Tim Tanasse
CSCD 240
April 30, 2015
Notes # #

Problem 1: Write, compile, and execute the following C file and answer the following questions

- **What is the size of ptr on cslinux?**
  ptr takes up 8 bytes
- **What is the size of arr[0] on cslinux?**
  arr[0] takes up 4 bytes
- **Draw a diagram of the memeory map of ptr and arr given that arr is at base address 0x5600bc**
  See last page

Problem 2: Take educated guesses about the functioning of some code involving pointers

-
  ```
      ptr++;
  printf("*ptr %d\n", *ptr);
  printf("ptr %p\n", ptr);
  ```

  Explanation: The pointer is incremented, which moves it by the size of an integer (4) from 0x5600bc to 0x5600c0. Prints the value at the ptr location: 400, or **arr[1]**. Prints the memory address contents of **ptr**, which should have shifted 4 bytes from the previous address.

-
  ```
      *++ptr;
  printf("*++ptr %d\n", *ptr);
  printf("ptr %p\n", ptr);
  ```

  Explanation: The unary operator * is read from right to left, which makes ***++ptr** execute similarly to ***(++ptr)**. The pointer is incremented from 0x5600c0 to 0x5600c4. Prints the value at the new memeory location: 600, or **arr[2]**. Prints the memory address contents of **ptr**, which should have shifted 4 bytes from the previous address.

-
  ```
      *ptr++;
  printf("*ptr++ %d\n", *ptr);
  printf("ptr %p\n", ptr);
  ```

  Explanation: In the same way as the previous explanation, ***ptr++** is interpreted as ***(ptr++)** and the pointer **ptr** is incremented. Prints the value at the new memory location: 800, or **arr[3]**. Prints the memory address contents of **ptr**, which should have shifted 4 bytes from the previous address 0x5600c4 to 0x5600c8.

-
  ```
      ptr = arr; // reset ptr, no need to explain this statement

      // fun with printf repeat last couple of commands
      printf("*++ptr %d\n", *++ptr);
      printf("ptr %p\n", ptr);
  ```

Explanation: Due to the functionality of ++ and the previously stated functionality of *, the pointer is incremented before being used in **printf**. Prints the value that the pointer now points to: 400, or **arr**[**1**]. Prints the memeory address contents of ptr: 0x5600c0.

-
  ```
      printf("*ptr++ %d\n", *ptr++);
      printf("ptr %p\n", ptr);
  ```

Explanation: Unlike the above, post incrementing results in the pointer being incremented after it is used in **printf**. Prints the dereferenced pointer value: 400 or **arr**[**1**], then increments the pointer. Prints the new memory address contents of **ptr**: 0x5600c4.

-
  ```
      ptr = arr; // reset ptr, no need to explain this statement
      *ptr += 1;
      printf("*ptr %d\n", *ptr);
      printf("ptr %p\n", ptr);
  ```

Explanation: The pointer is dereferenced, and then the value is incremented by 1 from 200 to 201. Prints the derefenced value: 201, or **arr**[**0**]. Prints the memory address contents of **ptr**: 0x5600bc.

-
  ```
  printf("*(ptr+1) = %d\n", *(ptr+1));
  ```

Explanation Takes the pointer value and increases it by 4 bytes. Prints the value at that adjusted location: 400.

-
  ```
    ptr = arr; // reset ptr, no need to explain this statement
    *(arr+2) = *ptr+100;
    printf("*(arr+2) = %d\n", *(arr+2));
  ```

Explanation: The **arr** variable is a memory address of the variable **arr**[**0**]. So the value at arr + 2 or **arr**[**2**] is getting the derefernced value of **ptr** + 100. Prints the new derefenced value: 301, or arr[2]

-
  ```
  ptr = arr + 5;
  printf("*ptr %d\n", *ptr);
  printf("ptr %p\n", ptr);
  ```

