Dates

Assigned: April 26th, 2016

Due: April 28tht, 2016, by end-of-day

Instructions

This is a solo project to teach the basics of dynamic variables and arrays.

Upload to D2L by due date. Keep it organized – name each source file as "exercise11_part1.cpp".

Table of Contents

Dynamic Variables

We can create new variables and arrays dynamically. With arrays, we no longer need to know the size of the array at compile-time, but we can decide that at run-time. For example, we could ask the user for the amount of items, and then create an array of that size.

To create dynamic variables and arrays, we need to use pointers. Remember that a pointer points to a memory address. By using the **new** keyword, we can allocate memory through a pointer – it will point to a new address, and have that memory address prepared for whatever we've created.

But, unlike our normal variables, dynamic variables do not free their memory automatically. This means, we have to manually free that memory with the **delete** keyword. So, a good practice to start with is, any time we write a **new** keyword, we also need a **delete** somewhere.

If you don't free your allocated memory, you will get a memory leak. If your program allocates a lot of memory and never frees it, it will eat up RAM and cause system problems – you can't get that memory back until the user resets their computer.

Dynamic Variable Reference

Here are the steps to creating a new dynamic variable:

`string* dynamicString;`	Create a pointer variable.
`dynamicString = new string;`	Create a new string with the pointer.
`*dynamicString = "Hello!";`	Assign a value to our new variable. (must go through the pointer – use the de-reference operator)
`cout << *dynamicString << endl;`	Display the value of the variable.
`delete dynamicString;`	Free up the memory when done

Exercise 13 Part 1

1. In your program, create three pointers:

- myInt, an integer pointer
- myStr, a string pointer
- myFloat, a float pointer

2. Afterward, you will allocate memory for each of these variables with the **new** keyword. Note that the data-type that goes after **new** must match the pointer type. (See the reference for sample code).

3. Then, assign values to each of the variables. Make sure to use the de-reference operator!

- myInt = 20
- myStr = your name
- myFloat = 199.99

4. Display the value of each of the variables. You will also need a de-reference operator here.

5. Finally, free the memory for each pointer using the **delete** keyword.

Sample Output

```
20      yourname    199.99
```

Exercise 13 Part 2

Add the following struct to the top of your source file (it can go in the same file as main()):

```
struct Person
{
    string name;
    Person* ptrFriend;
};
```

The Person class has a name, and a pointer to their friend. Their friend must also be a Person type.

Within main, do the following:

1. Create three Person pointers: personA, personB, and personC. Allocate memory for each of them – you will need to create **new Person** for each.

2. Set up names for personA, personB, and personC. You can set the value of a class member in one of two ways, when using pointers:

- `(*personA).name = "Bob"; // using the de-reference operator`
- `personA->name = "Bob";   // using the member-of operator`

Set the values to:

PersonA	PersonB	PersonC
Joe	**Austin**	**Elly**

3. Set each person's **ptrFriend** pointer to point to the address of another Person. You won't need the address-of operator, since personA, B, and C are ***already*** pointers, so you can set one pointer to another pointer!

- `personA->ptrFriend = personB;`

Person	PersonA, Joe	PersonB, Austin	PersonC, Elly
Friend	PersonB	PersonC	PersonA

4. Display each person's name, and then the name of their friend. You can access members through the friend pointer:

- `personA->ptrFriend->name`

Sample Output:

```
Person A: Joe, friend: Austin

Person B: Austin, friend: Elly

Person C: Elly, friend: Joe
```

Dynamic Arrays

The **new** and **delete** keywords can be used to make arrays, too. With dynamic arrays, you don't need to estimate the size of an array – which wastes memory, and could have problems if you run out of array space!

You also create dynamic arrays through pointers. Remember that you will still need to **delete** whatever you create with **new**!

Dynamic Array Reference

`float* prices;`	Create a pointer
`prices = new float[ 5 ];`	Allocate memory as an array

`for ( int i = 0; i < 5; i++ )`
`{`
    `prices[i] = i * 2;`
    `cout << prices[i] << endl;`
`}`
	Assign values and display the values.

`delete [] prices;`	Free the memory up afterward

Note that with dynamic arrays, you don't need to use the de-reference operator when accessing elements of the array – just treat it like a normal array!

You can also set a dynamic array's size based on a variable value:

```
int size;
cout << "Size: ";
cin >> size;

float* prices;
prices = new float[ size ];

// ...

delete [] prices;
```

So they are very useful for all sorts of applications!

Exercise 13 Part 3

You will use a random number generator for this program, so you will need the following libraries:

```
#include <cstdlib>        // to use rand()
#include <ctime>          // to use time()
```

1. Seed the random number generator with:

```
srand( time( NULL ) );
```

2. Ask the user to enter the amount of lotto balls there will be. Store the value in a **size** variable.

3. Create a dynamic array called **lottoNumbers**. It will be the size that the user specified.

4. For each element of the array, from 0 to **size**, set the element's value to a random number between 0 and 100.

5. Display the value of each lotto ball.

6. Delete the dynamic array before the program ends.

Sample Output

```
Lotto number count: 5
57      11      74      91      35
```

Exercise 13 Part 4

Create the following structs at the top of the source file:

```
struct Employee
{
    string name;
};

struct Manager
{
    string name;
    Employee* employees;
    int employeeCount;
};
```

1. Create an array of strings, that contains a list of names. This can be a normal static array with a set size. Example:

```
string names[ 16 ] = {
    "Koios", "Julianna", "Agata", "Arundhati",
    "Zemfira", "Fedya", "Kim", "Ashok",
    "Jouni", "Blandina", "Ampelio", "Rosmunda",
    "Breeshey", "Ferdinand", "Gertrude", "Sarit"
    };
```

2. Ask the user how many managers there should be. Store this in an integer called **managerCount**.

3. Create a Manager pointer called **managers**, and dynamically allocate an array of the size of **managerCount**.

4. Create a for-loop that will go from **0** until the **managerCount**. For each manager:

A. Assign the manager a random name from the names list:

```
managers[i].name = names[ rand() % 16 ];
```

B. Randomly set this manager's **employeeCount** to a value between 0 and 3.

C. Use the **employees** pointer in the Manager class to create a dynamic array of employees, if **employeeCount** is above 0.

D. Create a second for-loop that will set up each of the employees. Iterate from **0** to **managers[i].employeeCount**:

a. Assign the employee a random name from the names list:

```
managers[i].employees[j].name = names[ rand() % 16 ];
```

5. After all managers and employees are set up, display a list of all managers and their employees to the screen with another set of for-loops.

```
for ( int m = 0; m < managerCount; m++ )
{
    cout << endl;
    cout << "Manager:           "
        << managers[m].name << endl;
    cout << "Underling Count:   "
        << managers[m].employeeCount << endl;

    for ( int e = 0; e < managers[m].employeeCount; e++ )
    {
        cout << "\t" << e+1 << ". "
            << managers[m].employees[e].name << endl;
    }
}
```

Sample Output:

```
How many managers? 5

Manager:            Ashok
Underling Count:    1
        1. Blandina

Manager:            Arundhati
Underling Count:    2
        1. Sarit
        2. Ampelio

Manager:            Breeshey
Underling Count:    0

Manager:            Ferdinand
Underling Count:    2
        1. Rosmunda
        2. Agata

Manager:            Rosmunda
Underling Count:    2
        1. Kim
        2. Breeshey
```