Pointer Cheat Sheet

Address-of Operator &
When adding this to the beginning of a variable name, you will be accessing the **memory address**. Pointers **store** memory addresses.

De-reference Operator *
When adding this to the beginning of a pointer variable name, you will be accessing the **value** that the pointer is pointing to.

Using a pointer

1. Declare the pointer.

```
string* ptrText;
```

2. Assign the pointer to the address of another variable.

```
ptrText = &myVar;
```

3. To access the value of the pointed-to variable (*myVar)*, use the de-reference operator.

```
*ptrText = "hello!";
cout << *ptrText;
```

4. If you try to cout the pointer name, it will give you the **address** it is pointing to.

```
cout << ptrText;
```

Output: `0x7ffdff5afbb0`

Pointer Cheat Sheet

Address-of Operator &
When adding this to the beginning of a variable name, you will be accessing the **memory address**. Pointers **store** memory addresses.

De-reference Operator *
When adding this to the beginning of a pointer variable name, you will be accessing the **value** that the pointer is pointing to.

Using a pointer

1. Declare the pointer.

```
string* ptrText;
```

2. Assign the pointer to the address of another variable.

```
ptrText = &myVar;
```

3. To access the value of the pointed-to variable (*myVar)*, use the de-reference operator.

```
*ptrText = "hello!";
cout << *ptrText;
```

4. If you try to cout the pointer name, it will give you the **address** it is pointing to.

```
cout << ptrText;
```

Output: `0x7ffdff5afbb0`

Dynamic Array Cheat Sheet

1. First you declare a pointer.

```
string* studentList;
```

2. Then you allocate memory for an array through the pointer. You can hard-code a size or use a variable for the size.

```
studentList = new string[ size ];
```

3. Use the array like normal

```
studentList[0] = "Ada";
```

4. Deallocate the memory when done with it.

```
delete [] studentList;
```

"Resizing" a dynamic array

1. Create a bigger array

```
string* newArr = string[ bigsize ];
```

2. Copy the data over

```
for ( i = 0; i < count; i++ )
{
    newArr[i] = origArr[i];
}
```

3. Free the old array's memory.

```
delete [] origArr;
```

4. Update the pointer.

```
origArr = newArr;
```


Dynamic Array Cheat Sheet

1. First you declare a pointer.

```
string* studentList;
```

2. Then you allocate memory for an array through the pointer. You can hard-code a size or use a variable for the size.

```
studentList = new string[ size ];
```

3. Use the array like normal

```
studentList[0] = "Ada";
```

4. Deallocate the memory when done with it.

```
delete [] studentList;
```

"Resizing" a dynamic array

1. Create a bigger array

```
string* newArr = string[ bigsize ];
```

2. Copy the data over

```
for ( i = 0; i < count; i++ )
{
    newArr[i] = origArr[i];
}
```

3. Free the old array's memory.

```
delete [] origArr;
```

4. Update the pointer.

```
origArr = newArr;
```