CS2211b

Software Tools and Systems Programming

Week 9a

Loops

Announcements

- Assignment #2 Released
- Midterm Grade Released
- Midterm Participation Mark Released
- Lab 7 Posted

- **March 7th:** Last day to drop course without penalty
  - Need to talk to your faculty's academic counsellors to drop course.

Announcements

March 7th: Last day to drop course without penalty

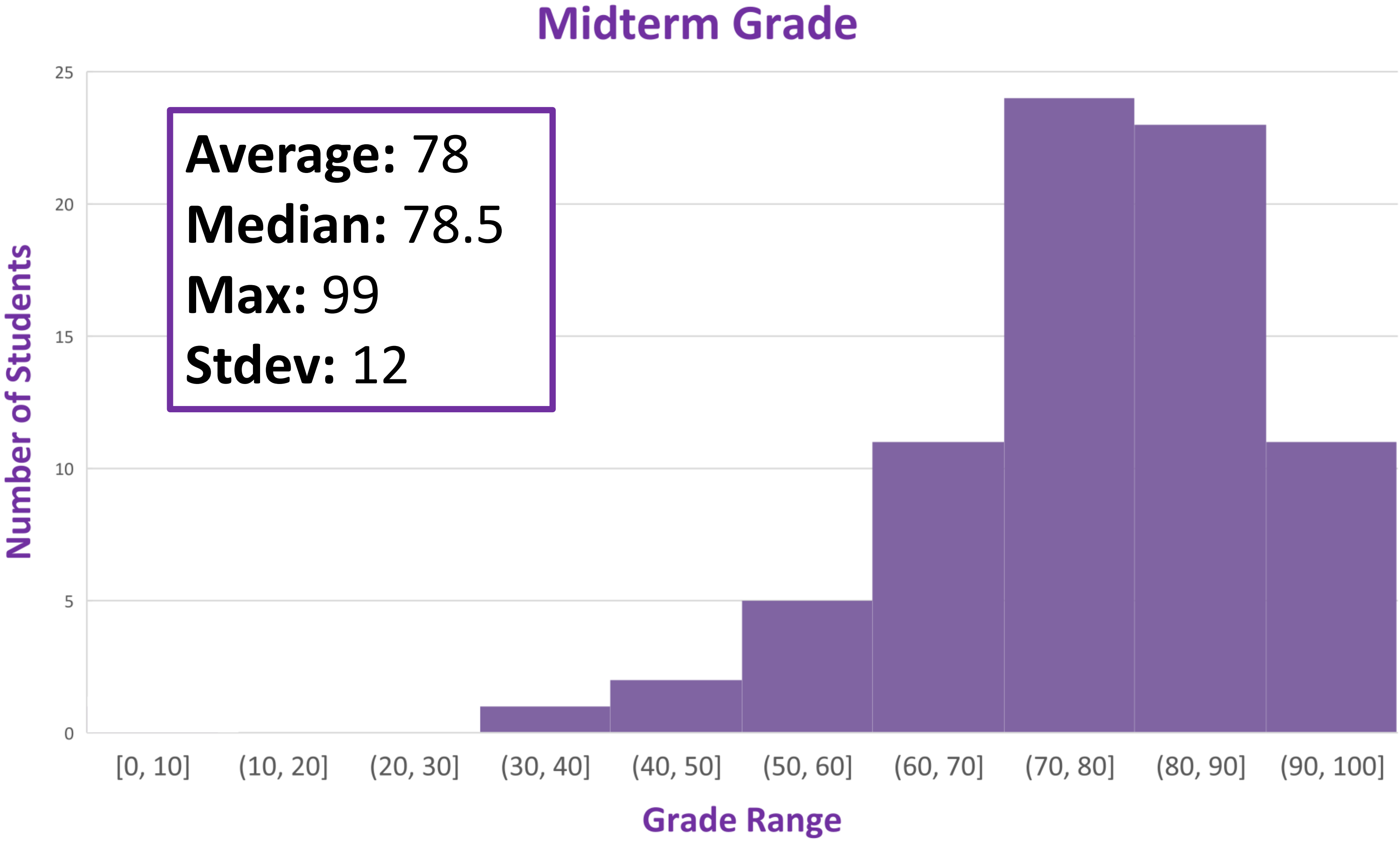
- Need to talk to your faculty's academic counsellors to drop course.

- **To be eligible to receive a passing grade in the course, your total weighted average on the midterm and final exams must be at least 50% and your total average on the assignments at least 40%.**
- **To be eligible to receive a grade of C (60%) or higher (i.e. to be eligible for Honours Programs), your total weighted average on the midterm and final exam must be at least 60% and your total average on the assignments at least 50%.**

Element	Weight
Assignments (4)	20%
Labs (11)	10%
Participation	10%
Midterm	20%
Final Exam	40%

About 35-40% of grade is set now.

