CSCI 135
Vectors

Vectors

Recall that various container types differ by homogeneity of element types, size characteristics, storage overhead, ordering, access, etc.

A **vector** is:

- A homogeneous sequence of elements that supports:
  - Fixed but changeable size (with some inefficiency)
  - Random access to elements (through index)
  - Minimal storage overhead
- The simplest and most general C++ container.
- Part of Standard Template Library (STL), like most containers
- Ex: [1.4, 2.7, 1.8], [“al”, “barb”, “carol”, “david”]
- Memory for vector may move from one address to another in RAM as vector grows in size; called **dynamic memory allocation**

Arrays vs. Vectors

Arrays

- C (thus, also C++)
- Fixed size (though not impossible to change)
- Random access using []
- Programmer needs to keep track of size
- Can't be assigned/copied (*e.g.*, `arr1 = arr2;` won't work)
- No bounds checking
- More efficient than vectors
- Map well to images, matrices, etc.

Vectors

- C++ only
- Size can grow/shrink (with some inefficiency)
- Random access using []
- size() function can be used to determine size
- Assignment statement (*e.g.*, `v1 = v2;` will copy all of v2 to v1)
- Bounds checking w/ some fcts.
- Less time- and space-efficient
- Map well to lists of words, symbol processing, etc.

Declaring a Vector

vector<Base_Type>

where Base_Type is any type (int, float, string, another vector, etc.)

Examples:

```
#include <vector>
vector<int> wv;          0-element vector of ints
vector<int> xv(16);      16-element vector
vector<string> yv;
enum Color {red, green, blue};
vector<Color> cv(256);
```

Default initial value is 0s for vector of ints; various other values for other types (but better to not rely on defaults).

⚠Don't confuse types with variables/objects – *e.g.*, cv is a variable whose type is a vector of Colors (so, each element of cv has type Color).

Templates

A **template** is C++'s way of supporting code that works for any type (other languages: generics).

So, this code works for vector<int>, vector<float>, vector<MyType>,... (as long as + is overloaded for that type):

```
for (int i = 0; i<v1.size(); i++)
  v1[i] = v1[i]+v2[i];
```

Formally, a template is like a function on types (*e.g.*, Types $\rightarrow$ Types), and vector<Base_Type> can be thought of as a function taking Base_Type as an argument and returning another type (a vector of Base_Type's).

Vector Usage

- Empty on initial declaration
- Retrieve size using `size()` operator. Can not access n+1'st element (index n) of a n-element vector!
- Use `[<index>]` to access element (indices start at 0)
- Use `push_back` to insert element at end of vector (and increment size)
- Use `pop_back` to delete element at end of vector (and decrement size)
  ⚠You *might* still be able to access deleted element for a little while (until OS gets around to actually reclaiming storage), but don't!

Vector Operations (partial list)

- `v.size()`: returns number of elements in vector v
- `v[i]` ($0 \leq$ i $<$ v.size()): i'th element of v (indexing); undefined behavior if i is not in range.
- Assignment operator overloaded to assign each element of operator; *e.g.*, `v = w` would make a copy of each element of w and assign it to the corresponding element of v.
- `v.push_back(x)`: expands v by one element at end, with value x
- `v.pop_back()`: removes/destroys last element from v (and no longer accessible), reducing v's size by 1
- Storage operators (when memory issues matter):
  - `v.reserve(n)`: Reserves n elements of storage for vector v, initialized as before. Also used for changing size, but reducing space may not actually do anything.
  - `v.resize(n)`: Changes the size of the vector (data lost if making v smaller).
  - `v.capacity()`: Number of elements currently allocated for v ($\geq$ v.size()). NOT same as size()!

Vector Example

Problem: Input a sequence of ≤ 16 non-negative integers, store, and output them in reverse order. A negative input indicates end.

```
vector<int> v;
cout << ``Enter list of positive numbers ``
     << ``(negative number to signify end)\n'';
int num=0;
int i=0;
while ((i<16) && (num >= 0)) {
  // inv: v[0...(i-1)] has input data
  cin >> num;
  v[i] = num;                    Fill up v
  i = i+1;
};                               Either i≥16 or num<0 exits loop
// assert: v[0..15] has input data
cout << "Reverse Order:" << endl;
for (int i=15; i>=0; i--)
  cout << v[i] << endl;
```

Vector Example

Problem: Input a sequence of ≤ 16 non-negative integers, store, and output them in reverse order. A negative input indicates end.

```
vector<int> v;
cout << ``Enter list of positive numbers ``
     << ``(negative number to signify end)\n'';
int num=0;
int i=0;
while ((i<16) && (num >= 0)) {
  // inv: v[0...(i-1)] has input data
  cin >> num;
  v[i] = num;                    Fill up v
  i = i+1;
};                               Either i≥16 or num<0 exits loop
// assert: v[0..15] has input data
cout << "Reverse Order:" << endl;
for (int i=15; i>=0; i--)
  cout << v[i] << endl;
```


This wont work (why not?)

