The while Looping Structure

CMSC 104, Spring 2014

Christopher S. Marron

(thanks to John Park for slides)

The while Looping Structure

Topics

- The while Loop
- Program Versatility
  - Sentinel Values and Priming Reads
- Checking User Input Using a while Loop

Reading

- Section 3.7

Review: Repetition Structure

- A **repetition structure** allows the programmer to specify that an action is to be repeated while some condition remains true.
- There are three repetition structures in C, the **while** loop, the **for** loop, and the **do-while** loop.

The while Repetition Structure

```
while ( condition ) {
    statement(s)
}
```

The braces are not required if the loop body contains only a single statement. However, they are a good idea and are required by the 104 C Coding Standards.

Example

```
while ( children > 0 ) {
    children = children - 1 ;
    cookies = cookies * 2 ;
}
```

Good Programming Practice

- Always place braces around the body of a while loop.
- Advantages:
  - Easier to read
  - Will not forget to add the braces if you go back and add a second statement to the loop body
  - Less likely to make a semantic error
- Indent the body of a while loop 3 to 4 spaces -- be consistent!

Example

```
while ( input < 0 )
      scanf("%d", &input);
printf("Finally, got something positive\n");
```

Example

```
while ( input < 0 )
    printf("Enter a positive number: ");
    scanf("%d", &input);

printf("Finally, got something positive\n");
```

Another while Loop Example

- Problem: Write a program that calculates the average exam grade for a class of 10 students.
- What are the program inputs?
    - the exam grades
- What are the program outputs?
    - the average exam grade

The Pseudocode

```
<total> = 0
<grade_counter> = 1

While  (<grade_counter> <= 10)
    Display “Enter a grade: ”
    Read <grade>
    <total> = <total> + <grade>
    <grade_counter> = <grade_counter> + 1
End_while
<average> = <total> / 10
Display “Class average is: “, <average>
```

The C Code

```
#include <stdio.h>
int main ( ) {
        int  counter, grade, total, average ;
        total = 0 ;
        counter = 1 ;
        while ( counter <= 10 ) {
            printf (“Enter a grade : “) ;
            scanf (“%d”, &grade) ;
            total = total + grade ;
            counter = counter + 1 ;
        }
        average = total / 10 ;
        printf (“Class average is: %d\n”, average) ;
    return 0 ;
}
```

Versatile?

- How versatile is this program?
- It only works with class sizes of 10.
- We would like it to work with any class size.
- A better way :
  - Ask the user how many students are in the class. Use that number in the condition of the while loop and when computing the average.

New Pseudocode

```
<total> = 0
<grade_counter> = 1



While  (<grade_counter>  <=  10)
    Display "Enter a grade: "
    Read <grade>
    <total> = <total> + <grade>
    <grade_counter> = <grade_counter> + 1
End_while
<average> = <total> / 10
Display "Class average is: ", <average>
```

New Pseudocode

```
<total> = 0
<grade_counter> = 1
**Display “Enter the number of students: “**
**Read <num_students>**
While  (<grade_counter>  <=  10)
    Display “Enter a grade: ”
    Read <grade>
    <total> = <total> + <grade>
    <grade_counter> = <grade_counter> + 1
End_while
<average> = <total> / 10
Display “Class average is: “, <average>
```

New Pseudocode

```
<total> = 0
<grade_counter> = 1

Display "Enter the number of students: "
Read <num_students>
While  (<grade_counter>  <=  **<num_students>**)
    Display "Enter a grade: "
    Read <grade>
    <total> = <total> + <grade>
    <grade_counter> = <grade_counter> + 1
End_while
<average> = <total> / **<num_students>**
Display "Class average is: ", <average>
```

New C Code

```
#include <stdio.h>
int main ( ) {
    int  numStudents, counter, grade, total, average ;
    total = 0 ;
    counter = 1 ;
    printf (“Enter the number of students: “) ;
    scanf (“%d”, &numStudents) ;
    while ( counter <= numStudents) {
        printf (“Enter a grade : “) ;
        scanf (“%d”, &grade) ;
        total = total + grade ;
        counter = counter + 1 ;
    }
    average = total / numStudents ;
    printf (“Class average is: %d\n”, average) ;
    return 0 ;
}
```

Why Bother to Make It Easier?

- Why do we write programs?
  - So the user can perform some task
- The more versatile the program, the more difficult it is to write. BUT it is more useable.
- The more complex the task, the more difficult it is to write. But that is often what a user needs.
- Always consider the user first.

Using a Sentinel Value

- We could let the user keep entering grades and when he’s done enter some special value that signals us that he’s done.
- This special signal value is called a **sentinel value**.
- We have to make sure that the value we choose as the sentinel isn’t a legal value. For example, we can’t use 0 as the sentinel in our example as it is a legal value for an exam score.

The Priming Read

- When we use a sentinel value to control a while loop, we have to get the first value from the user before we encounter the loop so that it will be tested and the loop can be entered.
- This is known as a **priming read**.
- We have to give significant thought to the initialization of variables, the sentinel value, and getting into the loop.

New Pseudocode

```
<total> = 0
<grade_counter> = 0
**Display “Enter a grade: “**
**Read <grade>**
While ( <grade>  !=  -1 )
    <total> = <total> + <grade>
    <grade_counter> = <grade_counter> + 1
    Display “Enter another grade: ”
    Read <grade>
End_while
<average> = <total> / <grade_counter>
Display “Class average is: “, <average>
```

New C Code

```
#include <stdio.h>
int main ( ) {
    int counter, grade, total, average ;

    total = 0 ;
    counter = 0 ;
    printf(“Enter a grade: “) ;
    scanf(“%d”, &grade) ;
    while (grade != -1) {
        total = total + grade ;
        counter = counter + 1 ;
        printf(“Enter another grade: “) ;
        scanf(“%d”, &grade) ;
    }

    average = total / counter ;
    printf (“Class average is: %d\n”, average) ;
    return 0 ;
}
```

Final “Clean” C Code

```
#include <stdio.h>

int main ( ) {
    int counter ;    /* counts number of grades entered */
    int grade ;      /* individual grade                */
    int total;       /* total of all grades             */
    int average ;    /* average grade                   */

    /* Initializations */
    total = 0 ;
    counter = 0 ;

    /* Priming read to get initial grade from user     */
    printf(“Enter a grade: “) ;
    scanf(“%d”, &grade) ;
```

Final “Clean” C Code (con’t)

```
    /* Get grades until user enters -1.  Compute grade total
       and grade count.   */
    while (grade != -1) {
        total = total + grade ;
        counter = counter + 1 ;
        printf(“Enter another grade: “) ;
        scanf(“%d”, &grade) ;
    }

    /* Compute and display the average grade */
    average = total / counter ;
    printf (“Class average is: %d\n”, average) ;

    return 0 ;
}
```

Using a while Loop to Check User Input

```
#include <stdio.h>
int main ( ) {
    int number ;
    printf (“Enter a positive integer :  “) ;
    scanf (“%d”, &number) ;
     while ( number <= 0 ) {
        printf (“\nThat’s incorrect.  Try again.\n”) ;
        printf (“Enter a positive integer:  “) ;
        scanf (“%d”, &number) ;
     }
     printf (“You entered: %d\n”, number) ;
     return 0 ;
}
```