Midterm Results



Midterm Grade

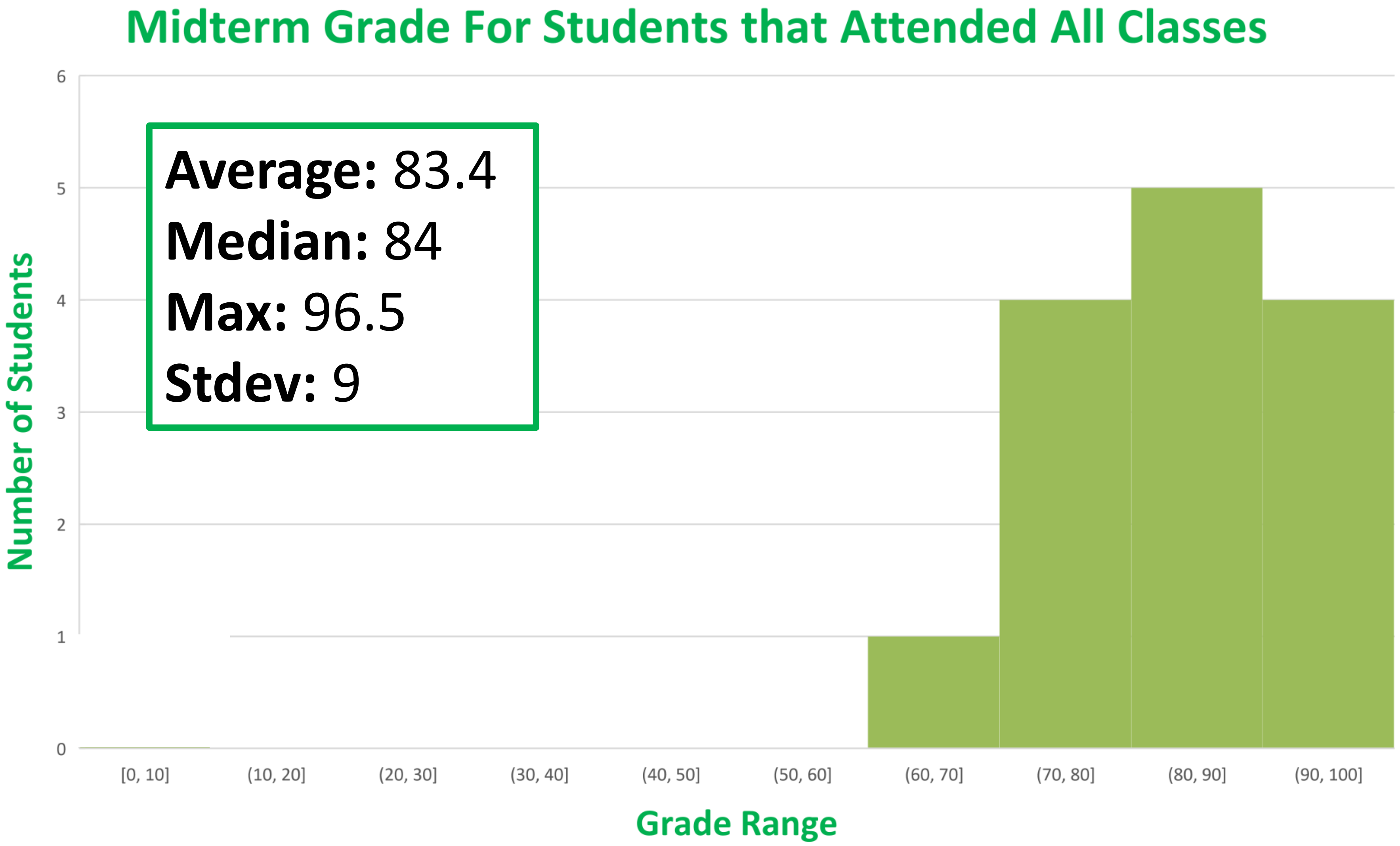
Average: 78

Median: 78.5

Max: 99

Stdev: 12

Midterm Results



Average: 83.4

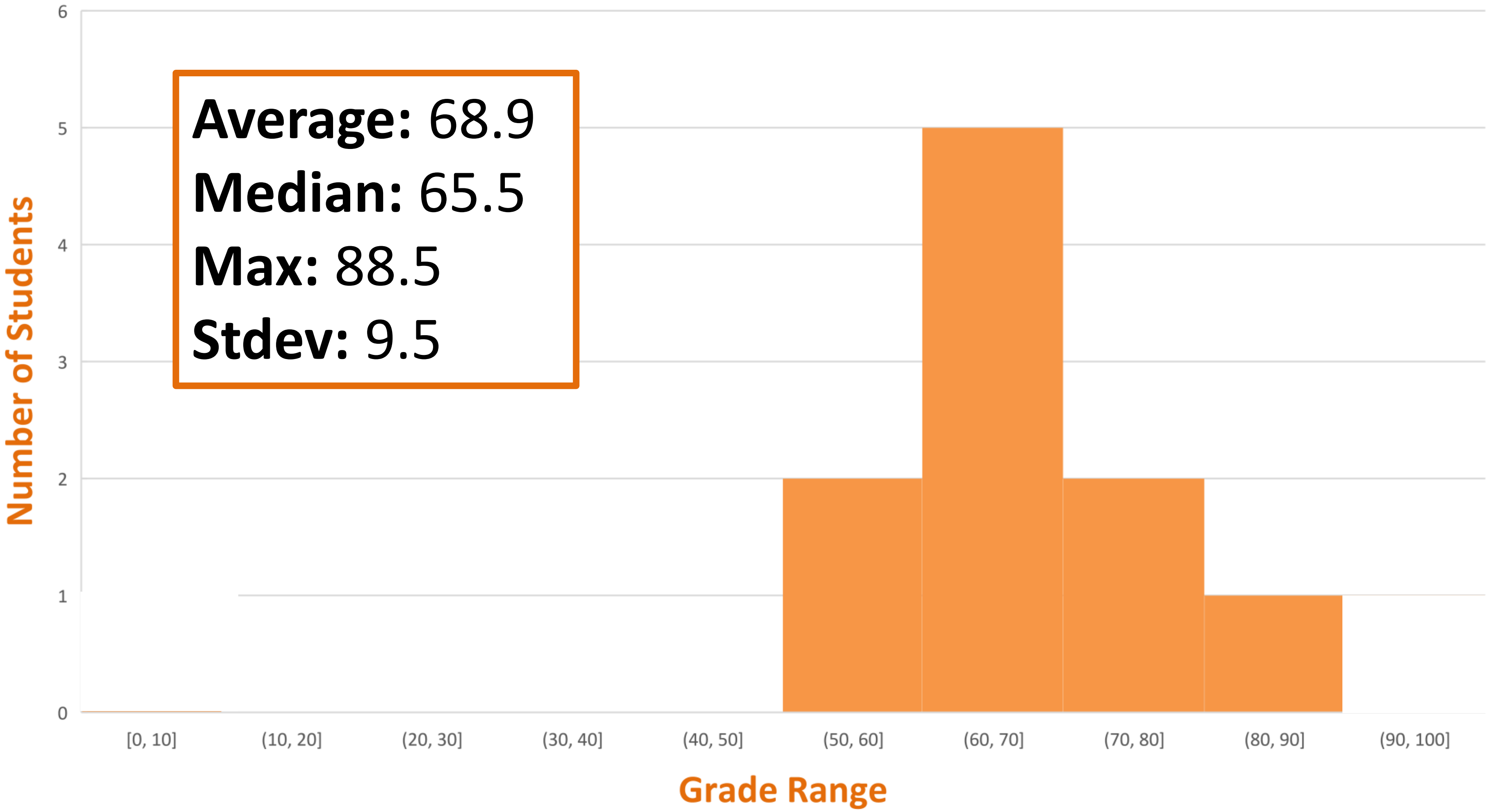
Median: 84

Max: 96.5

Stdev: 9

Midterm Results

Midterm Grade For Students that Attended Less Than 30% of Classes



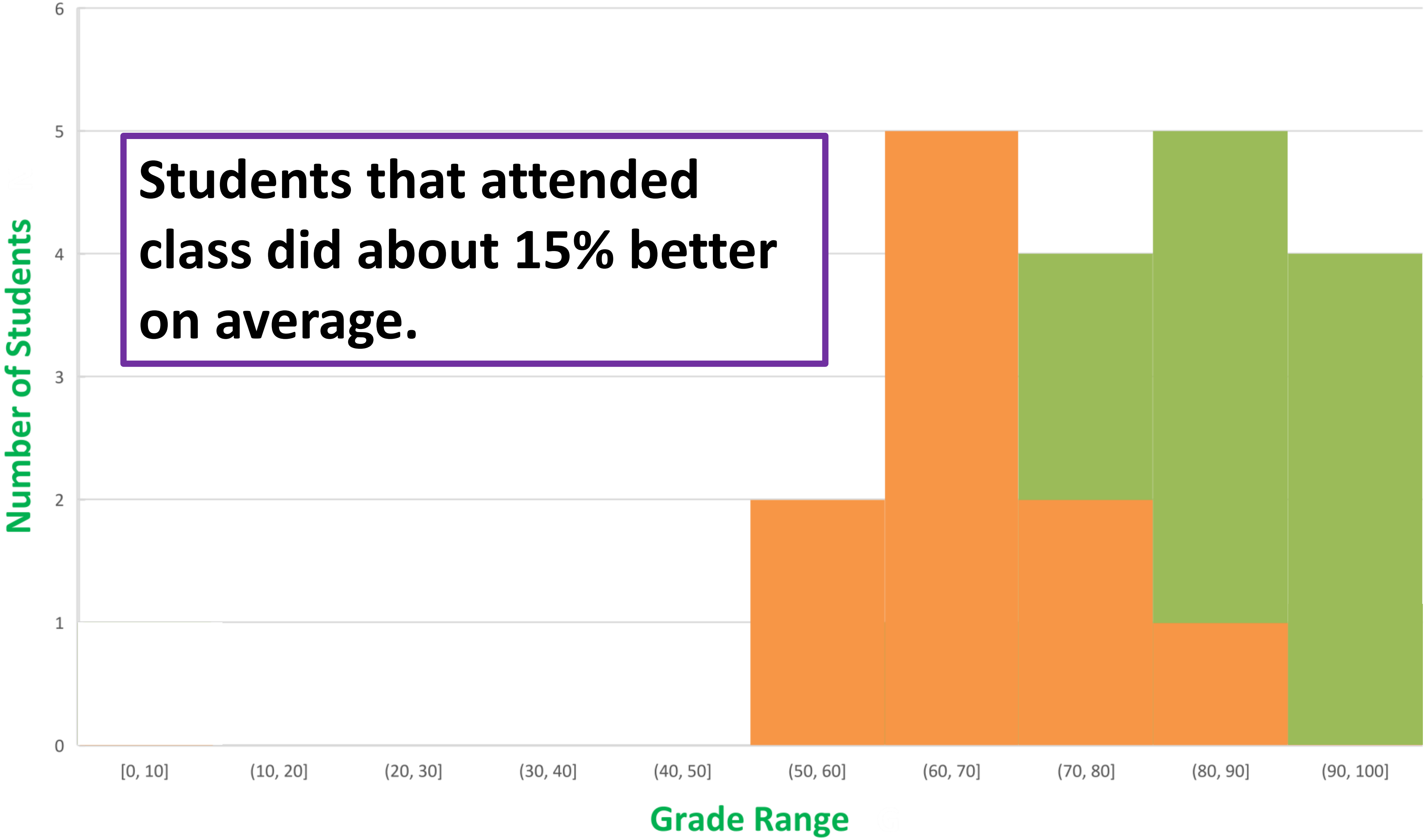
Average: 68.9

Median: 65.5

Max: 88.5

Stdev: 9.5

Midterm Results



Students that attended class did about 15% better on average.

Midterm Pickup & Remarking

- I will have open office hours this week to pick up midterms and ask questions about the midterm (other reasons need an appointment).
- You have 2 weeks to request a question be remarked if you feel the TA made an error.
- Pickup times (at my office MC25):
  - Tuesday 4PM to 5PM
  - Thursday 2PM to 4PM
  - Friday 10:30AM to 12:30PM
- After this week, you can make an office hour appointment to pick up.

Loops

Western Science

While Loop

- The **while** loop repeatedly executes a statement **while** a logical expression is **true**.
- As with the IF statement **true** is defined as being a non-zero value and **false** as zero.
- **Syntax:**

```
while ( expression )
   statement
```

