COP 3223H: Introduction to C Programming

Fall 2023

University of Central Florida

Dr. Kevin Moran

Week 6 - Class 1: Loops Part I

Administrivia

- *Small Programming Assignment 2* and *Large Programming Assignment 1* will come out soon!
- All assignments will be returned this week.
- Exams grades will be released by Friday.

Today's Agenda

1. Quick Recap of past concepts
2. Introduction to Repetition and Loops

Quick Review

Function Definitions & Conditions

- Function definitions allow us to define our own instructions • Reliable
  - Reusable
  - Good Practice!
- Conditions allows to execute a set of instructions based on a condition test.
  - Compare and Relational Operators
  - Logical Operators

Introduction to Repetition

- Programmers may need a set instructions to repeat a certain amount of times.
- Programmers could write the same set of instructions multiple times until it is satisfied. (*Bad Practice!*)
  - This can make files look unnecessarily large and complex.
  - It becomes less readable and maintainable.
- In C (and most languages), there is a special syntax that allows programmers to set instructions to repeat.

Different Kinds of Loops

Comparison of Different Loop Types		
Type	**When to Use**	**C Implementation**
Counting Loop	When you know the number of iterations the loop will need.	while, for
Sentinel Controlled Loop	Input a list of data of any length ended by a special value.	while, for
Endfile-controlled Loop	Input any list of data of any length from a data file.	while, for
Input Validation Loop	Repeated interactive input of a data value until this value is within the desired range	do-while
General Conditional Loop	Repeated processing of data until a desired condition is met	while, for

Counting Loops

- Counting loops are loops whose required number of iterations can be determined before loop execution begins.

- In simple English, Programmers know how many times a set of instructions needs to be executed.

- Example:

  - Multiplication

While Loops

- A type of loop that repeats a set of instructions until a condition is met.
- Loop repetition condition is the condition that controls loop repetition.
- Syntax:

```
while(condition)
{
    // instructions go here
}
```

Understanding the While Loop Flow

First condition is evaluated

```
while(condition)
{
    // instructions go here
}
```

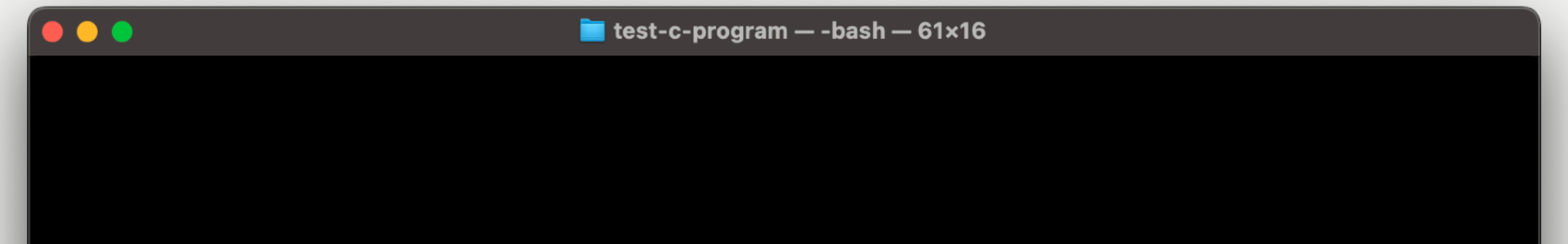
Code inside the control structure is evaluated if the condition was *true*

While Loop Example

Here

```c
int main(void){

int num = 1;

while(num < 3){

    printf("num = %d\n.", num);
    num = num +1;
}

return 0;

}
```

