Loops I

Chapter 6

Objectives

You will be able to:

- Use “while” loops correctly.
- Use the “break” statement correctly.
- Recognize common errors in the above.
- Apply good programming style in writing the above.

Loops

- Often we want to execute a block of code multiple times.
  - Something is always different each time through the block.
  - Typically a variable is being incremented.
- C provides several ways to do this.
  - We will learn one of them now.
  - The “while” loop

The **while** Loop

Same as for an “if” statement

```
while (condition)
{
    /* Code block to execute
       repeatedly until condition
       is not true   */
}
```

The code block must do something to ensure that the condition is eventually not true!

The `while` Loop

- A counting loop

```
int i=1, sum=0;
while (i <= limit)
{
    sum = sum + i;
    i = i + 1;
}
```

Parentheses are required (just like an "if" statement.)

Condition

Loop Body

Be sure to increment the loop counter

- Condition is evaluated first.
- If it is true
  - loop body is executed.
  - Repeat

The `while` Loop

```
/*
    This program computes the sum of an arithmetic series,
    1 + 2 + 3 + ... + N, where the value of N is supplied
    by the user
*/

#include <stdio.h>

int main()
{
    int limit = 0;
    int sum = 0;
    int i = 1;          /* Loop counter */

    printf( "This program computes the sum of integers 1 to N.\n" );
    printf( "Please enter the value for N: ");
    scanf ("%d", &limit);

    while (i <= limit)
    {
        sum = sum + i;
        i = i + 1;
    }

    printf ("The sum is %d\n", sum);
    getchar();                          // Keep window open
    getchar();
    return 0;
}
```

Program Running

c:\Documents and Settings\Rollins\My Documents\@Courses_Temporary\Program_Design\Examples\demo\debug\demo.exe

```
This program computes the sum of integers 1 to N.
Please enter the value for N: 22
The sum is 253
```

The `while` Loop

- Things to notice

No semicolon at end of line.

```
while (i <= limit)
{
    sum = sum + i;
    i = i + 1;
}
```

No semicolon at end of code block.

Programming Style

```
while (i <= limit)
{
     sum = sum + i;
     i = i + 1;
}
```

Indent the block code three spaces beyond the brackets.

Align the curly brackets, even with the "while".

(This differs from the style in the book.)

Some Questions about `while`

What happens if the condition is not true when we reach the while loop?

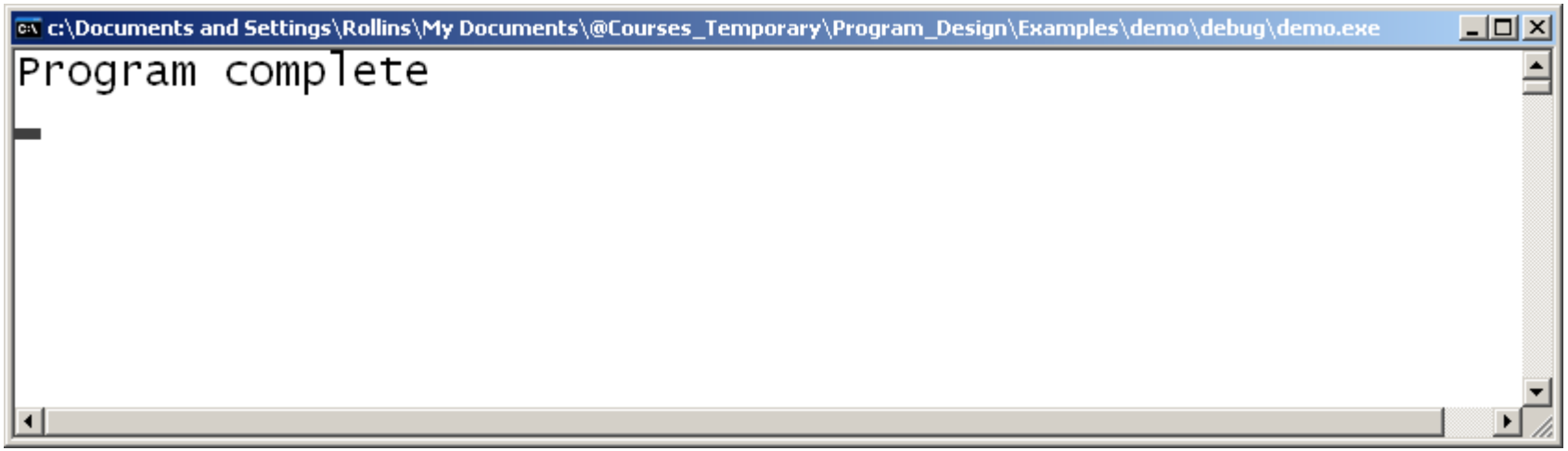
```
int main()
{
    int i = 100;

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

    printf ("Program complete\n");
    getchar();  // Keep window open
    return 0;
}
```

Some Questions about `while`

The code block is not executed at all.



Some Questions about `while`

- What happens if we forget to increment the loop counter?

```
#include <stdio.h>

int main()
{
    int i;
    i = 0;
    while (i < 10)
    {
        printf ("i = %d\n", i);
    }
    printf ("Program complete\n");
    getchar();  // Keep the window open.
    return 0;
}
```

What happens if we forget to increment the loop counter?

c:\Documents and Settings\Rollins\My Documents\@Courses_Temporary\Program_Design\Examples\demo\debug\demo.exe

```
i = 0
i = 0
i = 0
i = 0
i = 0
i = 0
i = 0
i = 0
i = 0
i = 0
```

The code block is repeated indefinitely!