- **Example:**

```
int i = 1;
while ( i < 10 )
   printf("%d\n", i++ * 2);
```

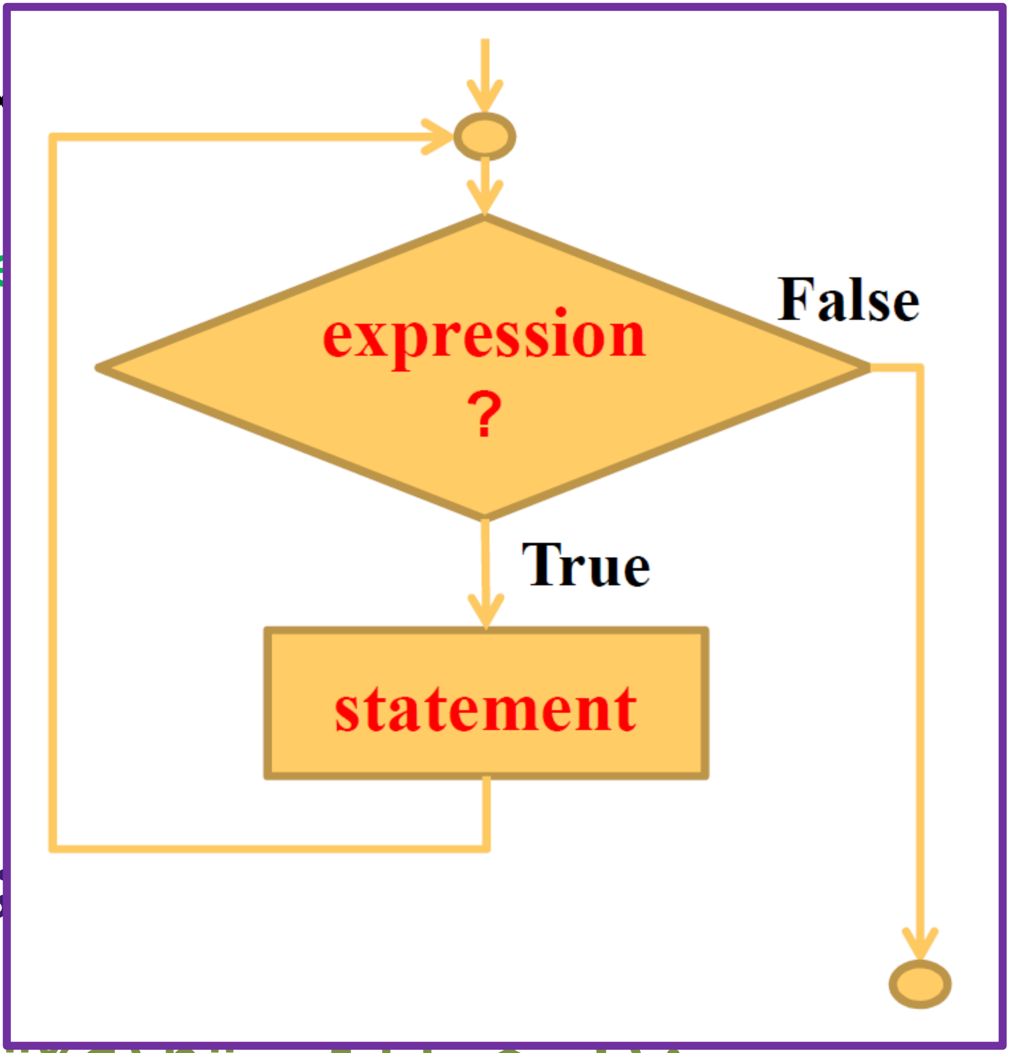
While Loop

- The **while** loop repeatedly ex
  expression is **true**.
- As with the IF statement **true**
  and **false** as zero.
- **Syntax:**

```
while ( expression )
   statement
```

- **Example:**

```
        int i = 1;
        while ( i
           printf("%d\n", i++ * 2);
```



While Loop

- The **while** loop repeatedly executes a statement **while** a logical expression is **true**.
- As with the IF statement **true** is defined as being a non-zero value and **false** as zero.
- **Syntax:**

```
while ( expression )
   statement
```

- **Example:**

```
int i = 1;
while ( i < 10 )
   printf("%d\n", i++ * 2);
```

While Loop

Example:

```
int i = 1;
while ( i < 10 )
   printf("%d\n", i++ * 2);
```

Variable	Value
i	

Output:

While Loop

Example:

```
int i = 1;
while ( i < 10 )
   printf("%d\n", i++ * 2);
```

Variable	Value
i	1

Output:

While Loop

Example:

```
int i = 1;
while ( i < 10 )
   printf("%d\n", i++ * 2);
```

Variable	Value
i	1

Output:

Logical expression i < 10 is evaluated.

i < 10
= 1 < 10
= 1 (True)

Result is non-zero so loop is entred and the statement is executed.

While Loop

Example:

```
int i = 1;
while ( i < 10 )
   printf("%d\n", i++ * 2);
```

Variable	Value
i	1

i * 2 is evaluated, i is not incremented yet as the ++ is a postfix.

i * 2 = 1 * 2 = 2

Output:

While Loop

Example:

```
int i = 1;
while ( i < 10 )
   printf("%d\n", i++ * 2);
```

Variable	Value
i	1

printf outputs the result of the expression: 2

Output:

```
2
```

While Loop

Example:

```
int i = 1;
while ( i < 10 )
   printf("%d\n", i++ * 2);
```

Variable	Value
i	2

i is now incremented by 1 (now equals 2).

Output:

```
2
```

While Loop

Example:

```
int i = 1;
while ( i < 10 )
   printf("%d\n", i++ * 2);
```

Variable	Value
i	2

Output:

```
2
```

Logical expression in while loop is evaluated again.

i < 10
= 2 < 10
= 1 (True)

Statement is executed again.

While Loop

Example:

```
int i = 1;
while ( i < 10 )
   printf("%d\n", i++ * 2);
```

Variable	Value
i	2

Value of i * 2 is output:

2 * 2 = 2 * 2 = 4

Output:

```
2
4
```

While Loop

Example:

```
int i = 1;
while ( i < 10 )
   printf("%d\n", i++ * 2);
```

Variable	Value
i	3

Value of i is incremented by one (now equals 3).

Output:

```
2
4
```

While Loop

Example:

```
int i = 1;
while ( i < 10 )
   printf("%d\n", i++ * 2);
```

Variable	Value
i	3

Logical expression in while loop is evaluated again.

i < 10
= 3 < 10
= 1 (True)

Statement is executed again

Output:

```
2
4
```

While Loop

Example:

```
int i = 1;
while ( i < 10 )
   printf("%d\n", i++ * 2);
```

Variable	Value
i	4

i * 2 = 3 * 2 = 6 is output.

Value of i is incremented by one (now 4).

Output:

```
2
4
6
```

While Loop

Example:

```
int i = 1;
while ( i < 10 )
   printf("%d\n", i++ * 2);
```

Variable	Value
i	4

Loop continues to execute the statement in this manner until i < 10 is no longer true.

Output:

```
2
4
6
```

While Loop

Example:

```
int i = 1;
while ( i < 10 )
    printf("%d\n", i++ * 2);
```

i < 10

= 4 < 10

= 1 (True)

Variable	Value
i	4 5

Loop continues to execute the statement in this manner until i < 10 is no longer true.

Output:

```
2
4
6
8
```

While Loop

Example:

```
int i = 1;
while ( i < 10 )
   printf("%d\n", i++ * 2);
```

i < 10

= 5 < 10

= 1 (True)

Variable	Value
i	~~5~~ 6

Loop continues to execute the statement in this manner until i < 10 is no longer true.

Output:

```
2
4
6
8
10
```

While Loop

Example:

```
int i = 1;
while ( i < 10 )
   printf("%d\n", i++ * 2);
```

i < 10

= 6 < 10

= 1 (True)

Variable	Value
i	~~6~~ 7

Loop continues to execute the statement in this manner until i < 10 is no longer true.

Output:

```
2                  12
4
6
8
10
```

While Loop

Example:

```
int i = 1;
while ( i < 10 )
   printf("%d\n", i++ * 2);
```

i < 10

= 7 < 10

= 1 (True)

Variable	Value
i	~~7~~ 8

Loop continues to execute the statement in this manner until i < 10 is no longer true.

Output:

```
2          12
4          14
6
8
10
```

While Loop

Example:

```
int i = 1;
while ( i < 10 )
   printf("%d\n", i++ * 2);
```

i < 10

= 8 < 10

= 1 (True)

Variable	Value
i	~~8~~ 9

Loop continues to execute the statement in this manner until i < 10 is no longer true.

Output:

```
2          12
4          14
6          16
8
10
```

While Loop

Example:

```
int i = 1;
while ( i < 10 )
   printf("%d\n", i++ * 2);
```

i < 10
= 9 < 10
= 1 (True)

Variable	Value
i	~~9~~ 10

Loop continues to execute the statement in this manner until i < 10 is no longer true.

Output:

```
2          12
4          14
6          16
8          18
10
```

While Loop

Example:

```
int i = 1;
while ( i < 10 )
   printf("%d\n", i++ * 2);
```

i < 10
= 10 < 10
= 0 (False)

Variable	Value
i	~~9~~ 10

i < 10 is no longer true. While loop exits (without executing statement) and moves on to next line of code (after statement).

Output:

```
2           12
4           14
6           16
8           18
10
```

While Loop

- As with IF statements we can use **compound statements** to run execute statements inside a while loop.
- Recall that **compound statements** are surrounded by curly braces { and } and tell the compiler to treat the group of statements as one.
- **Syntax:**

```
while ( expression ) {
   statement1;
   statement2;
   …
   statementN;
}
```

While Loop

Example 1: Write a program to count how many digits are in a positive integer input by the user.

/cs2211/week9/ex1.c

```
#include <stdio.h>

int main() {
        int num, digits = 0;

        printf("Input a positive integer: ");
        scanf("%d", &num);

        while(num > 0) {
                num /= 10;
                digits++;
        }

        printf("%d digits.\n", digits);
        return 0;
}
```

While Loop

Example 1: Write a program

positive integer input by the

/cs2211/week9/ex1.c

Declare and initialize variables.

