Programming in C

Lecture #3
Kenneth W. Flynn
RPI CS

Outline
- Theory: Loops
- Application: While Loops
- Application: Do-While Loops
- Exercise: Activity 3.1
- Theory: Logical Operators / Boolean Logic
- Application: C Logical Operators
- Application: For loops
- Exercise: Activity 3.2

Loops
- Most computer programs do the same things, over and over again
- For example, consider a calculator: it sits there until you type input, does your operation, and then waits for more input. Repeat until the user says stop
- Loops are a way of repeating code as many times as needed

Gotos
- Many languages (BASIC, etc.) used to have "Goto" statements, which would go to a specific line number in a program
- It was proven that all such control flow can be done with basic looping structures
- Loops are clearer, allowing the programmer to more easily see what code is repeated, and can be modified more easily
- C has goto. NEVER use it

Kinds of Loops
- While
  - Simplest
  - While some condition is true, repeat a series of statements
- Do - While
  - Do the following set of statements. While some condition is true, do it again
- Repeat - Until
  - Do the following set of statement, until some condition is not true. (Can be simulated with Do-While)
Kinds of Loops II

- For loops
  - Initialize a counter
  - If the counter is less than some value, do some set of statements
  - Increment the counter. Repeat Step 2.

- C has While loops, Do-While, and For loops. For loops in C are more general however.

C While City

- while (condition)
  /* Loop Body */

- The "Loop Body" is one of the following:
  - Single statement (including a mere ;)
  - Block

While-y Examples

- x = 6;
  while (x > 5)
    { printf("Num < 5: ");
      scanf("%d", &x);
    }
  printf("%d is < 5\n", x);

More Examples

- x = 0;
  while (x++ < 16)  
    printf("%d\n");
- x = 1;
  while (x != 1);

Do-While

- do
  /* Loop Body */
  while (condition);
  
- int x = 1;
  do
  { 
    x++;
    printf("%d\n", x);
  } while (x < 1);
Hints For Loops

△ Select appropriate kind of loop
△ Initialize what it is you're checking to some known state (x = 0)
△ Check your termination condition carefully. Make sure it will eventually be reached
△ Infinite loops occur when a loop runs forever, without stopping. Ctrl-C is useful.

The Multiplier

△ Write an algorithm, and a program to multiply numbers until the user says stop, as in the sample output below:

```
~> a.out
Num 1: 5
Num 2: 6
The product is: 30
0 = End: 1
Num 1: 95
Num 2: 65
The product is: 6175
0 = End: 0
~>
```

Boolean Logic

△ Logic is quite a complicated topic, let's cover the basics
△ Consider the following:
  □ If my first name is "Kenn"
  □ AND
  □ my last name is "Flynn"
  □ then I'm your instructor.
△ AND is a logical operator

Logical Operators

△ And, Or, Not
△ Consider:
  □ x AND y
  □ x OR y
  □ Not x

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>x AND y</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>x OR y</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
</tbody>
</table>

Kenneth W. Flynn (13-18)
11/08/98
Logic in C

- 0 = False
- Anything else (1, 2, -5) is True
- And: &&
- Or: ||
- Not: !

Logic Examples

- while (1) /* Loop Forever */
- while (0) /* Don't loop */
- while (1 && 1) /* Loop Forever */
- while (0 || 1) /* Loop Forever */
- while ((x > 0) && (x < 5))
- while (!x <= 0) && !x >= 5)
- while (!(x <= 0) || (x >= 5))

Lazy Evaluation

- Consider:
  - while (1 || (x++ < 5));
  - Will x be incremented? NO!
  - C has lazy evaluation
  - || stops evaluation when it reaches the first true argument, left to right
  - && stops evaluation when it reaches the first false argument, left to right
- Consider the following advanced example:

Lazy Example

```c
int x, y, z;

printf("Num Den: ");
while (scanf("%d%d", &x, &y) &&
 (y != 0) &&
 (z = x / y)
 )
 printf("%d, Next Num Den: ", z);
printf("Done.
");
```

```
~> a.out
Num Den: 5 3
1, Next Num Den: 5 0
Done.
~>
```
**Back to Loops**

- In C, the most commonly used loop is the very powerful For loop
- For loops in C differ from traditional For loops in many ways
- C For loops can serve as normal For loops, but they can also do much more, such as while type loops
- Use the simplest loop that will work

**C For Loops**

```c
for (initializations; continue-condition; updates)
    /* Loop Body */
```

- For loops cause the following steps to be evaluated in this order:
  - Execute initializations
  - Check continue condition
  - Execute loop body
  - Execute updates
  - Go to Step 2

**For Example**

```c
for (x=0; x<5; x++)
    printf("%d ",x);
```

- First x is set to 0
- Since x < 5, we execute the loop body
- 0 is printed
- x is incremented, x is now 1
- Since 1 < 5, we execute the loop body
- 1 is printed
- x is incremented, x is now 2
- ....

**For Example II**

```c
for (x=0; x<5; x++)
    printf("%d ",x);
```

- ... 4 is printed
- x is incremented, x is now 5
- Since 5 is not < 5, the loop ends.
- Output 0 1 2 3 4

**Observations**

- For loops are always used in count controlled loops (i.e. loops that increment a number)
- They are frequently used in other situations that are evocative of counter controlled loops
- Use your judgement in selecting appropriate loops, but keep in mind for loops can sometimes simplify your code

---

Activity 3.2

Kenneth W. Flynn (25-30)
11/08/98
The Squarer
△ Write a program to print the squares of all the numbers between two input values.
△ Make sure the first value is less than the second value, if not print an error message.

Switch
△ Switch statements are sort of a special if-then-else-if-then-else type construct
△ An integer value is determined, depending on this value, one of many cases is evaluated.

Switch Syntax
△ switch (value)
  {
    case val1: /* Statements */
      break;
    case val2: /* Statements */
      break;
    /* More cases */
    default: /* Statements */
      break;
  }

Switch Notes
△ Statements in a case are executed until a break is reached, even into other cases
△ Default case is evaluated if no other case matches. Include a default case in all switch statements
△ Use switch statements as an alternative to massively nested if - then - else structures

Switch Example
```c
int x, y, z;
int err = 0; /* Init to false */
printf("Num Den: ");
scanf("%d", &x, &y);
switch (y) /* Better as an if... */
  {
    case 0: err = 1; /* Set error */
      break;
    default: z = x / y;
      break;
  }
if (!err)
  printf("Z is %d\n", z);
else
  printf("ERROR!\n");
```
The Chooser

Write a program that gets one number from the user. If the number is 0, print out "Zero", if it is 1, print out "One", etc. Handle the numbers between 0 and 5. All other numbers should produce an error message. Use a switch statement (see handout).