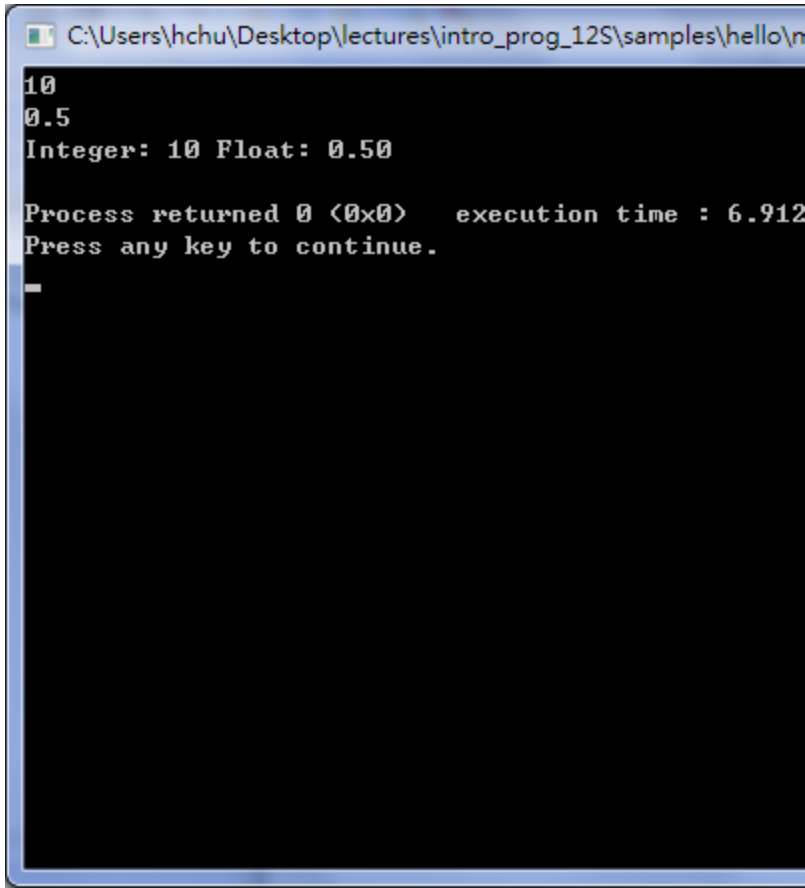
- printf(format string, val1, val2);
  - format string can include placeholders that specify how the arguments val1, val2, etc. should be formatted
  - %c : format as a character
  - %d : format as an integer
  - %f : format as a floating-point number
  - %% : print a % character
- Example

```
double f = 0.95;
printf("f = %f%%\n", f * 100);
```

scanf()

```
#include <stdio.h>

int main()
{
    int i;
    double f;
    scanf("%d", &i);
    scanf("%lf", &f);
    printf("Integer: %d Float: %2.2f\n", i, f);
    return 0;
}
```



New Stuff: Control Flow

Statement

<statement> := <expression>;

```
x = 0;
++i;                    // i = i + 1;
printf("%d", x);
```

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);
```

Inside: x = 5
Outside: x = 0

if Statement

if (<condition>) <statement>

```
// single statement
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.
  - boolean means true or false.
- 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!

Example

```
if (3)
        printf("True.\n");
if (!3)
        // unreachable code
if (a = 5)
        // always true, potential bug (a == 5)
int a = (5 == 5); // a = 1
```

In-Class Exercise #1

Write a program that reads a number grade (0-100) and converts it into a letter grade (ABCF): A:80-100, B:70-79, C:60-69, and F<60.

Your program should have the following input & output:

Input a number grade> 80
Letter grade: A

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 concise code.

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");
}
```

```
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;
}
```

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

```
if (month == 2)
{
    days = 28;
}
else if (month == 9))
|| (month == 6) || (month == 11))
{
    days = 30;
}
```

In-Class Exercise #2

Write a program that reads a month (1-12) and converts it into a season: Spring:3-5, Summer:6-8; Fall:9-11 Winter:12-2. **Use the switch statement**.

Your program should have the following input & output:
Input a month(1-12)> 1
Season: Winter

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)   // ++i is the same as i = i + 1;
{
    // 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;
    }
    printf(“%d\n”, n)
    --n;
}
```

continue

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

Common Pitfalls

Always use {}

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

Put literals on the left

```
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;
}
```

Review for Today

- **if, else-if**
- **< condition> ? <expression1> : <expression2>**
- **switch, case, default, break**
- **for (<exp1>; <exp2>; <exp3>) <statement>**
- **while (<condition>) <statement>**
- **do <statement> while (<condition>);**
- **break, continue**

In-Class Exercise #3

Write a program that print all Fibonacci numbers smaller than 100. Fibonacci numbers are numbers in the following integer sequence:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, …

Your program should output exactly the following line. Note that there should not be a “,” after the last number.

Fib: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89

In-Class Exercise #4

Write a program that print the pi number up to 100 smaller than 100.
Fibonacci numbers are numbers in the following integer sequence:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, …

Your program should output exactly the following line. Note that there should not be a “,” after the last number.

Fib: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89