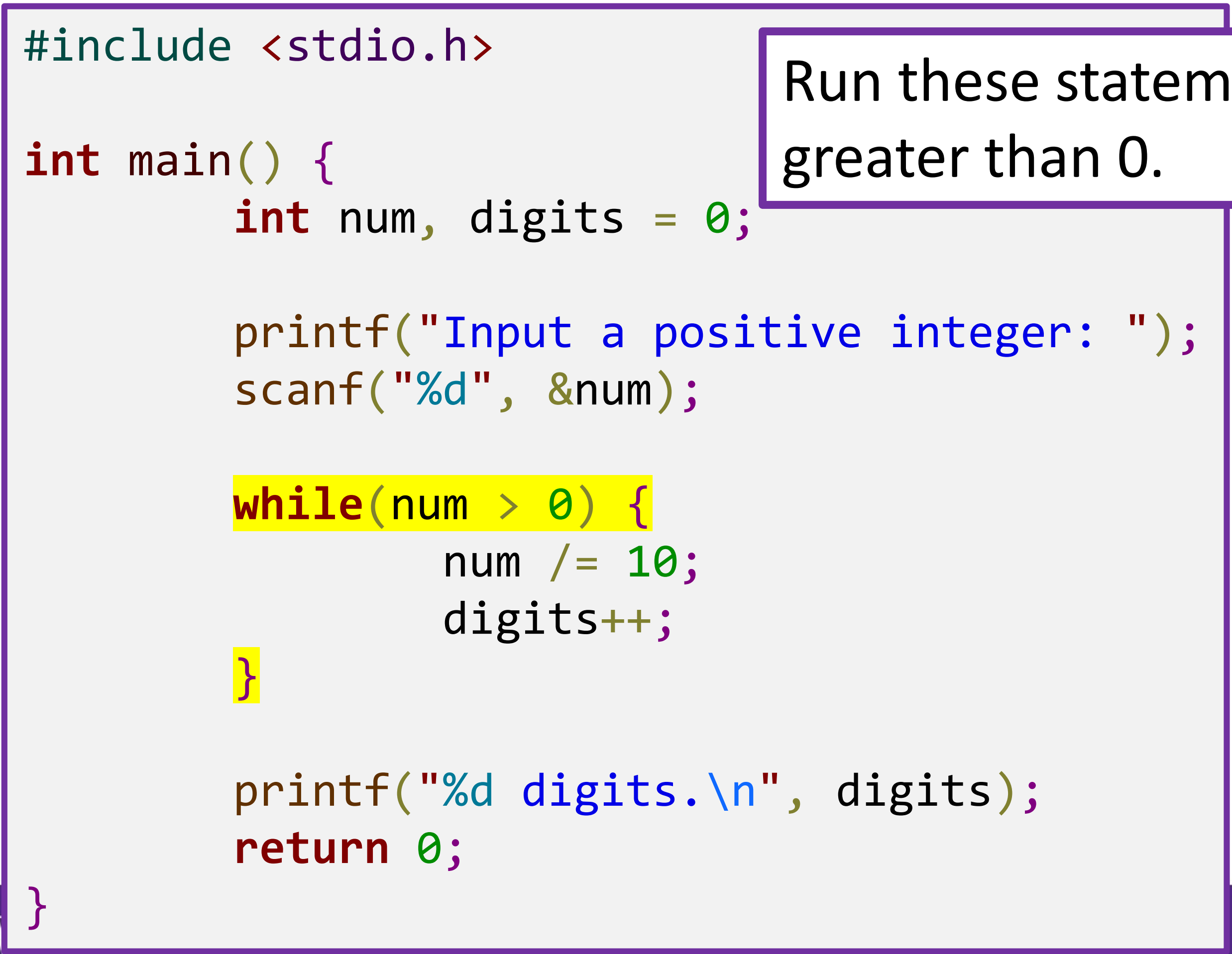
Ask for and receive input from user and store in num.

```
#include <stdio.h>

int main() {
        int num, digits = 0;

        printf("Input a positive integer: ");
        scanf("%d", &num);

        while(num > 0) {
                num /= 10;
                digits++;
        }

        printf("%d digits.\n", digits);
        return 0;
}
```

While Loop

Example 1: Write a program to count how many digits are in a positive integer input by the user.

/cs2211/week9/ex1.c



```c
#include <stdio.h>

int main() {
        int num, digits = 0;

        printf("Input a positive integer: ");
        scanf("%d", &num);

        while(num > 0) {
                num /= 10;
                digits++;
        }

        printf("%d digits.\n", digits);
        return 0;
}
```

Run these statements so long as num is greater than 0.

While Loop

Example 1: Write a program to count how many digits are in a positive integer input by the

/cs2211/week9/ex1.c

```
#include <stdio.h>

int main() {
        int num, digits = 0

        printf("Input a pos
        scanf("%d", &num);

        while(num > 0) {
                num /= 10;
                digits++;
        }

        printf("%d digits.\n", digits);
        return 0;
}
```

Divide num by 10 and store the result back in num (compound assignment).

Note that this will be integer division as the variable num and literal constant 10 are integers.

Example:
1234 / 10 = 123
123 / 10 = 12
12 / 10 = 1
1 / 10 = 0

While Loop

Example 1: Write a program to count how many digits are in a positive integer input by the user.

/cs2211/week9/ex1.c

```
#include <stdio.h>

int main() {
        int num, digits = 0;

        printf("Input a posit
        scanf("%d", &num);

        while(num > 0) {
                num /= 10;
                digits++;
        }

        printf("%d digits.\n", digits);
        return 0;
}
```

Increment `digits` by one.

Keeps track of how many times we had to divide by 10 to get to 0 (i.e. how many digits are in the number).

While Loop

Example 1: Write a program to count how many digits are in a positive integer input by the user.

/cs2211/week9/ex1.c

```c
#include <stdio.h>

int main() {
        int num, digits = 0;

        printf("Input a positive integer: ");
        scanf("%d", &num);

        while(num > 0) {
                num /= 10;
                digits++;
        }

        printf("%d digits.\n", digits);
        return 0;
}
```

Print the result and return.

While Loop

Exa

pos

/cs2

```
#in

int

}
```

Example Input/Output:

```
[dservos5@cs2211b week9]$ ex1
Input a positive integer: 54321
5 digits.

[dservos5@cs2211b week9]$ ex1
Input a positive integer: 1234
4 digits.

[dservos5@cs2211b week9]$ ex1
Input a positive integer: 1
1 digits.

[dservos5@cs2211b week9]$ ex1
Input a positive integer: -123
0 digits.
```

W

Example Input/Output:

```
[dservos5@cs2211b week9]$ ex1
Input a positive integer: -123
0 digits.
```

Exa

pos

/cs2211/week9/ex1.c

```c
#include <stdio.h>

int main() {
        int num, digits = 0;

        printf("Input a positive integer: ");
        scanf("%d", &num);

        while(num > 0) {
                num /= 10;
                digits++;
        }

        printf("%d digits.\n", digits);
        return 0;
}
```

How can we fix this program to work for negative integers?

Exiting Loops

- In addition to waiting for (or making) a logical expression to be false, we can also exit loops using the **break** statement.
- We have seen the **break** statement before when using the **switch** statement.
- The function is the same in loops and causes the loop to immediately exit and for execution to continue after the loop.
- **Simple Example:**

```
while ( i > 0 ) {
   printf("%d", i--);
   if(i % 6 == 0)
      break;
}
```

Exiting Loops

- In addition to waiting for (or making) a logical expression to be false, we can also exit loops using the **break** statement.
- We have seen the **break** statement before when using the **switch** statement.
- The function is the same in loops and causes the loop to immediately exit and for execution to continue after the loop.
- **Simple Example:**

```
while ( i > 0 ) {
   printf("%d", i--);
   if(i % 6 == 0)
      break;
}
```

Will print the values of i down to zero until a number is divisible by 6.

Exiting Loops

- In addition to waiting for (or making) a logical expression to be false, we can also exit loops using the **break** statement.
- We have seen the **break** statement before when using the **switch** statement.
- The function is the same in loops and immediately exit and for execution to
- **Simple Example:**

```
while ( i > 0 ) {
   printf("%d", i--);
   if(i % 6 == 0)
      break;
}
```

If i = 15 this code will print:

15
14
13

Once i = 12, it is divisible by 6 (12 / 6 has no remainder) and the break statement is executed, terminating the loop.

Exiting Loops

- The **break** statement only exits the **innermost** loop.
- **Simple Example**

```
while (…) {
    …
    while (…) {
        …
        break;
        …
    }
    …
}
```

Exiting Loops

- The **break** statement only exits the **innermost** loop.
- **Simple Example**

```
while (…) {
    …
    while (…) {
        …
        break;
        …
    }
    …
}
```

This break statement will only exit this innermost loop, the first while loop will still be running.

Exiting Loops

- C also provides a `continue` statement which causes a loop to restart.
- Variables are unchanged but execution jumps back to the start of the loop.
- **Simple Example:**

```
while ( i-- > 0 ) {
  if(I % 3 == 0)
    continue;
  printf("%d", i);
}
```

Exiting Loops

- C also provides a `continue` statement which causes a loop to restart.
- Variables are unchanged but execution jumps back to the start of the loop.
- **Simple Example:**

```
while ( i-- > 0 ) {
  if(i % 3 == 0)
    continue;
  printf("%d", i);
}
```

Prints the numbers i-1 to 1 but skips any number divisible by 3.

Exiting Loops

- C also provides a `continue` statement which causes a loop to restart.
- Variables are unchanged but execution jumps back to the start of the loop.
- **Simple Example:**

```
while ( i-- > 0 ) {
  if(i % 3 == 0)
    continue;
  printf("%d", i);
}
```

For example if i = 5, the output would be:

```
4
2
1
```

When i = 3, the `continue` statement is executed and the loop restarts before the `printf` line is run.

Infinite Loops

- Normally infinite loops are a bug or undesired and cause a program to lock up (run forever until terminated).
- In some cases, we may wish to intentionally create an infinite loop if we will be exiting them manually with a `break` or `return` statement.
- We can create an infinite loop with a while loop simply by giving it a non-zero literal constant.
- **Simple Example:**

```
while ( 1 )
  printf("Hello World!\n");
```

Infinite Loops

- Normally infinite loops are a bug or undesired and cause a program to lock up (run forever until terminated).
- In some cases, we may wish to intentionally create an infinite loop if we will be exiting them manually with a **break** or **return** statement.
- We can create an infinite loop with a while loop simply by giving it a non-zero literal constant.
- **Simple Example:**

Will keep printing "Hello World!" forever (until you terminate the program).

```
while ( 1 )
   printf("Hello World!\n");
```

Infinite Loops + Break Statement

Example 2: Keep inputting positive integers from the user until they input a negative number. Return the average of the numbers.