Have to stop it with Ctrl-C

What happens if we forget to increment the loop counter?

- If the code block does not do output, the program will appear to die.
  - It is hung in an endless loop!
- Here again, press Ctrl-C to stop the program.

Some Questions about `while`

- What happens if we incorrectly put a semicolon at the end of the condition?

```
while (i <= limit);
{
    sum = sum + i;
    i = i + 1;
}
```

"Nothing" is repeated indefinitely.

- The program hangs in a silent endless loop.

Some Questions about `while`

- What happens if we omit the curley brackets?

```
while (i <= limit)
    sum = sum + i;
    i = i + 1;
```

This statement is repeated indefinitely.

- Only the first statement following the "while" is repeated.
- In this case, the program hangs in an endless loop.

More Programming Style

- Legally you can write a single statement after the while, with no curly brackets:

```
while (i <= limit)
    printf (“i = %d\n”, i++);
```

- Don’t
- Same reasoning as for “if”
  - It is easy to forget to add the curly brackets if you add another statement to the loop later.

Infinite Loops

- Sometimes we intentionally write an “infinite” loop:

```
while (1)
{
    /* Repeat some action until something happens
       to cause us to stop. */
    if (finished)
    {
        break;
    }
    /* Do other stuff */
}
/* Continue here */
```

Go to next statement after the loop.

Typical example: Get input from user and validate. Keep trying until input is OK.

Infinite Loops

```
#include <stdio.h>
int main()
{
    double t = 0.0;
    while (1)
    {
        printf ("Enter amount of time to simulate: ");
        scanf("%lg", &t);
        if (t > 0.0)
        {
            break;
        }
        printf ("Invalid input %lg\n", t);
        printf ("Please try again\n\n");
    }
    printf ("t is %lg\n", t);
    getchar();  // Keep window open
    getchar();
    return 0;
}
```

Infinite Loops

```
h:\@mydocuments\@courses_2008_fall\program_design\examples\demo\debug\demo.exe
Enter amount of time to simulate: 0
Invalid input 0
Please try again

Enter amount of time to simulate: -22
Invalid input -22
Please try again

Enter amount of time to simulate: 33.3
t is 33.3
```

Infinite Loops

- Sometimes we intentionally write an “infinite” loop:

```
while (1)
{
    /* Repeat some action until something happens
       to cause us to stop. */
    if (finished)
    {
        break;
    }
    /* Do other stuff */
}
/* Continue here */
```

Go to next statement after the loop.

Typical example: Get input from user and validate. Keep trying until input is OK.

Infinite Loops

```
#include <stdio.h>
int main()
{
    double t = 0.0;
    while (1)
    {
        printf ("Enter amount of time to simulate: ");
        scanf("%lg", &t);
        if (t > 0.0)
        {
            break;
        }
        printf ("Invalid input %lg\n", t);
        printf ("Please try again\n\n");
    }
    printf ("t is %lg\n", t);
    getchar();  // Keep window open
    getchar();
    return 0;
}
```

Infinite Loops

```
h:\@mydocuments\@courses_2008_fall\program_design\examples\demo\debug\demo.exe
Enter amount of time to simulate: 0
Invalid input 0
Please try again

Enter amount of time to simulate: -22
Invalid input -22
Please try again

Enter amount of time to simulate: 33.3
t is 33.3
```

Another Run

```
C:\WINDOWS\system32\cmd.exe
Enter amount of time to simulate: abc
Invalid input 0
Please try again

Enter amount of time to simulate: Invalid input 0
Please try again

Enter amount of time to simulate: Invalid input 0
Please try again

Enter amount of time to simulate: Invalid input 0
Please try again

Enter amount of time to simulate: Invalid input 0
Please try again

Enter amount of time to simulate: Invalid input 0
Please try again

Enter amount of time to simulate: Invalid input 0
Please try again

Enter amount of time to simulate: Invalid input 0
```

What's happening here?

What's happening here?

- The "a" stops the scan.
- t keeps its initial value of 0.0.

- "abc" stays in the input buffer.
  - *Poisons the buffer!*

- while loop repeats.

The Cure

- After getting an invalid value, clear the keyboard input buffer.
- A handy system function for this is getchar()
  - Read one character from the keyboard input buffer.

The Cure

```
int main()
{
    double t = 0.0;
    char ch = 0;
    while (1)
    {
        printf ("Enter amount of time to simulate: ");
        scanf("%lg", &t);
        if (t > 0)
        {
            break;
        }
        printf ("Invalid input %lg\n", t);
        printf ("Please try again\n\n");

        while (ch != '\n')  /* Clear keyboard input buffer */
        {
            ch = getchar();
        }
    }
```

An Optimization

- Beloved by *real* C programmers

```
/* Clear keyboard input buffer */
while (getchar() != '\n') ;
```

Do nothing!

Another Form of `while` Loop

```
double t = 0.0;
do
{
    printf ("Enter amount of time to simulate: ");
    scanf("%lg", &t);
    if (t > 0.0)
    {
        break;
    }
    printf ("Invalid input %lg\n", t);
    printf ("Please try again\n\n");
    while (getchar() != '\n');  /* Clear keyboard input buffer */

} while (t <= 0.0);
```

Condition at the end of the loop.

Note semicolon following condition.

do ... while

- Use this form to definitely execute the loop once
  - and repeat only if necessary.

Assignment

- Do the examples from this presentation for yourself
- If anything doesn’t work, or doesn’t make sense
  - ask for help!

End of Presentation

Assignment 2

- You are now ready to do Assignment2
- Hint: Look at clear_input_buffer.c

End of Presentation