PIC 10A

Lecture 18: Vectors

Vectors

You may have wished at some point during your programming that you could have tens or hundreds of variables that you could keep saving data in.

Maybe you needed a variable number of variables. Perhaps in a loop a user is asked to enter data and you want to store all of it.

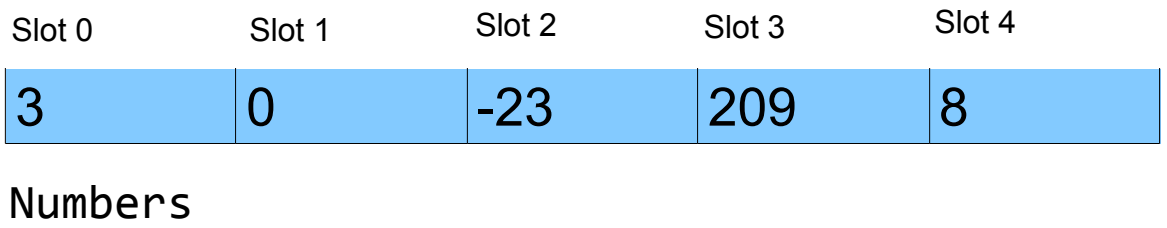
The solution to this kind of problems is vector or array variables. Today we will discuss vectors.

What is a vector?

Vector is a collection of variables of same data type that are accessed using a single name.

For example I might have a vector called `numbers` that stores many ints.

Graphically a vector might look like:



Idea is to access the data in the numbers vector via the slot number.

Declaring vectors

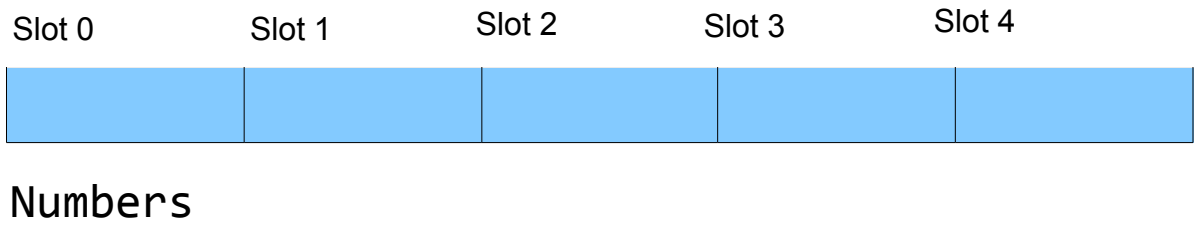
To declare a vector we use syntax:

vector<type> vector_name(vector_size);

Example:

```
vector<int> numbers(5);
```

This declares a vector called numbers with 5 slots, but this vector is uninitialized.



Vectors elements

The elements are accessed by their index (slot number).

To fill this vector we can write:

```
numbers[0] = 3;
numbers[1] = 0;
numbers[2] = -23;
numbers[3] = 209;
numbers[4] = 8;
```

We could output an elements value:

```
cout << "The 3rd element has value" << numbers[2];
```

Use them in assignments:

```
int number = numbers[0];
```

Individual elements act just like ordinary int variables!

Some noteworthy vector facts

Since the element of a vector is accessed by giving the slot number we will often use variables for the slot number.

If `i` is an int variable previously declared and initialized.

```
cout << numbers[i];
```

Vectors slots run from 0 to size-1. This is easy to make a mistake on.

Similar to the `length()` function strings have, vectors have a `size()` member function.

Filling a large vector

```
vector<double>my_list(10);

for (int i = 0; i < 10; i++)
{
cin >> my_list[i];
}
```

or

```
for (int i = 0; i < my_list.size(); i++)
{
cin >> my_list[i];
}
```

Outputting a vector

Note that we cannot write:

```
cout << my_list; // WRONG
```

I could write:

```
cout << my_list[2]; // Output the third element
```

To output a vector we write

```
for (int i=0; i < my_list.size(); ++i)
{
   cout << my_list[i] << " ";
}
```

Example

Create a vector filled with 100 random numbers in the range 1-100.

```
srand((int)(time(0)));
vector<int> random_numbers(100);

for (int i=0; i < random_numbers.size(); i++)
{
random_numbers[i]=rand()%100 +1;
}
```

Output the contents of the vector.

```
for (int i=0; i < random_numbers.size(); i++)
{
cout << random_numbers[i] <<  " ";
}
```

Classes and vectors

What kind of vectors can we have?

We have already seen vectors of ints and doubles. Elements of a vector could be strings, bools or even instances of a class.

```
vector<Point> many_points(10);

for (int i = 0; i < many_points.size(); i++)
{
many_points[i] = cwin.get_mouse("Click somewhere!");
}
```

Vectors and functions

```
double average(vector<int> numbers)
{
   int total=0;
   for(int i=0; i < numbers.size(); i++)
   {
      total+=numbers[i];
   }

   return (double)total / numbers.size();
}
```

average cont.

```
int main()
{
    vector<int> scores(100);
    double ave;

    cout << "Enter exam scores: ";

    for (int i=0; i < scores.size(); i++)
    {
        cin >> scores[i]
    }

    ave = average(scores);

    cout << "The average is: " << ave;

    return 0;
}
```

push_back

Vectors size is dynamic!