/cs2211/week9/ex2.c

```c
#include <stdio.h>

int main() {
        int sum, n, i = 0;

        while(1) {
                printf("Input Number: ");
                if(scanf("%d", &n) != 1) {
                        printf("Bad Number!\n");
                        continue;
                } else if(n < 0)
                        break;

                sum += n;
                i++;
        }

        printf("The average is %f\n", (float) sum / i);
        return 0;
}
```

Infinite Loops + Break Statement

Example 2: Keep inputting positive integers from the user until they input a negative number. Return the average of the numbers.

/cs2211/week9/ex2.c

Infinite loop, will keep running until we execute a **break** statement.

```
#include <stdio.h>

int main() {
        int sum, n, i = 0;

        while(1) {
                printf("Input Number: ");
                if(scanf("%d", &n) != 1) {
                        printf("Bad Number!\n");
                        continue;
                } else if(n < 0)
                        break;

                sum += n;
                i++;
        }

        printf("The average is %f\n", (float) sum / i);
        return 0;
}
```

Infinite Loops + Break Statement

Example 2: Keep inputting positive integers from the user until they input a negative number. Return the average of the numbers.

/cs2211/week9/ex2.c

```
#include <stdio.h>

int main() {
        int sum, n, i = 0;

        while(1) {
                printf("Input Number: ");
                if(scanf("%d", &n) != 1) {
                        printf("Bad Number!\n");
                        continue;
                } else if(n < 0)
                        break;

                sum += n;
                i++;
        }

        printf("The average is %f\n", (float) sum / i);
        return 0;
}
```

Input an integer from the user. If `scanf` fails to read in an integer (e.g. it gets the text "duck"), we will print "Bad number!" and restart the while loop (using the `continue` statement).

Infinite Loops + Break Statement

Example 2: Keep inputting positive integers from the user until they input a negative number. Return the average of the numbers.

/cs2211/week9/ex2.c

```c
#include <stdio.h>

int main() {
        int sum, n, i = 0;

        while(1) {
                printf("Input Number: ");
                if(scanf("%d", &n) != 1) {
                        printf("Bad Number!\n");
                        continue;
                } else if(n < 0)
                        break;

                sum += n;
                i++;
        }

        printf("The average is %f\n", (float) sum / i);
        return 0;
}
```

Sounds good in theory, but in practice this has some issues that we will see in a bit.

Infinite Loops + Break Statement

Example 2: Keep inputting positive integers from the user until they input a negative number. Return the average of the numbers.

/cs2211/week9/ex2.c

```c
#include <stdio.h>

int main() {
        int sum, n, i = 0;

        while(1) {
                printf("Input Number: ");
                if(scanf("%d", &n) != 1) {
                        printf("Bad Number!\n");
                        continue;
                } else if(n < 0)
                        break;

                sum += n;
                i++;
        }

        printf("The average is %f\n", (float) sum / i);
        return 0;
}
```

If the number that was input (n) is less than zero (negative) we execute the break statement to terminate the loop.

Infinite Loops + Break Statement

Example 2: Keep inputting positive integers from the user until they input a negative number. Return the average of the numbers.

If the input was valid we will add it to our sum and increment our counter `i` (so we know how many numbers were input).

/cs2211/week9/ex2.c

```
#include <stdio.h>

int main() {
        int sum, n, i = 0;

        while(1) {
                printf("Input Number: ");
                if(scanf("%d", &n) != 1) {
                        printf("Bad Number!\n");
                        continue;
                } else if(n < 0)
                        break;

                sum += n;
                i++;
        }

        printf("The average is %f\n", (float) sum / i);
        return 0;
}
```

Infinite Loops + Break Statement

Example 2: Keep inputting positive integers from the user until they input a negative number. Return the average of the numbers.

/cs2211/week9/ex2.c

```
#include <stdio.h>

int main() {
        int sum, n, i = 0;

        while(1) {
                printf("Inpu
                if(scanf("%
                        printf("Bad Number!\n");
                        continue;
                } else if(n < 0)
                        break;

                sum += n;
                i++;
        }

        printf("The average is %f\n", (float) sum / i);
        return 0;
}
```

Once the loop has completed, we calculate the average using the sum and our count of the number of integers inputted (`i`).

Note that we need to cast `sum` to float to ensure this is not integer division.

Infinite Loops + Break Statement

Example 2: Keep input

input a negative numb

/cs2211/week9/ex2.c

```
#include <stdio.h>

int main() {
        int sum, n, i

        while(1) {
                print
                if(sc


                } els
                        break;

                sum += n;
                i++;
        }

        printf("The average is %f\n", (float) sum / i);
        return 0;
}
```

Example Input/Output:

```
[dservos5@cs2211b week9]$ ex2
Input Number: 5
Input Number: 10
Input Number: 15
Input Number: 0
Input Number: -1
The average is 7.500000
```

Infinite Loops + Break Statement

Example 2: Keep inputting positive integers from the user until they input a negative numb

/cs2211/week9/ex2.c

Something went very wrong!

```
                if(sc

                } els

                sum +
                i++;
        }

        printf("The a
        return 0;
}
```

What about invalid input like "duck"?

```
[dservos5@cs2211b week9]$ ex2
Input Number: duck
Input Number: Bad Number!
Input Number: Bad Number!
Input Number: Bad Number!
Input Number: Bad Number!
Input Number: Bad Number!
Input Number: Bad Number!
Input Number: Bad Number!
Input Number: Bad Number!
Input Number: Bad Number!
Input Number: Bad Number!
Input Number: Bad Number!
Input Number: Bad Number!
Input Number: Bad Number!
```

Infinite Loops + Break Statement

Input Buffer:

d	u	c	k

The Issue/Error:

1. `scanf` attempts to read an integer but encounters the character 'd'.
2. This causes `scanf` to abort and leave the 'd' character on the input buffer.
3. `scanf` returns 0 (as it read/matched zero format specifiers) and the if statement causes "Bad Number!" to be output and the loop to restart (`continue` statement).
4. `scanf` is called again to read an integer but the 'd' is still on the input buffer and aborts.
5. Steps 1-4 keep repeating forever as the 'd' is never removed from the input buffer. This causes an infinite loop that prints "Input Number: Bad Number!".

Infinite Loops + Break Statement

Input Buffer:

d	u	c	k

The Solution:

- We need to clear the input buffer if scanf aborts.
- We can use `getchar` to remove a single character from the input buffer.
- Will need to remove more than one character (e.g. 'u' would be next character and would also cause the same issues).
- Can use a second `while` loop to keep calling `getchar` until we reach the end of a line (\n).

Infinite Loops + Break Statement

Example 2: Keep inputting positive integers from the user until they input a negative number. Return the average of the numbers.

/cs2211/week9/ex2b.c

```c
int main() {
        int sum, n, i = 0;

        while(1) {
                printf("Input Number: ");
                if(scanf("%d", &n) != 1) {
                        printf("Bad Number!\n");
                        while(getchar() != '\n');
                        continue;
                } else if(n < 0)
                        break;

                sum += n;
                i++;
        }

        printf("The average is %f\n", (float) sum / i);
        return 0;
}
```

Infinite Loops + Break Statement

Example 2: Keep inputting positive integers from the user until they input a negative number. Return the average of the numbers.

/cs2211/week9/ex2b.c

```c
int main() {
        int sum, n, i = 0;

        while(1) {
                printf("Input Number: ");
                if(scanf("%d", &n) != 1) {
                        printf("Bad Number!\n");
                        while(getchar() != '\n');
                        continue;








        }

        prin
        retu
}
```

Keeps calling `getchar()` until `getchar()` returns a line break (\n).

In this case, the while loop has no statement as it is immediately followed by a semicolon (;).

Will remove all characters from the input buffer until it encounters a line break (which it also removes).

