CSE 220 - C Programming
Fall 2015

Repetition and Loops
Control of Flow

• Selection statements:
  – Select a particular path of execution

• Iteration statements:
  – Repeat a particular fragment

• Jump statements:
  – Jump to another place in the code
Repetition Example

Recall the decimal to binary conversion algorithm

Repetition:
- division by 2

Stopping condition:
- Quotients becomes < 2
3 Types of Repetition Structures

- `while`
- `do/while`
- `for`

Objectives:

- Explain the three basic types of control structures
- Explain how the three kinds of repetition structures work
- Apply the concept of repetition control structures to few practical problems
3 Types of Repetition Structures

- **while** tests a *condition* at the *beginning* of the loop
  condition must first be true for the loop to run even once

- **do/while** tests a *condition* at the *end* of the loop
  loop will run at least once

- **for** facilitates initializing and incrementing the variable that
  controls the loop
  Especially helpful for:
  - Looping for a *known* number of times
  - Loops that *count* or that need to increment a variable
While Statement
The while Loop – Pseudocode and Flowchart

**WHILE condition** is TRUE, repeat:
- Statement1
- Statement2
- Statement3
- ...

**END WHILE**

- Statement(s) is(are) executed *while* condition is true

- Note that the condition must first be true in order for the statement to be executed even once
while loop C syntax

- General form of a while loop:

```c
while (condition) /* while condition is TRUE (i.e., ! = 0) */
    statement1; /* execute this statement */
```

When `condition` becomes false (i.e. `== 0`), looping stops and the next statement is executed.
while loop C syntax

- General form of a while loop:

```c
while (condition) /* while condition is TRUE (i.e., ! = 0) */
    statement1;  /* execute this statement */
```

When `condition` becomes false (i.e. == 0), looping stops and the next statement is executed.

- Compound statement form:

```c
while (condition) {
    statement1;
    statement2;
    statement3;
}
```

Body with multiple statements
While Statement

while ( expression ) statement;
while ( expression ) { statements; }

• () required around expression
• expression: controlling expression
• statement/statements: loop body

int distance = 1;
while (distance < 10)
    distance += 2;
Example 1: a simple counter

```c
#include <stdio.h>

int main()
{
    int counter = 1;

    while(counter <= 10)
    {
        printf("%d\n",counter);
        counter = counter + 1;
    }

    return 0;
}
```
Example 2: read until positive number

```c
#include <stdio.h>

int main()
{
    int x;
    printf("enter a positive number please: ");
    scanf("%d", &x);

    while(x <= 0)
    {
        printf("You should enter a positive number, please try again: ");
        scanf("%d", &x);
    }

    printf("The positive number that you entered is: %d \n", x);
    return 0;
}
```
#include <stdio.h>

int main()
{
    int i = 3;

    while(i > 0)
    {
        printf("Still positive: (%d). Decrement!\n", i);
        i = i - 1;
    }

    return 0;
}
Infinite Loop

```c
#include <stdio.h>

int main()
{
    int i = 3;
    while(i > 0)
    {
        printf("Still positive: (%d). Decrement!\n", i);
        i = i - 1;
    }
    return 0;
}
```

Still positive: (3). Decrement!
Still positive: (2). Decrement!
Still positive: (1). Decrement!
Infinite Loop

- If the controlling expression is always nonzero, the while statement won’t terminate

```c
#include <stdio.h>

int main()
{
    int i = 3;

    while(i > 0)
    {
        printf("Still positive: (%d). Decrement!\n", i);
        // i = i - 1;
    }

    return 0;
}
```

No decrement on i!
The condition always holds
Infinite Loop

• If the controlling expression is always nonzero, the while statement won’t terminate
Do/While Statement
do/while Structure flowchart

statement is executed at least once no matter what the condition is!
do/while Loop Structure

General form of a **do/while** loop:

```
do
  statement1; /* execute this statement */
while (condition); /* while condition is TRUE */
```
do/while Loop Structure

General form of a **do/while** loop:

```
    do
        statement1; /* execute this statement */
    while (condition); /* while condition is TRUE */
```

When `condition` becomes false, looping stops, and
the next statement after the `while` is executed

Note: `statement1` will be executed *at least once*

**Remember:** to DO more than one statement in the loop,
enclose the statements in curly braces `{ }`
called a `compound` statement
do while Statement

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

• () required around expression
• expression: controlling expression
• statement/statements: loop body

• Similar to while except: the controlling expression is executed after the loop body is executed
While / Do comparison

```c
int count = 0;
while (count < 5) {
    printf("%d\t",count++);
}
```

```c
int count = 0;
do {
    printf("%d\t",count++);
} while (count < 5);
```
While / Do comparison

```c
int count = 0;
while (count < 5)
{
    printf("%d\t",count++);
}
```

```c
int count = 0;
do
{
    printf("%d\t",count++);
}
while (count < 5);
```

0 1 2 3 4

0 1 2 3 4
While / Do comparison

int count = 10;
while (count < 5) {
    printf("%d\t",count++);
}

int count = 10;
do {
    printf("%d\t",count++);
} while (count < 5);

• do loop is always executed for the first iteration
While / Do comparison

```c
int count = 10;
while (count < 5)
{
    printf("%d\t",count++);
}
```

```
int count = 10;
do
{
    printf("%d\t",count++);
} while (count < 5);
```

Nothing is printed

• do loop is always executed for the first iteration
for statement
for Loop Structure

General form of a `for` loop:

```plaintext
for(expression1; expression2; expression3)
    statement1;  /* execute this statement */
```

- **expression1** initializes the variable controlling the loop
  
  ```plaintext
  i = 0;
  ```

- **expression2** is the `condition` for continuing the loop
  
  ```plaintext
  i <= 10;
  ```

- **expression3** increments the control variable
  
  ```plaintext
  i++ /* same as i=i+1 */
  ```
for Loop Structure

General form of a for loop:

```c
for(expression1; expression2; expression3)
    statement1;  /* execute this statement */
```

- **expression1** initializes the variable controlling the loop
  
  ```c
  i = 0;
  ```

- **expression2** is the condition for continuing the loop
  
  ```c
  i <= 10;
  ```

- **expression3** increments the control variable
  
  ```c
  i++;  /* same as i=i+1 */
  ```

- Note that there is **NO** semicolon after **expression3**! or after the closing parenthesis

- To execute more than one statement in the for loop, enclose them in curly braces `{ }`
for Loop Structure – Flow Chart

```latex
\textbf{for}(\textit{expression1}; \textit{expression2}; \textit{expression3})
```

- \textit{expression1} initializes the loop control variable: \( \text{ex. } i = 0; \)
- \textit{expression2} tests the loop control variable to see if it is time to quit looping: \( \text{ex. } i < 10; \)
- If \( \text{T} \), execute \textit{statement}.
- If \( \text{F} \), continue looping.
- \textit{expression3} increments the loop control variable: \( \text{ex. } i++ \)
for (i = 1 ; i < 10; i = i+1) {
    statement 1;
    statement 1;
    ...
}
for statement

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

Pseudo-code:

1. Evaluate **expr1**: initialize i to 0
2. Test **expr2**: is i < 10?
   - If true:
     - Execute **loop body**: `printf("...")`
     - Execute **expr3**: i++
     - Go to (2)
   - If false: exit loop