Lecture 5 Notes

There are two special data types you will now learn: std::vector and std::string. Both of these data types work by using an array behind the scenes.

Vectors

The C++ vector is more generally known as a dynamic array. Like the array, the std::vector stores elements which are each accessible by an index number. Unlike the array, std::vector does not have a fixed capacity, it automatically expands as you store more elements into it. Sit back and watch the magic of std::vector:

Listing 1: raffle.cpp

```
#include <iostream>
#include <vector>
#include <cstdlib>
using namespace std;

int main()
{
    int n;
    vector<int> tickets;

    while (cin.good()) {
        cout << "Enter raffle ticket number to play " <<
        "(Press CTRL-D to begin raffle): ";
        cin >> n;
        tickets.push_back(n);
    }
    srand(time(0));
    cout << "The winner is ticket number: " <<
    tickets[rand() % tickets.size()] << endl;
    return 0;
}
```

This example mimics a raffle. You can enter virtually any number of integers, then the program randomly picks one of the integers that you entered.

Now unlike the basic data types (bool, char, int, double), the vector type is a special data type just like how the data type for cin (which is of type istream) is special. These special data types have functions "attached" to them which you can access with the "." operator (member-access operator), we call these type of functions "methods" or "member functions".

So the "good" function attached to cin will return true if cin has more user-input to be read.

For vectors, the "push_back" method adds a new element with a value.

Now most software libraries would use the term "push", "append", or "concat" (short for concatenate). The creators of C++ chose the term "push_back" for inserting an

element at the end of the dynamic array. Not a very elegant name, one day we will get a software library with sane naming conventions.

Whenever we declare a vector variable, we have to specify the data type of it's elements inside angle brackets. So a vector of double would be `vector<double>`, a vector of bool would be `vector<bool>`, a vector of integer vectors would be `vector< vector<int> >` (The spaces are needed, don't ask why).

Creating a function with a vector parameter can be naively like this:

```
int sum(vector<int> a)
{
    int i, s;

    s = 0;
    for (i = 0; i < a.size(); ++i) {
        s = s + a[i];
    }
    return s;
}
```

Remember that when you pass a variable to a function call, the value of that variable gets copied into the local environment of the function call. The original variable is preserved and it will remain unmodified by the function. The same is true for vectors. Passing a vector by value into a function will create a local copy of that vector for that function to temporarily operate on.

This is not a problem unless the vector reaches sizes in the tens of thousands. It is better practice to have the function accept a vector parameter "by reference", meaning that a pointer or reference, that is to say, a memory address of that vector is used as a parameter instead of the vector itself.

```
int sum(const vector<int> &a);
```

The new prototype of the sum function will operate on: a constant reference to a vector of integers. A constant reference as a parameter means that the parameter's value is shared by a variable located somewhere else in the program, the contents of this variable shall not be modified (hence it being constant). The syntax for calling such a function is unchanged from that of calling a pass-by-value function.

It is computationally faster to pass a vector by reference than by value and should always be done with compound data types like vectors. For basic data types (int, double, etc.) pass by value should be used.

Here is a table of the basic vector methods:

Method	Description	Parameters	Return Type
size	Returns number of elements		
push_back	Insert element at the end	value	
pop_back	Remove element at the end		
clear	Remove all elements		

In addition to these methods, you may use the [] operator with vectors to access an element. You may assign vectors to each other using the = operator too.