Infinite Loops + Break Statement

Example 2:

input a neg

/cs2211/week9

```
int main()
        int

        whi








        }

        printf("The average is %f\n", (float) sum / i);
        return 0;
}
```

Example Input/Output:

```
[dservos5@cs2211b week9]$ ex2b
Input Number: duck
Bad Number!
Input Number: 10
Input Number: cat
Bad Number!
Input Number: 13
Input Number: 17
Input Number: This is not a valid number!
Bad Number!
Input Number: .1234
Bad Number!
Input Number: 7
Input Number: -123
The average is 11.750000
```

Do While Loop

- The **do while** loop functions similarly to a **while** loop but the logical expression is checked at the end of the loop rather than the beginning.
- A do while loop is always executed at least once.
- **Syntax:**

```
do statement while ( expression );
```

- **Example:**

```
do {
   scanf("%d", &n);
   sum += n;
} while(n > 0);
```

Do While Loop

- The do while loop functions similarly to a while loop but the logical expression is checked at the end of the loop rather than the beginning.
- A do while loop is always executed at least once.
- **Syntax:**

```
do statement while ( expression );
```

- **Example:**

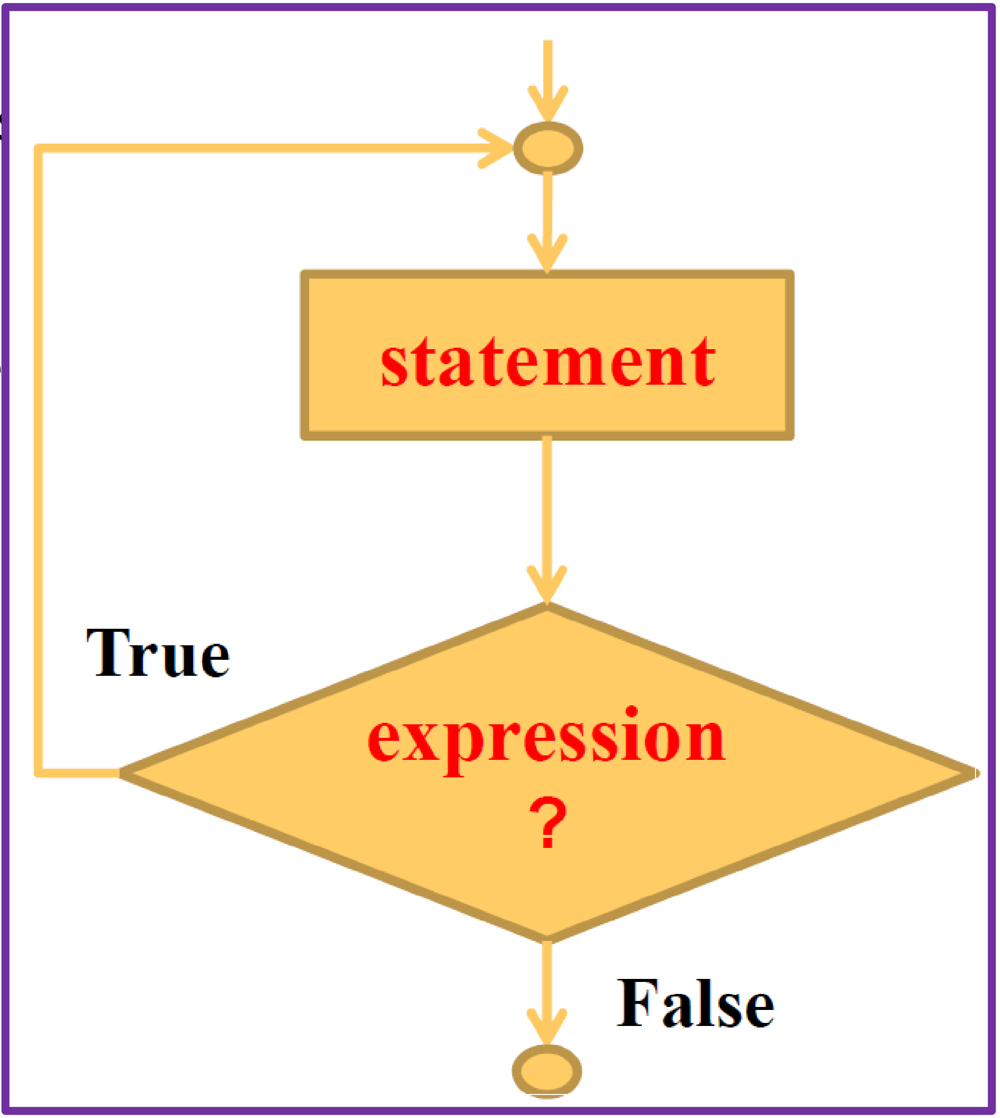
```
do {
   scanf("%d", &n);
   sum += n;
} while(n > 0);
```

Must have a semicolon at the end (unlike other loops).

Do While Loop

- The **do while** loop functions s
logical expression is checked
the beginning.
- A do while loop is always exe
- **Syntax:**

```
do
  statement
while ( expression );
```



Do While Loop

Keeps reading in numbers and adding them to `sum` until the number is negative.

Note that this adds that negative number to the `sum`. It will also always attempt to read in a number even if the value of `n` starts off at -1 (as do loop always run at least once).

- **Example:**

```
do {
   scanf("%d", &n);
   sum += n;
} while(n > 0);
```

Do While Loop

Example 3: Write a program read in integers from the user until a negative number is input. Return the largest number input.

/cs2211/week9/ex3.c

```c
#include <stdio.h>

int main() {
        int n, max = -1;

        do {
                printf("Input Number: ");
                scanf("%d", &n);

                if (n > max)
                        max = n;
        } while(n >= 0);

        printf("Largest number is: %d\n", max);
}
```

Do While Loop

Example 3: Write a program read in integers from the user until a negative number is input. Return the largest number input.

/cs2211/week9/ex3.c

```c
#include <stdio.h>

int main() {
        int n, max = -1;

        do {
                printf("Input Number: ");
                scanf("%d", &n);

                if (n > max)
                        max = n;
        } while(n >= 0);

        printf("Largest number is: %d\n", max);
}
```

Start max at a negative number, lower than lowest possible allowed number.

Do While Loop

Example 3: Write a prog
negative number is inpu

/cs2211/week9/ex3.c

```
#include <stdio.h>

int main() {
        int n, max = -1;

        do {
                printf("Input Number: ");
                scanf("%d", &n);

                if (n > max)
                        max = n;
        } while(n >= 0);

        printf("Largest number is: %d\n", max);
}
```

Enter do loop.

No condition or expression is checked at this point so it is ok that n is uninitialized.

This loop will always run at least once.

Do While Loop

Example 3: Write a program read in integers from the user until a negative number is input. Return the largest number input.

/cs2211/week9/ex3.c

```
#include <stdio.h>

int main() {
        int n, max = -1;

        do {
                printf("Input Number: ");
                scanf("%d", &n);

                if (n > max)
                        max = n;
        } while(n >= 0);

        printf("Largest number is: %d\n", max);
}
```

Read in an integer.

Note that we are not doing any error checking this time.

Do While Loop

Example 3: Write a program read in integers from the user until a negative number is input. Return the largest number input.

/cs2211/week9/ex3.c

```
#include <stdio.h>

int main() {
        int n, max = -1;

        do {
                printf("Input Number: ");
                scanf("%d", &n);

                if (n > max)
                        max = n;
        } while(n >= 0);

        printf("Largest number is: %d\n", max);
}
```

If the number we read in (n) is larger than the value of max, we know that we just encountered a new largest number so we update the value of max to n.

Do While Loop

Example 3: Write a program read in integers from the user until a negative numbe

/cs2211/week9/ex3.

```
#include <stdio

int main() {
        int n,

        do {




                if (n > max)
                        max = n;
        } while(n >= 0);

        printf("Largest number is: %d\n", max);
}
```

The logical expression `n >= 0` is checked at the end of the loop.

If the expression is true (returns 1), the loop will restart. If the expression is false (returns 0) the loop will terminate and the next line will be executed (in this case the printf that outputs the largest number).

Do While Loop

Example 3: Write a program read in integers from the user until a negative number is input. Return the largest number input.

/cs2211/week9/ex3.c

```c
#include <stdio.h>

int main() {
        int n, max = -1;

        do {
                printf("Input Number: ");
                scanf("%d", &n);

                if (n > max)
                        max = n;
        } while(n >= 0);

        printf("Largest number is: %d\n", max);
}
```

After the loop has completed, the value of max (the largest number input) is output to the user.

Do While Loo

Example 3: Write a progr
negative number is input.

/cs2211/week9/ex3.c

```
#include <stdio.h>

int main() {
        int n, max = -1;

        do {
                printf("I
                scanf("%d", &n);

                if (n > max)
                        max = n;
        } while(n >= 0);

        printf("Largest number is: %d\n", max);
}
```

Example Input/Output:

```
[dservos5@cs2211b week9]$ ex3
Input Number: 40
Input Number: 25
Input Number: 3
Input Number: 100
Input Number: 97
Input Number: -1
Largest number is: 100
```

Do While Loop

Example 3: Write a program read in integers from the user until a negative number is input. Return the largest number input.

/cs2

```
#i

in
        if (n > max)
            max = n;
    } while(n >= 0);

    printf("Largest number is: %d\n", max);
}
```

For practice (at home):

Try to do the following to example 3:

1. Add error checking. Number must be an integer (not a letter for example). Output an error if no numbers are input.
2. Find the smallest number rather than the largest.

For Loop

- Ideal loop for "counting". Loops that run from x to y sequentially (e.g. 1, 2, 3, 4 … or 2, 4, 8, …. or 5, 4, 3, 2 …).
- Normally have an index that keeps track of the iteration the loop is on. Index is commonly incremented or decremented each iteration of the loop.
- **Syntax:**

```
for(initial; condition; increment)
   statement
```

For Loop

- Ideal loop for "counting". Loops that run from x to y sequentially (e.g. 1, 2, 3, 4 … or 2, 4, 8, …. or 5, 4, 3, 2 …).
- Normally have an index that keeps track of the iteration the loop is on. Index is commonly incremented or decremented each iteration of the loop.
- **Syntax:**

```
for(initial; condition; increment)
   statement
```

Initial: This expression is evaluated only once at the start of the for loop. Commonly used for initializing an index variable. For example, `i = 1`

For Loop

- Ideal loop for "counting". Loops that run from x to y sequentially (e.g. 1, 2, 3, 4 … or 2, 4, 8, …. or 5, 4, 3, 2 …).
- Normally have an index that keeps track of the iteration the loop is on. Index is commonly incremented or decremented each iteration of the loop.
- **Syntax:**

```
for(initial; condition; increment)
   statement
```

Condition: This expression is evaluated at the beginning of each loop. The for loop will keep running so long as this expression evaluates to true (1). Just like in a while loop.