Explanation: The pointer is set to arr + 5 which is equivalent to getting the memory address of **arr**[**5**]. Prints the dereferenced value: 1200. Prints the memeory address content of **ptr**: 0x5600D0

-
  ```

  ptr = arr; // reset ptr, no need to explain this statement

  arr[2] = *(ptr + 5);
  printf("arr[2] = %d\n", arr[2]);
  ```

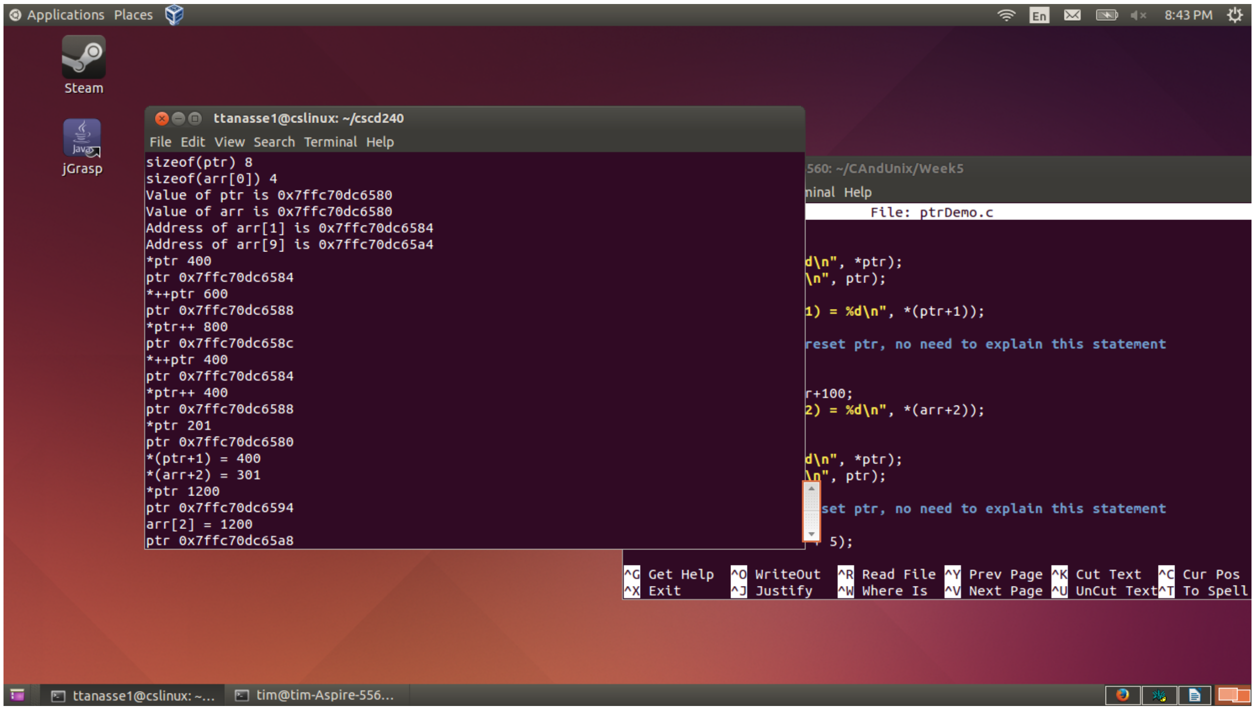
Explanation: **arr**[**2**] gets the dereferenced value of **ptr + 5**, which is **arr**[**5**]. Prints the value of **arr**[**2**]: 1200.

-
  ```
  ptr = (arr + 10);
  printf("ptr %p\n", ptr);
  printf("*ptr %d\n", *ptr);
  ```

Explanation: **ptr** gets ten 4 byte increments, which puts it outside of the array **arr**. Prints the memory address content of **ptr**: 0x5600e4. Prints the dereferenced value at that memeory location.