Stack	Space
AA9	
AA8	
AA7	
AA6	
AA5	
AA4	
AA3	
AA2	
AA1	
AA0	

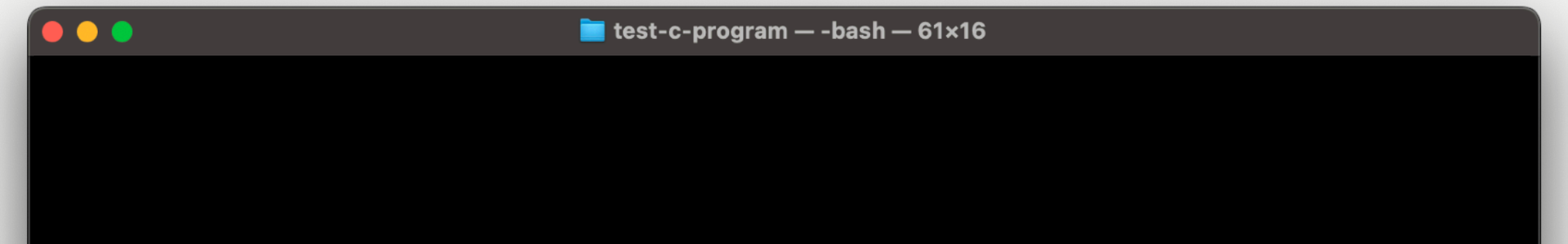


While Loop Example

Here

```c
int main(void){

int num = 1;

while(num < 3){

    printf("num = %d\n.", num);
    num = num +1;
}

return 0;

}
```

Stack	Space
AA9	
AA8	
AA7	
AA6	
AA5	
AA4	
AA3	
AA2	
AA1	
AA0	num1 = 1



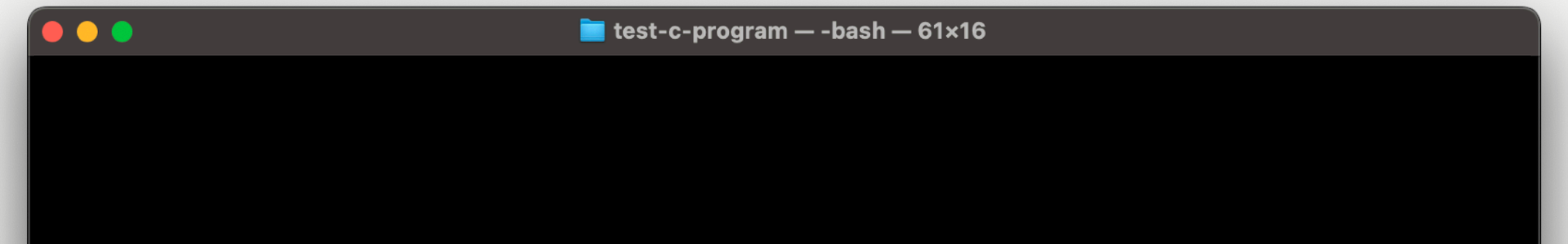
While Loop Example

Here

```c
int main(void){

int num = 1;

while(num < 3){

    printf("num = %d\n.", num);
    num = num +1;
}

return 0;

}
```

True

Stack	Space
AA9	
AA8	
AA7	
AA6	
AA5	
AA4	
AA3	
AA2	
AA1	
AA0	num1 = 1



While Loop Example

```c
int main(void){

int num = 1;

while(num < 3){

    printf("num = %d\n.", num);
    num = num +1;
}

return 0;

}
```

Here

Stack	Space
AA9	
AA8	
AA7	
AA6	
AA5	
AA4	
AA3	
AA2	
AA1	
AA0	num1 = 1

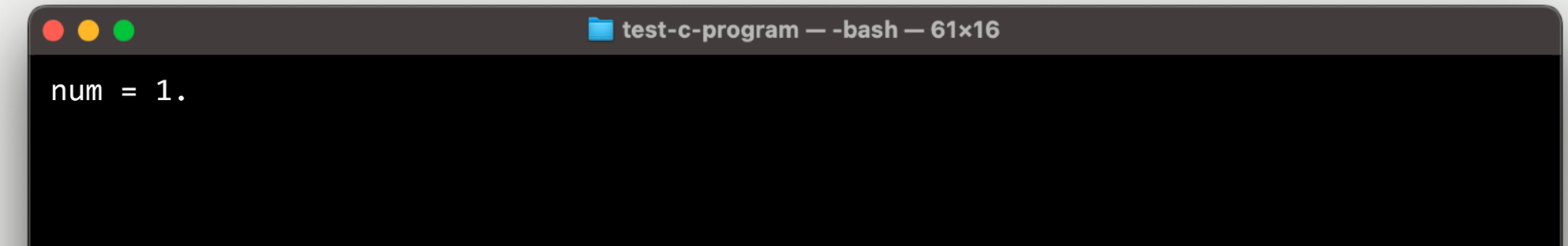
```
test-c-program — -bash — 61×16
num = 1.
```