For Loop

- Ideal loop for "counting". Loops that run from x to y sequentially (e.g. 1, 2, 3, 4 … or 2, 4, 8, …. or 5, 4, 3, 2 …).
- Normally have an index that keeps track of the iteration the loop is on. Index is commonly incremented or decremented each iteration of the loop.
- **Syntax:**

```
for(initial; condition; increment)
  statement
```

Increment: This expression is executed at the end of the loop. Commonly used for incrementing or decrementing the index variable. For example, `i++` or `i--`.

For Loop

- Ideal loop for "counting". Loops that run from x to y sequentially (e.g. 1, 2, 3, 4 … or 2, 4, 8, …. or 5, 4, 3, 2 …).
- Normally have an index that keeps track of the iteration the loop is on. Index is commonly incremented or decremented each iteration of the loop.
- **Syntax:**

```
for(initial; condition; increment)
   statement
```

Statement: The statement to be run by the for loop while the condition is true. You can use a compound statement to include a group of statements as with other loops and IF statements.

For Loop

Simple Examples:

```
int i;
for(i = 1; i <= 10; i++)
  printf("%d\n", i);
```

```
int i;
for(i = 10; i > 0; i--)
  printf("%d\n", i);
```

For Loop

Simple Examples:

Initializes i to 1 at start of loop (is only called once).

```
int i;
for(i = 1; i <= 10; i++)
  printf("%d\n", i);
```

```
int i;
for(i = 10; i > 0; i--)
  printf("%d\n", i);
```

For Loop

Simple Examples:

```
int i;
for(i = 1; i <= 10; i++)
   printf("%d\n", i);
```

Expression is checked each time the loop restarts.

Loop will keep running while i is greater than or equal to 10

```
int i;
for(i = 10; i > 0; i--)
   printf("%d\n", i);
```

For Loop

Simple Examples:

Value of i is incremented by 1 each time the loop restarts

```
int i;
for(i = 1; i <= 10; i++)
  printf("%d\n", i);
```

```
int i;
for(i = 10; i > 0; i--)
  printf("%d\n", i);
```

For Loop

Simple Examples:

```
int i;
for(i = 1; i <= 10; i++)
  printf("%d\n", i);
```

Prints the current value of i.

```
int i;
for(i = 10; i > 0; i--)
  printf("%d\n", i);
```

For Loop

Simple Examples:

This loop prints the value 1 to 10 (inclusive) to the screen in order.

The value of i after the loop runs is 11.

```
int i;
for(i = 1; i <= 10; i++)
  printf("%d\n", i);
```

```
int i;
for(i = 10; i > 0; i--)
  printf("%d\n", i);
```

For Loop

Simple Examples:

This loop is similar to the last but prints the values 10 to 1 (inclusive) in order.

The value of i starts at 10 and is decremented by one each iteration of the loop. The value of i is 0 after the loop is finished.

```
int i;
for(i = 10; i > 0; i--)
  printf("%d\n", i);
```

For Loop

C89 vs. C99

Important for Assignments #3 and #4!!

In C89 the following is invalid and will cause an error:

```
for(int i = 1; i <= 10; i++)
   printf("%d\n", i);
```

- **You are not allowed to declare a variable in the initial expression.**
- **This will cause an error on the course server by default.**
- **Many students have lost marks on their assignments in the past for not testing this on the course server (it will only work with the `-std=c99` or `-std=gnu99` options).**
- **This is valid syntax in C99, but you must let the TA know you are using C99 in your comments.**

For Loop

Example 4: Write a program that prints out the ASCII table (letters space to 'z').

/cs2211/week9/ex4.c

```c
#include <stdio.h>

int main() {
        int i;

        printf("ASCII TABLE\n");
        for(i = ' '; i <= 'z'; i++) {
                printf("%10d%10c\n", i, i);
        }

        return 0;
}
```

For Loop

Example 4: Write a program that prints out the ASCII table (letters space to 'z').

/cs2211/week9/ex4.c

```
#include <stdio.h>

int main() {
        int i;

        printf("ASCII TABLE\n");
        for(i = ' '; i <= 'z'; i++) {
                printf("%10d%10c\n", i, i);
        }

        return 0;
}
```

Initialize i to ' ' (equals 32).

For Loop

Example 4: Write a program that prints out the ASCII table (letters space to 'z').

/cs2211/week9/ex4.c

```c
#include <stdio.h>

int main() {
        int i;

        printf("ASCII TABLE\n");
        for(i = ' '; i <= 'z'; i++) {
                printf("%10d%10c\n", i, i);
        }

        return 0;
}
```

Keep running the loop while i is less than or equal to 'z' (122).

For Loop

Example 4: Write a program that prints out the ASCII table (letters space to 'z').

/cs2211/week9/ex4.c

```
#include <stdio.h>

int main() {
        int i;

        printf("ASCII TABLE\n");
        for(i = ' '; i <= 'z'; i++) {
                printf("%10d%10c\n", i, i);
        }

        return 0;
}
```

Increment the value of i by one at the end of the loop.

For Loop

Example 4: Write a program that prints out the ASCII table (letters space to 'z').

/cs2211/week9/ex4.c

```
#include <stdio.h>

int main() {
        int i;

        printf("ASCII TABLE\n");
        for(i = ' '; i <= 'z'; i++) {
                printf("%10d%10c\n", i, i);
        }

        return 0;
}
```

Print i as an integer with a minimum width of 10.

For Loop

Example 4: Write a program that prints out the ASCII table (letters space to 'z').

/cs2211/week9/ex4.c

```c
#include <stdio.h>

int main() {
        int i;

        printf("ASCII TABLE\n");
        for(i = ' '; i <= 'z'; i++) {
                printf("%10d%10c\n", i, i);
        }

        return 0;
}
```

Print i as a character with a minimum width of 10.

For Loo

Example 4: V (letters space to 'z').

/cs2211/week9/(

```
#include <st

int main() {
        int

        prin
        for(

        }

        retu
}
```

Example Output:

```
[dservos5@cs2211b week9]$ ex4
ASCII TABLE
        32
        33         !
        34         "
        35         #
        36         $
        37         %
        38         &
        39         '
        40         (
        41         )
        42         *
        43         +
        44         ,
        45         -
        46         .
        47         /
        48         0
        49         1
        50         2
        51         3
        52         4
        53         5
```

For Loop: Comma Operator

- The **comma operator** allows us to include multiple expressions in the **initial** and **increment** parts of a for statement.

- **Syntax:**

```
for(initial1, initial2, …; condition; increment1, increment2, …)
   statement
```

- **Simple Example:**

```
for(i = 1, k = 10; i <= 10; i++, k--)
   print("%4d%4d\n", i, k);
```

Prints the following table:

```
   1  10
   2   9
   3   8
   4   7
   5   6
   6   5
   7   4
   8   3
   9   2
  10   1
```

i starts at 1, k starts at 10. i is incremented by 1 each loop, k is decremented by 1 at the end of each loop.

- **Simple Example:**

```
for(i = 1, k = 10; i <= 10; i++, k--)
   print("%4d%4d\n", i, k);
```

For Loop

Example 4b: Update example 4, to print two columns and the ASCII letters from space up to 127 (DEL).

/cs2211/week9/ex4b.c

```c
#include <stdio.h>

int main() {
        int i, k;

        printf("ASCII TABLE\n");
        for(i = ' ', k = 'P'; k <= 127; i++, k++) {
                printf("%4d%4c%10d%4c\n", i, i, k, k);
        }

        return 0;
}
```

For Loop

Example 4b: Update example 4, to print two columns and the ASCII letters from space up to 127 (DEL).

/cs2211/week9/ex4b.c

```
#include <stdio.h>

int main() {
        int i, k;

        printf("ASCII TABLE\n");
        for(i = ' ', k = 'P'; k <= 127; i++, k++) {
                printf("%4d%4c%10d%4c\n", i, i, k, k);
        }

        return 0;
}
```

i starts at ' ' (32), k starts at 'P' (80).

For Loop

Example 4b: U

letters from s

For loop will check that k is less than or equal to 127 each iteration of the loop. If this is no longer true, the loop will exit.

In this case 127 is the DEL character, the last one on the normal (non-extended) ASCII table

/cs2211/week9/e

```
#include <std

int main() {
        int i, k;

        printf("ASCII TABLE\n");
        for(i = ' ', k = 'P'; k <= 127; i++, k++) {
                printf("%4d%4c%10d%4c\n", i, i, k, k);
        }

        return 0;
}
```

For Loop

Example 4b: Update example 4, to print two columns and the ASCII letters from space up to 127 (DEL).

/cs2211/week9/ex4b.c

```
#include <stdio.h>

int main() {
        int i, k;

        printf("ASCII TABLE\n");
        for(i = ' ', k = 'P'; k <= 127; i++, k++) {
                printf("%4d%4c%10d%4c\n", i, i, k, k);
        }

        return 0;
}
```

Both i and k are incremented by 1 at the end of the loop.

For Loop

Example 4b: Update example 4, to print two columns and the ASCII letters from space u

Both i and k are output as their integer and character representation.

The minimum width specifier is used to provide consistent spacing between the numbers/letters.