Vector Example - Attempt 2

Problem: Input a sequence of ≤ 16 non-negative integers, store, and output them in reverse order. A negative input indicates end.

```
vector<int> v(16);                Initialized to 16 0's
cout << "Enter list of positive numbers "
     << "(negative number to signify end)\n";
int num=0;
int i=0;
while ((i<16) && (num >= 0)) {
  // assert: v[0...(i-1)] has input data
  cin >> num;
  v[i] = num;                     Fill up v
  i = i+1;
};                                Either i≥16 or num<0 exits loop
cout << "Reverse Order:" << endl;
for (int i=15; i>=0; i--)
  cout << v[i] << endl;
```

Vector Example - Attempt 2

Problem: Input a sequence of ≤ 16 non-negative integers, store, and output them in reverse order. A negative input indicates end.

```
vector<int> v(16);
cout << "Enter list of positive numbers "
     << "(negative number to signify end)\n";
int num=0;
int i=0;
while ((i<16) && (num >= 0)) {
  // assert: v[0...(i-1)] has input data
  cin >> num;
  v[i] = num;
  i = i+1;
};
cout << "Reverse Order:" << endl;
for (int i=15; i>=0; i--)
  cout << v[i] << endl;
```

Initialized to 16 0's

Fill up v

Either i≥16 or num<0 exits loop

Doesn't work properly if < 16 elements (quick homework to fix!)

Vector Example - Revised

Problem: Same as above, but with arbitrarily many elements

```
vector<int> v;                    Null vector initially
// Store a list of positive integers in v
cout << "Enter list of positive numbers "
     << "(negative number to signify end)\n";
int num=0;
while (num >= 0) {
   cin >> num;
   v.push_back(num);              Also increments v's size
   };
// Print v in reverse order
cout << "Reverse Order:" << endl;
for (int i=v.size()-1; i>=0; i--)
   cout << v[i] << endl;
```

Example: Delete Element (Pseudocode)

Problem: Given a vector of ints v, and an element n, delete that element from v and return the position.
Algorithm:

1. Scan through v, until n is found, denoting that index as pos.
2. Move each element with index>pos up 1
3. Delete last element of v.

Example: Delete Element (Pseudocode)

Problem: Given a vector of ints v, and an element n, delete that element from v and return the position.
Algorithm:

1. Scan through v, until n is found, denoting that index as pos.
   Specification problems:
   - What if v contains multiple n's?
     $\Rightarrow$ Change specification to first instance of n in list.
   - What if n is not found?
     $\Rightarrow$ Change specification to not change v and return -1 in this case.
2. Move each element with index>pos up 1
3. Delete last element of v.

Example: Delete Element

```
vector<int> v;
... // code to initialize v here
int pos=0;
while ((pos < v.size()) && (n != v[pos]))
   pos++;
if (pos==v.size())                    n is not in v
   return(-1);
// Assert: n == v[pos]
for (int i=pos+1; i<v.size(); i++)
   v[i-1]=v[i];                       easy to be off by 1!
v.resize(v.size()-1);                 or v.pop_back()
return(pos);
```

Compare to arrays version!

Short Circuit Evaluation

Lazy Evaluation: Evaluate expressions only if needed for result (not supported in C/C++)
Short Circuit Evaluation: Special case of lazy evaluation, where boolean subexpressions are evaluated only if needed.
Examples (P is some arbitrary condition/expression):

- `(x==0) && P`: The P never gets evaluated if x is not 0 (since the result is false regardless)
- `(x==0) || P`: The P never gets evaluated if x is 0 (since the result is true regardless)

Iterating Through Vectors

Suppose a vector has *n* elements (indexed 0 ... n-1), and we wish to iterate through the vector until its end (so a variable i is incremented on each iteration).

- `while ((v[i]==...) && (i < n)) ...`
  $\Rightarrow$ error if i==n since v[n] is out of bounds.
- `while ((i < n) && (v[i]==...)) ...`
  $\Rightarrow$ `i < n` evaluates to false when i==n and v[n] is never accessed.

⚠ Always check that an index is in bounds **before** accessing vector.

Other Facts About Vectors

- A single assignment statement such as `v1 = v2` will make a copy of each element of v2 and store it at the corresponding index of v1, *provided* the type of the vector elements makes a copy on assignment (not so for all types!).
- Vector elements can be of an arbitrary type. *i.e.*, a 2-D matrix is just a vector of vectors:

```
typedef vector<int> T;      T is the type of a vector of ints
vector<T> x;                x is a variable of type 'matrix'
```

OR

```
vector<vector<int> > x;     Note: space is needed
```

Exercises

1. Given a sentence represented as a vector of words, change every "a" to an "an" (and vice versa), depending on whether the first letter of the following word starts with a vowel. Yes, this isn't exactly what English rules are, but we will live with it.
2. Given a sentence, output a histogram of word lengths.
   Ex: with the input "To be or not to be that is the question", output:

```
Length     1  2  3  4  5  6  7  8
# Words    0  6  2  1  0  0  0  1
```

3. Write a function that has 4 parameters: a vector of names, a vector of salaries, a name x, and a salary. The two vectors are 'synched'. Store x's salary in the salary parameter. If x is not in the vector return -1; otherwise return 0.
4. Given the previous function, write a function that prints the name of the best-paid person.
5. Write the insertion and selection sorts from earlier using vectors instead of arrays.