While Loop Example

```
int main(void){

int num = 1;

while(num < 3){

    printf("num = %d\n.", num);
    num = num +1;
}

return 0;

}
```

Here

Stack	Space
AA9	
AA8	
AA7	
AA6	
AA5	
AA4	
AA3	
AA2	
AA1	
AA0	num1 = 2



While Loop Example

```c
int main(void){

int num = 1;

while(num < 3){

    printf("num = %d\n.", num);
    num = num +1;
}

return 0;

}
```

Here

Now we go back to the condition to see if it is still true.

Stack	Space
AA9	
AA8	
AA7	
AA6	
AA5	
AA4	
AA3	
AA2	
AA1	
AA0	num1 = 2

```
test-c-program — -bash — 61×16
num = 1.
```

While Loop Example

Here

```c
int main(void){

int num = 1;

while(num < 3){

    printf("num = %d\n.", num);
    num = num +1;
}

return 0;

}
```

True

Stack	Space
AA9	
AA8	
AA7	
AA6	
AA5	
AA4	
AA3	
AA2	
AA1	
AA0	num1 = 2

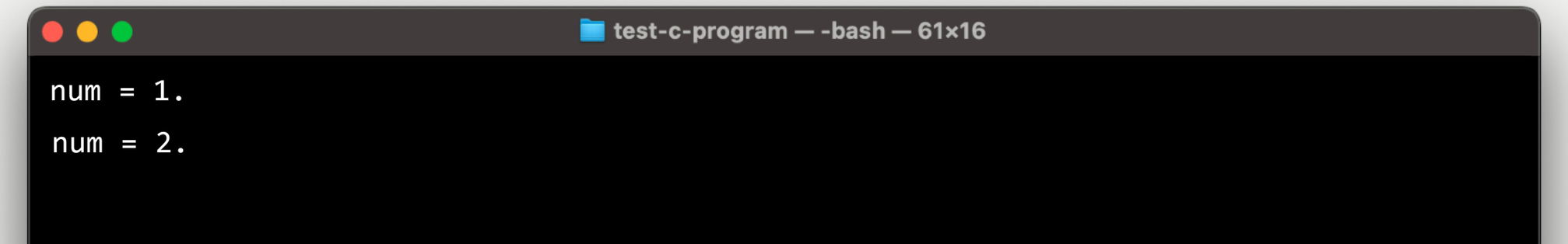
```
test-c-program — -bash — 61×16
num = 1.
```

While Loop Example

```
int main(void){

int num = 1;

while(num < 3){

    printf("num = %d\n.", num);
    num = num +1;
}

return 0;

}
```