/cs2211/week9/ex4b.c

```
#include <stdio.h>

int main() {
        int i, k;

        printf("ASCII TABLE\n");
        for(i = ' ', k = 'P'; k <= 127; i++, k++) {
                printf("%4d%4c%10d%4c\n", i, i, k, k);
        }

        return 0;
}
```

For L

Example 4 … e ASCII

letters fro

/cs2211/wee

```
#include

int main(
        i

        p
        f

        }

        r
}
```

Example Output:

```
[dservos5@cs2211b week9]$ ex4b
ASCII TABLE
  32            80   P
  33   !        81   Q
  34   "        82   R
  35   #        83   S
  36   $        84   T
  37   %        85   U
  38   &        86   V
  39   '        87   W
  40   (        88   X
  41   )        89   Y
  42   *        90   Z
  43   +        91   [
  44   ,        92   \
  45   -        93   ]
  46   .        94   ^
  47   /        95   _
  48   0        96   `
  49   1        97   a
  50   2        98   b
  51   3        99   c
  52   4       100   d
  53   5       101   e
  54   6       102   f
  55   7       103   g
  56   8       104   h
```

For Loop

- The **initial**, **condition** and **increment** expression in a for loop are optional.
- **Simple Examples:**

```
int i = 0;
for(; i < 10; i++)
   …
```

```
int i = 0;
for(; i < 10;) {
   i++;
   …
}
```

```
for(i = 0; i < 10; ) {
   i++;
   …
}
```

```
for(; ; )
   …
```

For Loop

- The **initial**, **condition** and **increment** expression in a for loop are optional.
- **Simple Examples:**

```
int i = 0;
for(; i < 10; i++)
   …
```

Equivalent to a while loop.

```
int i = 0;
for(; i < 10;) {
   i++;
   …
}
```

```
for(i = 0; i < 10; ) {
   i++;
   …
}
```

```
for(; ; )
   …
```

For Loop

- The **initial**, **condition** and **increment** expression in a for loop are optional.
- **Simple Examples:**

```
int i = 0;
for(; i < 10; i++)
   …
```

```
int i = 0;
for(; i < 10;) {
   i++;
   …
}
```

```
for(i = 0; i < 10; ) {
   i++;
   …
}
```

Infinite loop, similar to while(1).

```
for(; ; )
   …
```

For vs. While vs. Do While

- In most case, each type of loop can be used interchangeably with a few adjustments.
- Use best loop for the job. Often this is subjective and up to the programmer.
- **break** and **continue** statements can be used with each loop type.
- Each loop supports nesting (loops inside of loops).
- The **for loop** allows null statements (empty **expressions**). For example: `for( ; ; )`. The **while** and **do loop** require an **expression**. For example: `while(i>10)`

Nested For Loop Example

Example 5: Print a timetable for the numbers 1 to 9 (previously we did this with a shell script, now we will recreate it in C).

/cs2211/week9/ex5.c

```c
#include <stdio.h>

int main() {
        int i, j;

        for(i = 1; i < 10; i++) {
                for(j = 1; j < 10; j++) {
                        printf("%4d", i * j);
                }
                putchar('\n');
        }

        return 0;
}
```

Nested For Loop Example

Example 5:

did this wit

/cs2211/week9

```
#include <s

int main()
        int

        for




        }

        return 0;
}
```

Example Output:

```
[dservos5@cs2211b week9]$ ex5
   1   2   3   4   5   6   7   8   9
   2   4   6   8  10  12  14  16  18
   3   6   9  12  15  18  21  24  27
   4   8  12  16  20  24  28  32  36
   5  10  15  20  25  30  35  40  45
   6  12  18  24  30  36  42  48  54
   7  14  21  28  35  42  49  56  63
   8  16  24  32  40  48  56  64  72
   9  18  27  36  45  54  63  72  81
```

Ne

Exa ve

did

/cs2

```
#i

int

}
```

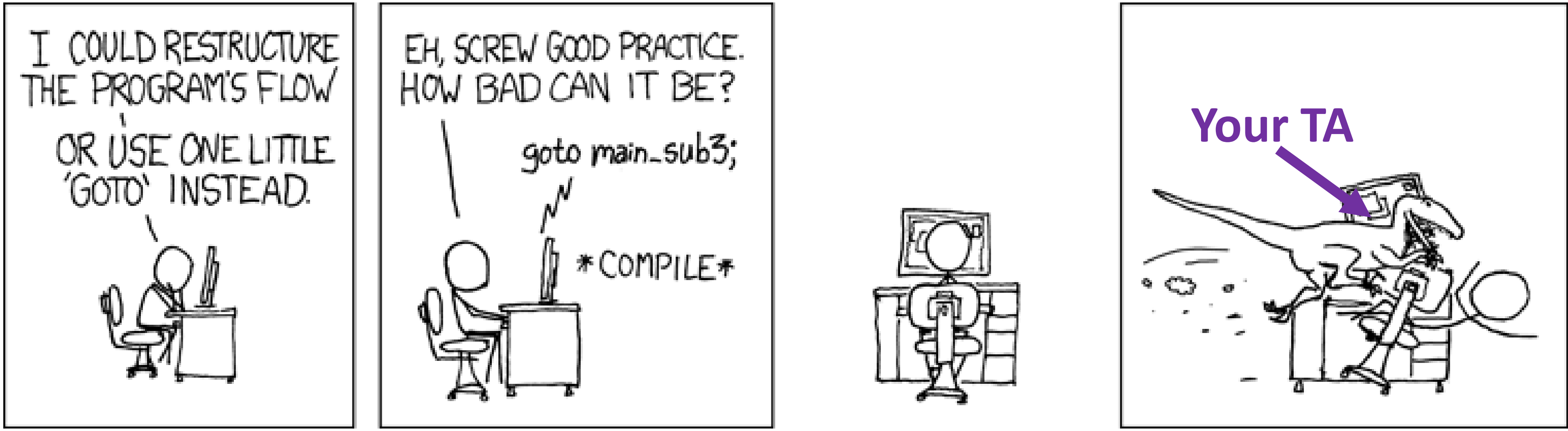
For practice (at home):

Try adding a row and column header so the table looks like this:

```
      1    2    3    4    5    6    7    8    9

     ---------------------------------------------
1  |  1    2    3    4    5    6    7    8    9
2  |  2    4    6    8   10   12   14   16   18
3  |  3    6    9   12   15   18   21   24   27
4  |  4    8   12   16   20   24   28   32   36
5  |  5   10   15   20   25   30   35   40   45
6  |  6   12   18   24   30   36   42   48   54
7  |  7   14   21   28   35   42   49   56   63
8  |  8   16   24   32   40   48   56   64   72
9  |  9   18   27   36   45   54   63   72   81
```

We

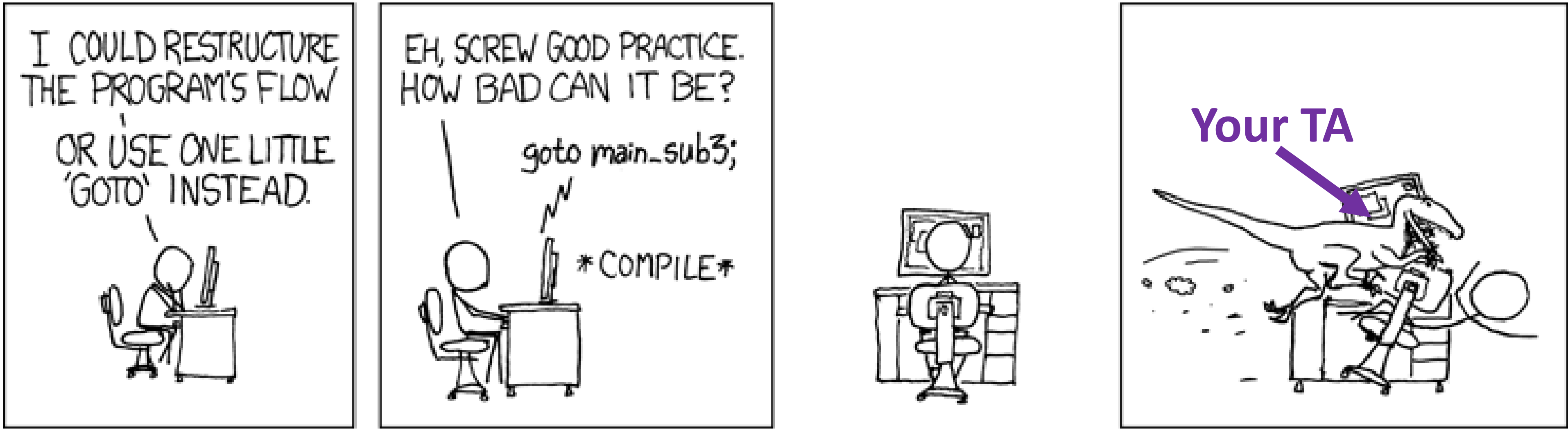
goto


I COULD RESTRUCTURE THE PROGRAM'S FLOW
OR USE ONE LITTLE 'GOTO' INSTEAD.
EH, SCREW GOOD PRACTICE. HOW BAD CAN IT BE?
goto main_sub3;
COMPILE
Your TA

Optional Readings (arguments for and against goto):

- Go To Statement Considered Harmful
- "Go To Statement Considered Harmful" Considered Harmful

goto


I COULD RESTRUCTURE THE PROGRAM'S FLOW
OR USE ONE LITTLE 'GOTO' INSTEAD.
EH, SCREW GOOD PRACTICE. HOW BAD CAN IT BE?
goto main_sub3;
COMPILE
Your TA

You can read about goto in your C textbook:

- Chapter 6.4, pages 113 to 116.
- Also see Q&A about goto in textbook on page 120.

Don't use it in your assignments or exam.