Problem 3: Corrections

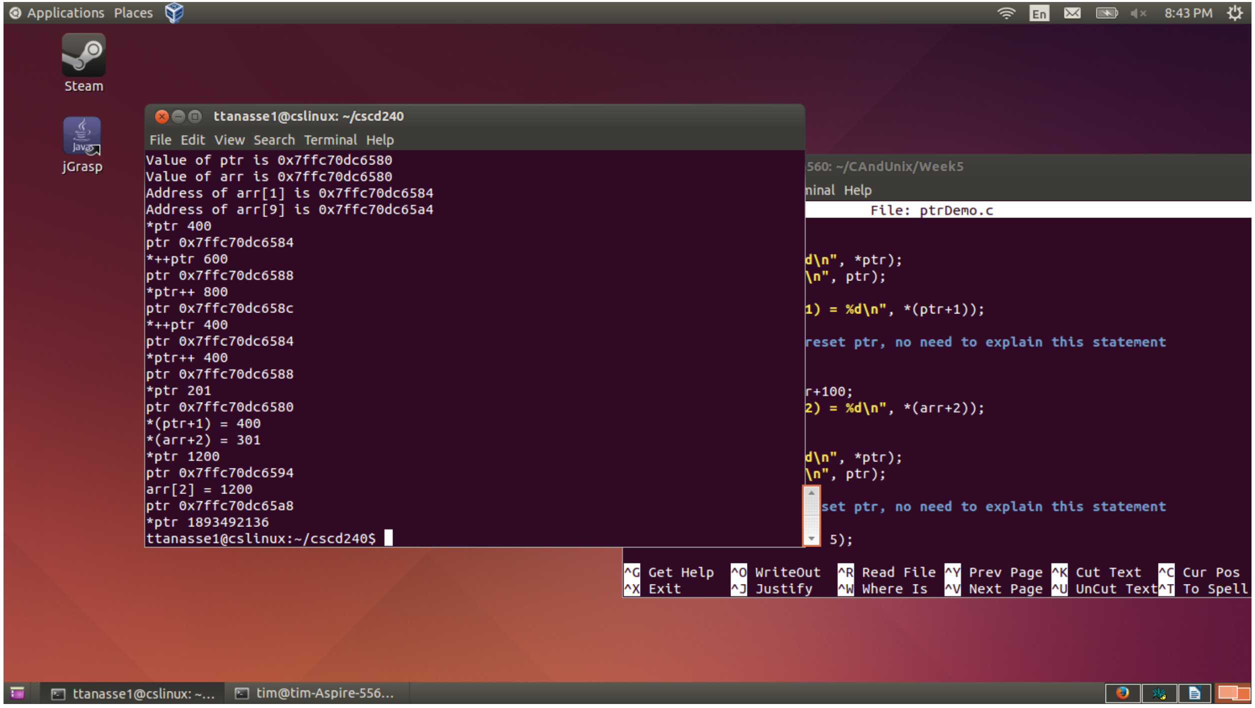
- **19:** Correct Guess
- **23:** Correct Guess
- **27:** Correct Guess
- **34:** Correct Guess
- **37:** Correct Guess
- **42:** Correct Guess
- **46:** Correct Guess
- **50:** Correct Guess
- **53:** Correct Guess
- **59:** Correct Guess
- **62:** Correct Guess



```
sizeof(ptr) 8
sizeof(arr[0]) 4
Value of ptr is 0x7ffc70dc6580
Value of arr is 0x7ffc70dc6580
Address of arr[1] is 0x7ffc70dc6584
Address of arr[9] is 0x7ffc70dc65a4
*ptr 400
ptr 0x7ffc70dc6584
*++ptr 600
ptr 0x7ffc70dc6588
*ptr++ 800
ptr 0x7ffc70dc658c
*++ptr 400
ptr 0x7ffc70dc6584
*ptr++ 400
ptr 0x7ffc70dc6588
*ptr 201
ptr 0x7ffc70dc6580
*(ptr+1) = 400
*(arr+2) = 301
*ptr 1200
ptr 0x7ffc70dc6594
arr[2] = 1200
ptr 0x7ffc70dc65a8
```

```
          File: ptrDemo.c

d\n", *ptr);
\n", ptr);

1) = %d\n", *(ptr+1));

reset ptr, no need to explain this statement


r+100;
2) = %d\n", *(arr+2));


d\n", *ptr);
\n", ptr);

 set ptr, no need to explain this statement

  5);
```

