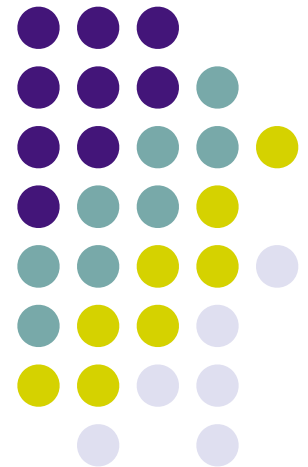


The while Looping Structure

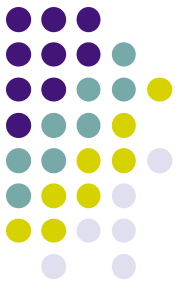
CMSC 104, Spring 2014

Christopher S. Marron

(thanks to John Park for slides)



The while Looping Structure



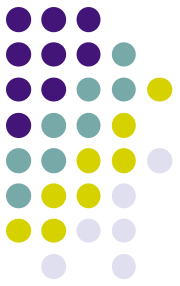
Topics

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

Reading

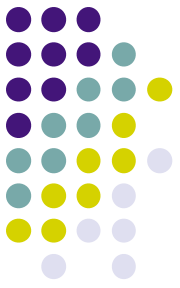
- Section 3.7

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.

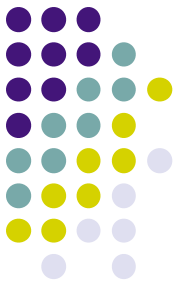
The while Repetition Structure



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

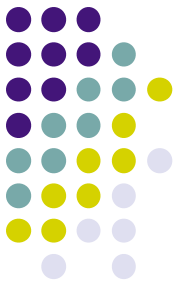
The braces are not required if the loop body contains only a single statement. However, they are a good idea and are required by the 104 C Coding Standards.

Example



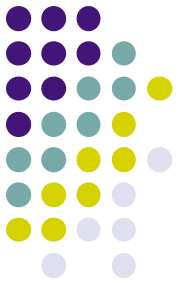
```
while ( children > 0 ) {  
    children = children - 1 ;  
    cookies = cookies * 2 ;  
}
```

Good Programming Practice



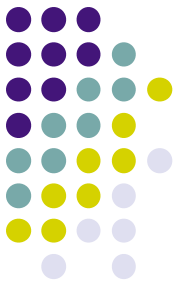
- Always place braces around the body of a while loop.
- Advantages:
 - Easier to read
 - Will not forget to add the braces if you go back and add a second statement to the loop body
 - Less likely to make a semantic error
- Indent the body of a while loop 3 to 4 spaces -- be consistent!

Example

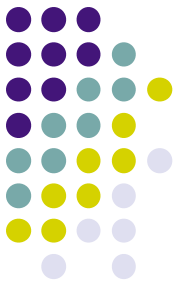


```
while ( input < 0 )
    scanf("%d", &input);
printf("Finally, got something positive\n");
```

Example



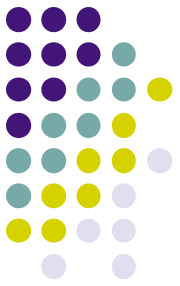
```
while ( input < 0 )  
    printf("Enter a positive number: ");  
    scanf("%d", &input);  
printf("Finally, got something positive\n");
```

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>

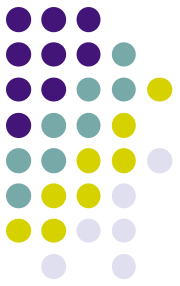
 <grade_counter> = <grade_counter> + 1

End_while

<average> = <total> / 10

Display “Class average is: “, <average>

The C Code



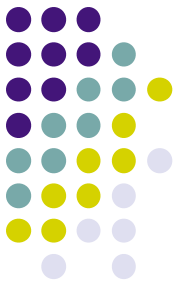
```
1.  #include <stdio.h>
2.  int main ( ) {
3.      int counter, grade, total, average ;
4.      total = 0 ;
5.      counter = 1 ;
6.      while ( counter <= 10 ) {
7.          printf ("Enter a grade : ") ;
8.          scanf ("%d", &grade) ;
9.          total = total + grade ;
10.         counter = counter + 1 ;
11.     }
12.     average = total / 10 ;
13.     printf ("Class average is: %d\n", average) ;
14.     return 0 ;
15. }
```