Here

Stack	Space
AA9	
AA8	
AA7	
AA6	
AA5	
AA4	
AA3	
AA2	
AA1	
AA0	num1 = 2

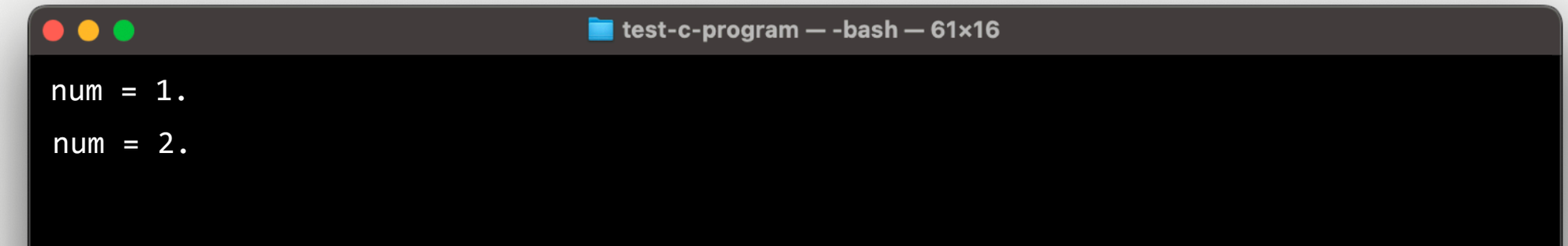


While Loop Example

```c
int main(void){

int num = 1;

while(num < 3){

    printf("num = %d\n.", num);
    num = num +1;
}

return 0;

}
```

Here

Stack	Space
AA9	
AA8	
AA7	
AA6	
AA5	
AA4	
AA3	
AA2	
AA1	
AA0	num1 = 3



While Loop Example

```c
int main(void){

int num = 1;

while(num < 3){

    printf("num = %d\n.", num);
    num = num +1;
}

return 0;

}
```

Here

Now we go back to the condition to see if it is still true.

Stack	Space
AA9	
AA8	
AA7	
AA6	
AA5	
AA4	
AA3	
AA2	
AA1	
AA0	num1 = 3

```
test-c-program — -bash — 61×16
num = 1.
num = 2.
```

While Loop Example

Here

```c
int main(void){

int num = 1;

while(num < 3){

    printf("num = %d\n.", num);
    num = num +1;
}

return 0;

}
```

False!

Stack	Space
AA9	
AA8	
AA7	
AA6	
AA5	
AA4	
AA3	
AA2	
AA1	
AA0	num1 = 3

```
test-c-program — -bash — 61×16
num = 1.
num = 2.
```

While Loop Example

```c
int main(void){

int num = 1;

while(num < 3){

    printf("num = %d\n.", num);
    num = num +1;
}

return 0;

}
```

Here

Stack	Space
AA9	
AA8	
AA7	
AA6	
AA5	
AA4	
AA3	
AA2	
AA1	
AA0	num1 = 3

```
test-c-program — -bash — 61×16
num = 1.
num = 2.
```

Continue Statement

- There is a special keyword in C called continue that can cause an iteration to be skipped.
- *What will the code fragment display?*
- Why does this even exist?
  - In larger programs, there might be special iterations where a certain set values may be invalid to use.

```
int num = 10;

while(num > 0){
    if(num ==5){
        num -= 1;
        continue;
    }

    printf("Continue: num=%d\n.", num);

    num -= 1;

}
```

In-Class Exercise

- What is displayed by the program fragment for an input of *8*?

```
int n;
printf("Please enter a loop number:\n");
scanf("%d", &n);
int ev = 0;

while(ev < n){
    printf("%3d", ev);
    ev +=2;
}

printf("\n");
```

http://bit.ly/3t5fcDp

In-Class Exercise

- You may have noticed instructions where variable have assignment statement that involves itself.
  - `var1 = var1 + 1;`
  - `var2 = var2 - 2;`
- C, this can be rewritten as a compound statement.
  - `+: +=  e.g., var1 += 1;`
  - `-: -=  e.g., var2 -= 2;`
  - `*: *=`
  - `/: /=`
  - `%: %=`

Examples of Compound Assignment Operators

Compound Assignment Operators	
`count_emp = count_emp + 1;`	`count_emp += 1;`
`time = time - 1;`	`time -= 1;`
`total_time = total_time + times;`	`total_time += times;`
`product = product * item;`	`product *= item;`
`n = n * (x + 1);`	`n *= (x + 1);`

In-Class Exercise

- Write the equivalents for the following statements using compound assignment operators:

```
s = s / 5;

q = q * (n + 4);

z = z - x * y;

t = t + (u % v);
```

http://bit.ly/3LCpwcp

Acknowledgements

Slides adapted from Dr. Andrew Steinberg's COP 3223H course