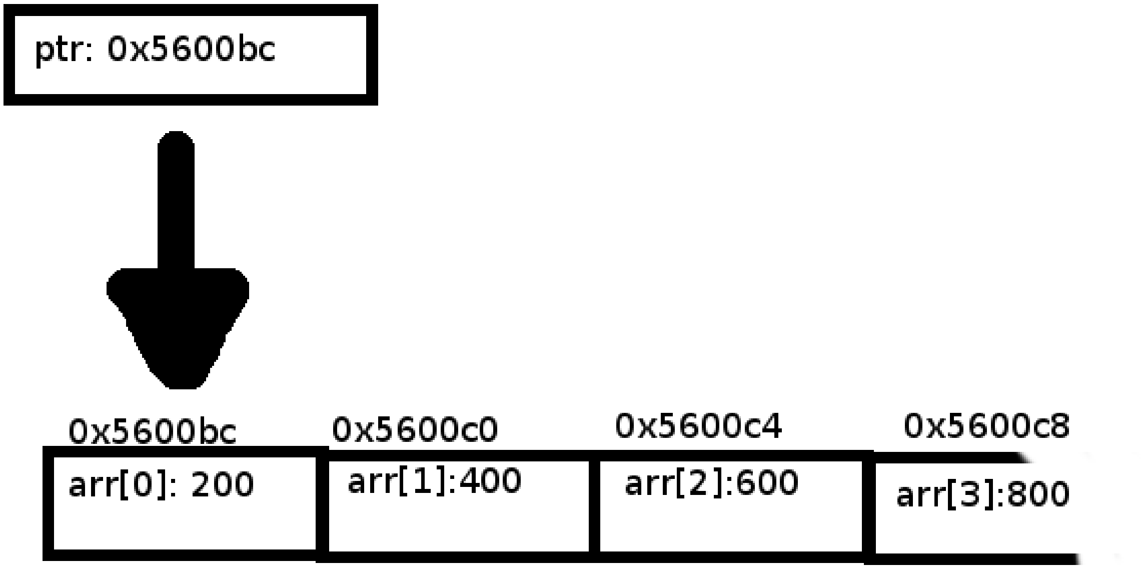
```
^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Text^T To Spell
```



```
Value of ptr is 0x7ffc70dc6580
Value of arr is 0x7ffc70dc6580
Address of arr[1] is 0x7ffc70dc6584
Address of arr[9] is 0x7ffc70dc65a4
*ptr 400
ptr 0x7ffc70dc6584
*++ptr 600
ptr 0x7ffc70dc6588
*ptr++ 800
ptr 0x7ffc70dc658c
*++ptr 400
ptr 0x7ffc70dc6584
*ptr++ 400
ptr 0x7ffc70dc6588
*ptr 201
ptr 0x7ffc70dc6580
*(ptr+1) = 400
*(arr+2) = 301
*ptr 1200
ptr 0x7ffc70dc6594
arr[2] = 1200
ptr 0x7ffc70dc65a8
*ptr 1893492136
ttanasse1@cslinux:~/cscd240$ 
```

```
          File: ptrDemo.c

d\n", *ptr);
\n", ptr);

1) = %d\n", *(ptr+1));

reset ptr, no need to explain this statement


r+100;
2) = %d\n", *(arr+2));


d\n", *ptr);
\n", ptr);

 set ptr, no need to explain this statement

  5);
```

```
^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Text^T To Spell
```


ptr: 0x5600bc
0x5600bc
0x5600c0
0x5600c4
0x5600c8
arr[0]: 200
arr[1]:400
arr[2]:600
arr[3]:800