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>

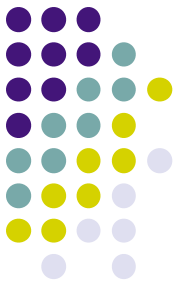
 <grade_counter> = <grade_counter> + 1

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>

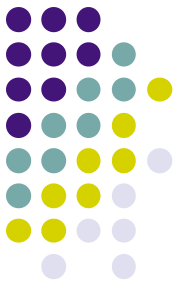
 <grade_counter> = <grade_counter> + 1

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>

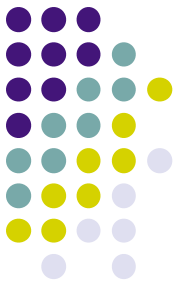
 <grade_counter> = <grade_counter> + 1

End_while

<average> = <total> / <num_students>

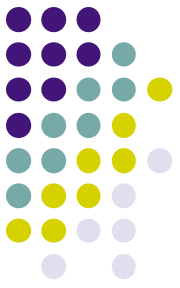
Display “Class average is: “, <average>

New C Code



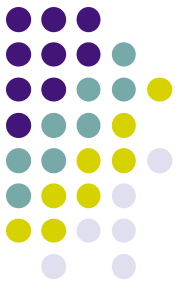
```
1.  #include <stdio.h>
2.  int main ( ) {
3.      int numStudents, counter, grade, total, average ;
4.      total = 0 ;
5.      counter = 1 ;
6.      printf (“Enter the number of students: “) ;
7.      scanf (“%d”, &numStudents) ;
8.      while ( counter <= numStudents) {
9.          printf (“Enter a grade : “) ;
10.         scanf (“%d”, &grade) ;
11.         total = total + grade ;
12.         counter = counter + 1 ;
13.     }
14.     average = total / numStudents ;
15.     printf (“Class average is: %d\n”, average) ;
16.     return 0 ;
17. }
```


Why Bother to Make It Easier?



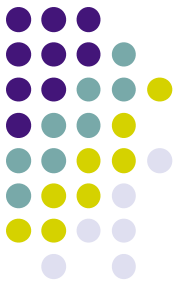
- Why do we write programs?
 - So the user can perform some task
- The more versatile the program, the more difficult it is to write. BUT it is more useable.
- The more complex the task, the more difficult it is to write. But that is often what a user needs.
- Always consider the user first.

Using a Sentinel Value



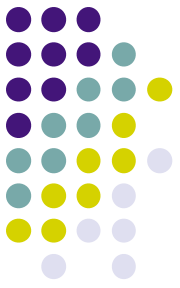
- We could let the user keep entering grades and when he's done enter some special value that signals us that he's done.
- This special signal value is called a **sentinel value**.
- We have to make sure that the value we choose as the sentinel isn't a legal value. For example, we can't use 0 as the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

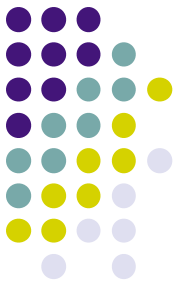
Read <grade>

End_while

<average> = <total> / <grade_counter>

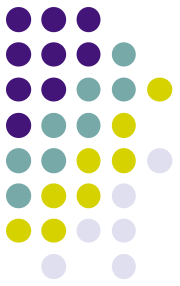
Display “Class average is: “, <average>

New C Code



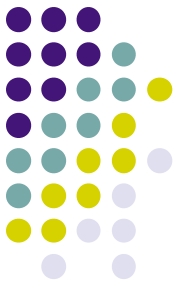
```
1. #include <stdio.h>
2. int main ( ) {
3.     int counter, grade, total, average ;
4.     total = 0 ;
5.     counter = 0 ;
6.     printf("Enter a grade: ") ;
7.     scanf("%d", &grade) ;
8.     while (grade != -1) {
9.         total = total + grade ;
10.        counter = counter + 1 ;
11.        printf("Enter another grade: ") ;
12.        scanf("%d", &grade) ;
13.    }
14.    average = total / counter ;
15.    printf ("Class average is: %d\n", average) ;
16.    return 0 ;
17. }
```

Final “Clean” C Code



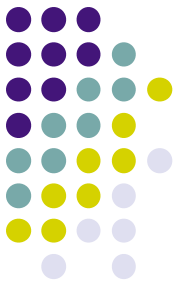
```
1. #include <stdio.h>
2.
3. int main ( ) {
4.     int counter ;    /* counts number of grades entered */
5.     int grade ;     /* individual grade */
6.     int total;      /* total of all grades */
7.     int average ;  /* average grade */
8.
9.     /* Initializations */
10.    total = 0 ;
11.    counter = 0 ;
12.
13.    /* Priming read to get initial grade from user */
14.    printf("Enter a grade: ") ;
15.    scanf("%d", &grade) ;
```

Final “Clean” C Code (con’t)



```
16.  /* Get grades until user enters -1. Compute grade total
17.     and grade count.  */
18.  while (grade != -1) {
19.     total = total + grade ;
20.     counter = counter + 1 ;
21.     printf("Enter another grade: ");
22.     scanf("%d", &grade) ;
23. }
24.
25.  /* Compute and display the average grade */
26.  average = total / counter ;
27.  printf ("Class average is: %d\n", average) ;
28.
29.  return 0 ;
30. }
```

Using a while Loop to Check User Input



```
1.  #include <stdio.h>
2.  int main ( ) {
3.      int number ;
4.      printf ("Enter a positive integer : " ) ;
5.      scanf ("%d", &number) ;
6.      while ( number <= 0 ) {
7.          printf ("\nThat's incorrect. Try again.\n") ;
8.          printf ("Enter a positive integer: " ) ;
9.          scanf ("%d", &number) ;
10.     }
11.     printf ("You entered: %d\n", number) ;
12.     return 0 ;
13. }
```