Lecture Notes 4

Exceptions

- `try catch` – The statements used to `try` to execute some code and to `catch` an exception if one is thrown
- `throw` – Throws an exception that can be caught by a catch statement
    - Example:

```
struct OneInAHundred : std::exception{
    const char* what() const noexcept{
        return "One in a hundred!";
    }
};

void foo(int val){
    if((val % 100) == 0){
        throw OneInAHundred();
    }
}

int main(){
    bool NoException = true;
    try{
        foo(rand());
    }
    catch(std::exception &Ex){
        NoException = false;
        std::cout<<Ex.what()<<std::endl;
    }
    return 0;
}
```

- `noexcept` – A specifier stating that the function will not throw any exceptions, allows for compiler optimizations and gives information to the user of the function

Arrays

- Array – Collection of elements that are contiguous in memory and fixed in size at its creation
- Array Element Access – Elements are access using the `[]` operator, where the value between the brackets will be the element accessed (0 indexed)
- Static Array – The array size is fixed at compile time
  - Example:
```
int main(){
    int Array1[5];             // Static array of 5 ints
    int Array2[] = {1, 2, 3}; // Array of 3 ints

    Array1[2] = 7;    // Sets 3nd int of Array1 to 7
    return 0;
}
```

- Dynamic Array – The array size is fixed at run-time, must be created using `new`, and destroyed up using `delete`
  - Example:
```
int main(){
    int *Array1 = new int [5]; // Dynamic array of 5 ints
    int *Array2 = new int [3]{1, 2, 3}; // 3 int array

    Array1[2] = 7;    // Sets 3nd int of Array1 to 7
    delete [] Array1;
    delete [] Array2;
    return 0;
}
```
- Passing Arrays to Functions – Arrays are passed as pointers, but there is no way to determine their length, so a separate parameter usually needs to be passed
  - Input Example:
```
int foo(const int *arr, int cnt){
    int Total = 0;
    for(int Index = 0; Index < cnt; Index++){
        Total += arr[Index];
    }
    return Total;
}
int main(){
    int Array1[] = {1, 2, 3};
    int *Array2 = new int [3]{4, 5, 6};
    int Total1 = foo(Array1, 3);
    int Total2 = foo(Array2, 3);

    delete [] Array2;
    return 0;
}
```

- Output Example:
  ```
  void foo(int *arr, int cnt){
      for(int Index = 0; Index < cnt; Index++){
          arr[Index] = Index * 2;
      }
  }
  int main(){
      int Array1[3];
      int *Array2 = new int [3];
      int Total1 = foo(Array1, 3);
      int Total2 = foo(Array2, 3);
      for(int Index = 0; Index < 3; Index++){
          if(Array1[Index] != Array2[Index]){
              std::cout<<"Mismatch at "<<Index<<std::endl;
          }
      }
      delete [] Array2;
      return 0;
  }
  ```
- Static Arrays and Ranged `for` – Range based for loops can be used for static arrays declared in scope (not allowed for function parameters)
  - Example:
    ```
    int main(){
        int Array1[5] = {1, 2, 3, 4, 5};

        for(auto &Val : Array){
            std::cout<<Val<<std::endl;
        }
        return 0;
    }
    ```
- Array Terminators – A common method to mark the end of the array is to use a value not allowed in the data
  - NULL Termination – Literal strings and "C-style" strings are an array of characters with a null character `'\0'` that marks the end of the string
  - Example:
    ```
    int foo(const int *arr){
        int Total = 0;
        while(*arr >= 0){
            Total += *arr++;
        }
        return Total;
    }
    int main(){
        int Array1[] = {1, 2, 3, -1};
        int *Array2 = new int [4]{4, 5, 6, -1};
        int Total1 = foo(Array1);
    ```

```
        int Total2 = foo(Array2);

        delete [] Array2;
        return 0;
    }
```

- `std::array` – C++11 added the `std::array` type to the Standard Template Library, it has some advantages such as knowing the size and being able to be returned from functions
  - Example:

```
int foo(std::array<int, 3> arr){
    int Total = 0;
    for(auto &Val : arr){
        Total += Val;
    }
    return Total;
}
int main(){
    std::array<int,3> Array = {1, 2, 3};
    int Total = foo(Array);

    return 0;
}
```