Pointers and Dynamic Arrays

A pointer is a variable which stores the address of another variable, using the dereference operator (*)

The declaration of pointers has the following syntax:

type variable_name*

For a better understanding, read it right to left.

For example:

int number translates to:*

"The variable called number points to the data type int"

The ampersand sign (&) is known as the address-of-operator

For example:

A = &B translates to:

"The variable called A is equal to the address of B"

Examples:

```
int num1 = 5;
int* ptr = &num1;
cout << ptr;
```

Output:

(Random address) Not our value!

This is because the pointer variable stores the address of another variable. If we want the value, we need to use the dereference operator:

```
int num1 = 5;
int* ptr = &num1;
cout << *ptr;
```

Output:
5

*Note that the (*) asterisk used when declaring a point only means that is **IS** a pointer. It is different than the dereference operator.* ***(See next page)***

For example:

*A = *B translates to:*

"The variable A is equal to the value pointed to by B"

Dynamic Arrays

Arrays work very much like pointers, and an array can always be converted to the pointer of the proper type.

For Example:

```
int arr[10];
int* ptr;
ptr = arr;  - This is a valid statement!
```

The declaration of a dynamic array has the following syntax:

type my_variable_name – First implement a pointer*

my_variable_name = new type[size] – dynamically allocates a block of memory

In this setup, you can first initialize the pointer, and then allocate the array with the size later, or when you need it.

The difference between the declaration of a normal array and a dynamic array is allocation performed by **"new",** which allows memory to be assigned using any variable for size rather than a constant integer like the case of a regular array.

You can declare the pointer, then allocate the array, or you can do it all on the same line.

(See next page)

For example:

Two ways:

1.

   int ptr = new int[10];*

2.

   int ptr;*

   ptr = new int[10];

translates to:

> *"A variable called pointer that points to an integer is equal to a pointer that points to the first element of a sequence of integers allocated with a size of 10."*

On the left side of the "=" sign, you have a pointer that points to a type, int. On the right side, the system dynamically allocates space for 10 elements of type int, and a returns a pointer to the first element of the sequence.

If I want to print a dynamic array, it works the same as a regular array.

For example:

Say my dynamic array has values: 1 2 3 with a size of 3

```
for(int i = 0; i < size; i++)
cout << dyn_array[i] <<  " ";
```

Output:

1 2 3

A dynamic array is very similar to a regular array!!! The main difference is that a dynamic array's size is modifiable at runtime, and the declaration is different.

Key points for dynamic arrays:

- You must create a pointer of your desired type
- Dynamic arrays are declared using new keyword
- We use square brackets to specify the number of elements to be stored
- Use the delete operator with [] to free up memory for the whole array