In-class Exercise 11
Assigned: April 19th, 2016
Due: April 21st, 2016, by midnight

All parts to be turned in

Please turn in each part as a separate zip/rar file. Example: ex11-p1.zip, ex11-p2.zip, etc.

7zip is a nice, free compression tool if you do not have one available.

Table of Contents

Memory Addresses

Any time we create a variable, that variable requires some **memory** from our computer. Different types of variables require different amounts of memory – Booleans and characters need 1 byte (8 bits), integers and floats need 4 bytes, and doubles need 8 bytes.

The following 8 bits (1 byte) makes up the number 65, which in ASCII represents the character 'A'.
So, 'A' in binary is 01000001.

Bit Position	128	64	32	16	8	4	2	1
Value	0	1	0	0	0	0	0	1

This is just the value, however. When we tell the program that we need 1 byte in order to store a **character**, we reserve a **memory address** to store any character in this space.

C++	**Translation**
`char choice;`	I need 1 byte to store some character. Allocate 1 byte at memory address **0x7fff95**.
`choice = 'Y';`	Assign the value of **01011001** (Y) to address **0x7fff95.**

Every time we run a program, even if the code is the same, each variable will get a different memory address – it takes any space that is available, and memory is shared across all applications on a computer. While your program is using memory address **0x7fff95** , no other programs *should be able to* access that memory – They can, if the code is written poorly or if it is a malicious program, but in general the memory allocated for your program is just used for your program until the program ends.

In C++, the "Address-of" operator is the ampersand: &. If you put this operator before a variable name, you will get its memory address. (Note that this is not the reference operator – same symbol, different context.)

C++	**Output**
`int myNumber;` `cout << &myNumber << endl;`	`0x69feec`

Arrays are also variables and have memory addresses, but you do not need the address-of operator to get its address:

C++	**Output**
`int myNumbers[20];` `cout << myNumbers << endl;`	`0x69fe9c`

Exercise 11 Part 1

Write a small program where you will create a series of variables, assign them values, then output the address of that variable and its value. Use the address-of operator &.

Variables to declare:

- An Integer
- A Float
- A String

Sample Output:

```
0x69fee8 = 20
0x69fee4 = 9.99
0x69fee0 = Bob
```

Exercise 11 Part 2

Write a small program that will declare an array, initialize values for each element of the array, and then displays the **address of the array** and the **address of each element of the array**.

Remember that, to output the address of an array, you do not use the address-of operator, you just cout the array's name.

For an **element** of the array, it is treated like a variable – so if you want to output the address of element #0, you would have to add the & before the array and subscript.

Notice that the address of the array is the same address as the first element. This is not a coincidence!

Sample Output:

```
Array address: 0x69fec8
Item 0 address: 0x69fec8 value: CL
Item 1 address: 0x69fecc value: Bom
Item 2 address: 0x69fed0 value: Dara
Item 3 address: 0x69fed4 value: Minzy
```

Pointers

Pointers are a type of variable that stores a memory address. So, instead of storing a value like “Hello, World!” or 4.99, it would store the address to another variable.

A pointer’s data-type needs to match the kind of data that it will point to, but with a * at the end.

So remember, if we wanted to output the address of a variable, we would use:

```
cout << &myName << endl;
```

And we use the same if we want to assign its address to a pointer.

```
int myNumber = 10;
string myName = "Bob";
float myPrice = 9.99;

int* ptrInteger = &myNumber;
string* ptrString = &myName;
float* ptrFloat = &myPrice;
```

If we cout the pointer, it will give us the same value as if we had output the variable with the address-of operator. So, both of these statements would output the same thing:

```
cout << &myName << endl;

cout << ptrString << endl;
```

Exercise 11 Part 3

Update Exercise 11 Part 1. Continue creating the int, string, and float variables, but also make corresponding pointer variables that will point to each of their addresses. Then, cout the pointers, rather than the addresses of the variables.

Submit this as ex11-p3.zip.

Sample Output:

```
0x69fee8 = 20
0x69fee4 = 9.99
0x69fee0 = Bob
```

Exercise 11 Part 4

Write a program that will ask the user whether they want to see information about **integers**, **floats**, **booleans**, or **doubles**.

Whichever data-type the user selects:

1. Create a variable of that type
2. Create a pointer and assign it the address of that variable
3. Output the size of the variable using the sizeof(…) function.
4. Output the address of the variable by outputting the pointer (no address-of operator).

You can pass any variable name into the sizeof function to get its size. For example:

C++	Output
`char choice;` `cout << sizeof( choice ) << endl;`	`1`

Sample Output:

```
[i]nt, [f]loat, [b]oolean, or [d]ouble? i
Integer size: 4, address: 0x69fee8
```

Dereferencing Pointers

So, we have a pointer that is pointing to a memory address. What do we do with this?

Using the de-reference operator * , we can get the **value** that is stored in the memory address.

Note that this * is different from when we're declaring a pointer variable:

Creating a pointer	**Outputting the value at an address**
`int* ptrInteger;`	`cout << *ptrInteger << endl;`

So let's look at this step-by-step:

`string studentA = "Ashley";`
`string studentB = "Kaidan";`
`string studentC = "Liara";`
	Creating normal variables

`string* ptrStudent;`	Creating a string pointer
`ptrStudent = &studentA;`	Pointing *ptrStudent* to one memory address
`cout << &studentA << endl;` **`cout << ptrStudent << endl;`**	Outputting the address of a student
`cout << studentA << endl;` **`cout << *ptrStudent << endl;`**	Outputting the value of studentA, directly and through a pointer.
`ptrStudent = &studentB;`	Changing who *ptrStudent* is pointing to.

Exercise 11 Part 5

Write a program that has 3 float variables, named **priceHamburger, priceFries, priceSalad**. Assign some prices to each of these variables.

Create a variable that is a **float pointer**, called **ptrPrice**. Do not assign it to anything initially.

Ask the user whether they want a hamburger, fries, or a salad. Based on what their answer is, assign **ptrPrice** the address of the corresponding price variable (priceHamburger, priceFries, or priceSalad).

Create another float called **taxAmt**. Figure out the tax by multiplying the value of **ptrPrice** by 0.065. Then, increase the price by the tax amount by adding the **taxAmt** to the value of **ptrPrice**.

Finally, output the price-plus-tax, by outputting the value of **ptrPrice**.

Sample Output:

```
[h]amburger, [f]ries, or [s]alad? h
Original price: $3.99
With tax price: $4.24935
```

Exercise 11 Part 6

Write a program that has an array of strings, called **students**. The array should be of size 3. Assign some values to these array elements.

Then, create a variable that is a **string pointer**, called **ptrStudent**.

Ask the user to enter an index: 0, 1, or 2. Assign **ptrStudent** the address of the student at this index (e.g., student[1]'s address).

Then, ask the user to enter a new name for the student. Use cin and store the result to the appropriate element through the ptrStudent pointer – use the de-reference operator to assign a value to this item.

Finally, display the name of all students with a for-loop:

```
for ( int i = 0; i < 3; i++ )
{
    cout << "student " << i << " = " << students[i] << endl;
}
```

Sample Output:

```
Enter 0, 1, or 2: 2
New name: Wrex

student 0 = Ashley
student 1 = Kaidan
student 2 = Wrex
```