POINTERS:

```
int number = 1;
int *ptrNumber = &number;
cout << ptrNumber;                                  // address of number
cout << *ptrNumber;                                 // value stored there.. 1
```

```
int val = 42;
int *pVal = &val;                                   // assign pVal the address of val
int *pVal2 = pVal;                                  // assign pVal2 the address of val

cout << pVal;                                       // address of val
cout << pVal2;                                      // address of val

cout << *pVal2;                                     // 42
cout << *pVal;                                      // 42

cout << &pVal2;                                     // address of val2
cout << &pVal;                                      // address of val
```

```
int num = 42;                                       // assign 42 to num
int *pnum = &num;                                   // assign address of num to pointer pnum
cout << pnum << endl;                               // address of num;
cout << *pnum << endl;                              // 42

*pnum = 65;
cout << *pnum << endl;                              // 65
cout << num << endl;                                // 65
```

```
const int size = 4;
int numbers[size] = {5, 10, 15, 20};
for(int i : numbers)
cout << i << " ";
int *pnumbers = &numbers[0];
cout << endl << *pnumbers << " " << *(pnumbers + 1);    // 5 10
```

```
const int size = 4;
int numbers[size] = {5, 10, 15, 20};
for(int i = 0; i < size; i++)
       cout << *(numbers + i);                      // 5 10 15 20
```

```
void pswap(int *n1, int *n2)
{                                                   // take de-referenced n1, n2
       int temp = *n1;
       *n1 = *n2;
       *n2 = temp;
}
                                                    // Only use if you might send empties
int main()
{
       int num1 = 100;
       int num2 = 200;
       cout << "Num1: " << num1 << endl;            // 100
       cout << "Num2: " << num2 << endl;            // 200

       pswap(&num1, &num2);                         // send address reference to pswap

       cout << "Num1: " << num1 << endl;            // 200
       cout << "Num2: " << num2 << endl;            // 100

       return 0;
}
```

CLASS STRUCTURE:

```
class Name
{
	private:                                        // Private variables and Methods
	string first, last;

	public:                                         // Public variables and Methods
	Name(string fname, string lname)                // Constructor
	{
		first = fname;
		last = lname;
	}
	Name()                                          // default Constructor
	{
		first = "empty";
		last = "empty";
	}
	const string getFirst()                         // Getter -returns const
	{
		return first;
	}
	void setFirst(string f)                         // Setter
	{
		first = f;
	}
	string toString()                               // Display method
	{
		return getFirst() + ' ' + last + '\n';
	}
```

*****These overloaded operators are from another class, using (int, addition, and increment)*****

*****USE ONLY INSIDE CLASS*****

```
	void operator+(int n)                           // Arithmetic overloading - must be in class
	{
		day += n;                                   // increase day by number when using +
	}
	Date &operator=(int n)                          // Assignment operator overloading
	{
		day = n;
		return *this;                               // return current object
	}
	Date &operator++()                              // increment operator overloading
	{
		day++;
		return *this;
	}
};                                                  // Close class with semicolon

int main()
{
	Name myName("James", "Coan");                   // Create Object
	Name yourName;                                  // Create object with default constructor
	cout << myName.toString() << endl;              // Call objects toString function

	cout << "Enter a name: ";
	cin >> overloadName;                            // using input overloading
	cout << "\nYour name is: ";
	cout << overloadName << endl;                   // using output overloading
	return 0;
}
```

External overloaded operations – ADD IN GlOBAL AREA

```
bool operator==(Name &name1, Name &name2)              // Operator overloading - must be global
{
       return ((name1.getFirst() == name2.getFirst()) &&
              (name1.getLast() == name2.getLast()));
}

istream &operator>>(istream &input, Name &name)        // Input stream overloading
{
       string first, last;
       input >> first >> last;
       name = Name(first, last);
       return input;
}

ostream &operator<<(ostream &output, Name &name)       // Output stream overloading
{
       return output << name.getFirst() << ' ' << name.getLast();
}
```

DYNAMIC ARRAY:

```
int size = 2;
int *dynamicArray = new int[size];
dynamicArray[0] = 20;
cout << dynamicArray[0];                               // 20;
delete dynamicArray[size];                             // Must delete entire array
```

STATIC ARRAY:

```
const int size = 2;
int staticArray[size];
staticArray[0] = 20;
cout << staticArray[0];                                // 20;
```

2 DIMENTIONAL ARRAY:

```
const int rows = 2;
const int cols = 6;
int multiArray[rows][cols] = {{87, 85, 89, 95, 88, 90},
{79, 86, 85, 88, 93, 91}};
for(int r = 0; r < rows; r++)
{
       for(int c = 0; c < cols; c++)
              cout << temps[r][c] << " ";
       cout << endl;                                   // Output two rows of numbers
}
```

VECTOR:

```
vector<int> numbers;
numbers.push_back(5);                                  // Add 5
numbers.push_back(10);                                 // Add 10
numbers.pop_back();                                    // Remove 10
for(int i : numbers)
cout << i << endl;                                     // 5
```

SWITCH:

```
switch(test)                                           // Int or Char
{
       case 'y':
              cout << "Yes";
              return "Yes";
              break;
       default:
              cout << "No";
              return "No";
}
```

OUTPUT TO FILE:

```
#include <iostream>, #include <string>
#include <fstream>
int main()
{
	ofstream outFile("C:\\Users\\Simkyn\\Desktop\\file.txt", ios::out);  // write in file.txt
	outFile << "This text will appear in a file." << endl;
	outFile.close();
	return 0;
}
```

INPUT FROM FILE(STRING):

```
#include <iostream>, #include <string>
#include <fstream>
int main()
{
	int grade;
	string fileName;
	cout << "Enter a file name: ";
	getline(cin, fileName);                  // get fileName with grade info in it
	ifstream inFile(fileName.c_str(), ios::in);// get fileName, convert it to c-string for reading
	if(!inFile)
	{
		cout << "File not found!" << endl;
		exit(1);                         // cstdlib - causes program to close to os
	}
	while(!inFile.eof())                     // loop through each int till end of file (eof)
	{
		inFile >> grade;                 // store each int into grade
		cout << grade << endl;
	}
	inFile.close();                          // Need to close the file.
	return 0;
}
```

INPUT FROM FILE (CHAR):

```
#include <iostream>
#include <fstream>
int main()
{
	char c;
	ifstream inFile("C:\\Users\\Simkyn\\Desktop\\file.txt", ios::in);
	inFile.get(c);
	while(!inFile.eof())
	{
		cout << c;
		inFile.get(c);
	}
	inFile.close();
	return 0;
}
```

MODES

`ios::in`	Open for input operations.
`ios::out`	Open for output operations.
`ios::binary`	Open in binary mode.
`ios::ate`	Set the initial position at the end of the file. If this flag is not set, the initial position is the beginning of the file.
`ios::app`	All output operations are performed at the end of the file, appending the content to the current content of the file.
`ios::trunc`	If the file is opened for output operations and it already existed, its previous content is deleted and replaced by the new one.