We don't need to know how big of a vector we need. We can always make it bigger later.

push_back member function adds number to the end of the vector numbers.

```
vector <int> numbers; // This vector has no elements!
int number;

while ( cin >> number)
{
numbers.push_back(number);
}
```

pop_back

pop_back removes the last element from the end of the vector.

Example:

numbers vector

```
numbers.pop_back();   // Note how pop_back takes no arguments!
```

resize

We can also resize an existing vector.

```
numbers.resize(20); //Resizes numbers vector to 20
                    // elements.
```

If numbers had more than 20 elements the extra elements were cut from the end of the vector.

If numbers had less than 20 elements the new elements are added to the end of the array and are blank.

We could even make the vector empty

```
numbers.resize(0);
```

Example

Lets write a function that takes as arguments two vectors and appends them together.

```
vector <int> append (vector <int> v1, vector <int> v2)
{
   for (int i = 0; i < v2.size(); i++)
   {
      v1.push_back(v2[i]);
   }
   return v1;
}
```

Note that we can return a vector.

Calling a function that returns a vector

```
int main()
{
   vector <int> v1;
   vector <int> v2;
   vector <int> v3;

   // Vectors v1 and v2 are filled.

   v3 = append(v1,v2);

   return 0;
}
```

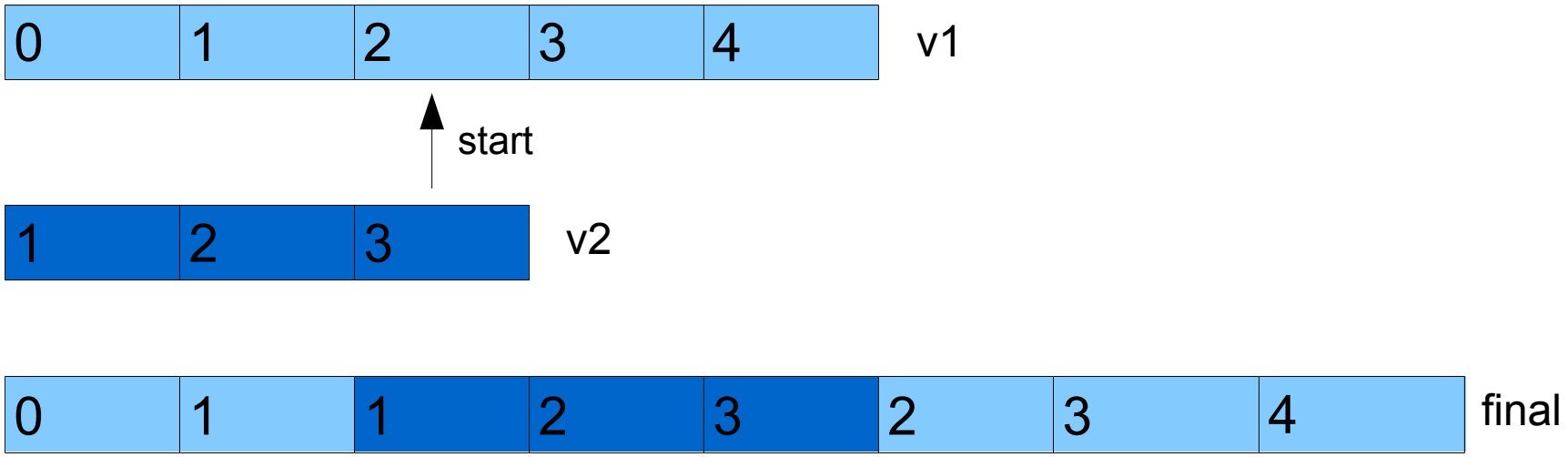
Has v1 changed in the main?

sub vector example

```
vector <int> sub_vector (vector <int> v,  int start,
int end)
{
   vector <int> sub;

   for (int i = start; i <= end ; i++)
   {
      sub.push_back(v[i]);
   }
   return sub;
}
```

Insert vector example



Plan:
Copy onto final elements from v1 until we get to start.

Copy elements from v2 to final.

Copy rest of elements of v1 to final. We start copying v1 elements at start.

insert example

```
vector <double> insert (vector <double> v1, vector <double> v2, int
start)
{
    vector<double> final;

    for (int i=0; i < start; i++)
    {
        final.push_back(v1[i]);
    }

    for (int i=0; i < v2.size(); i++)
    {
        final.push_back(v2[i]);
    }

    for (int i=start; i < v1.size(); i++)
    {
        final.push_back(v1[i]);
    }
    return final;
```