COSC 220: Data Structures in C++

Data Structures: The Vector Class

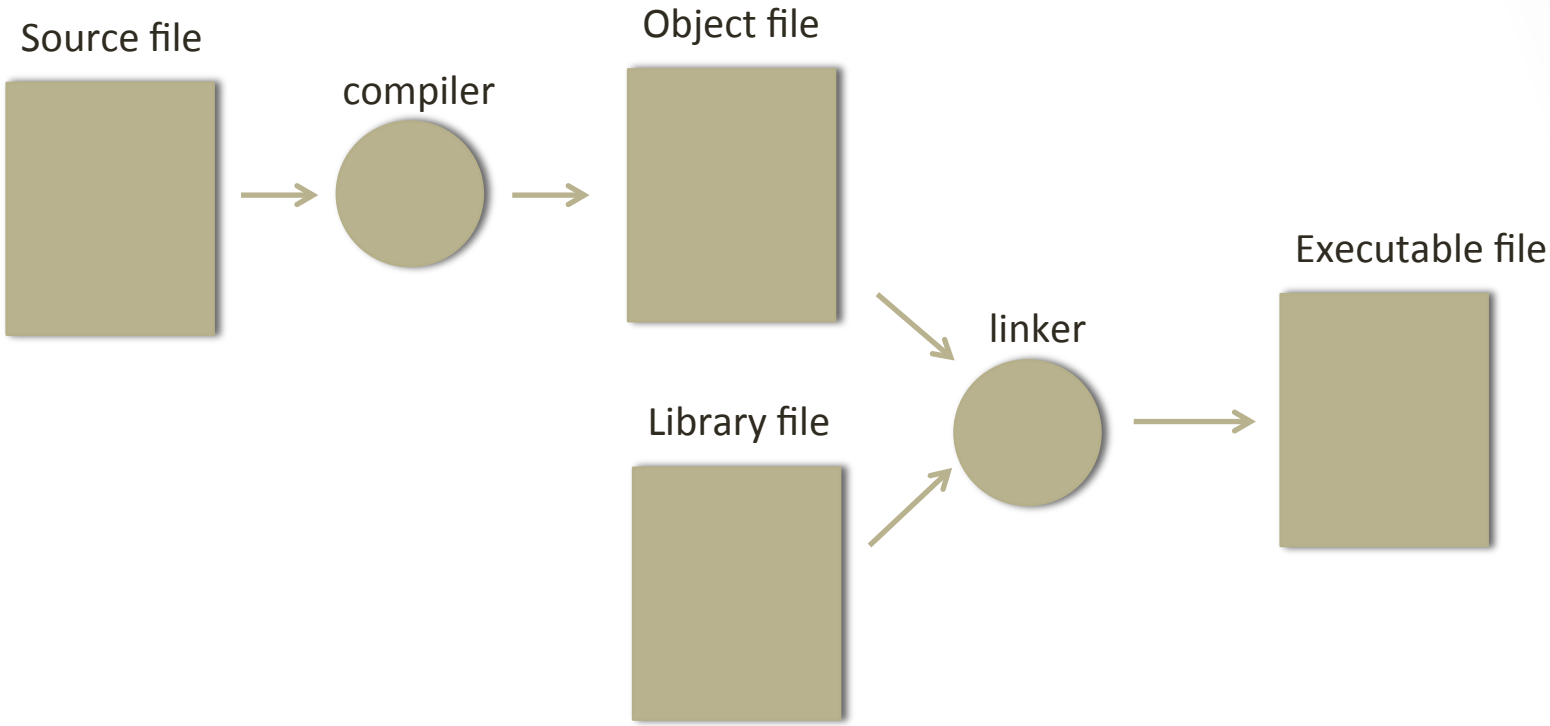
This week (week 7):
5 data structures will be discussed

The first is the Vector Class

Make sure the Static Stanford Libraries are working on your computer!!

COSC 220: Data Structures in C++

Review of the Compilation Process:



1. **Source file**: text of your code
2. **Compiler** → translate source to **object file**
3. **Object file**: machine-language instructs
4. Object file combined with other object files via Linker → **Executable file**
5. Other object files: predefined obj files called **libraries**
6. **Libraries**: contain machine-lang instructs for various operations req'd by program
7. **Linking**: combining object files into an executable

COSC 220: Data Structures in C++

Data Structures: The Vector Class

Data Structures: assembled to form hierarchies
atomic data types: int, char, double are building blocks
Larger structures in open ended process → *Data Structures*

A type def'd in terms of behavior instead of representation
called: *Abstract Data Type* (ADT)
Think more holistically about Data Structures

Next few lectures, 5 data structures intro'd:
Vector, Stack, Queue, Map, and Set

These are collection classes: contain collections of values of simpler types

COSC 220: Data Structures in C++

The Vector Class

Separating behavior from underlying implementation:
→ *Encapsulation*

Encapsulation is fundamental principle in OOP

- **Simplicity:** hiding internal rep'n from client → fewer details for client to understand
- **Flexibility:** class def'd by its public behavior, implementer is free to change underlying private representation. can change implementation if interface is unchanged
- **Security:** interface bndry is wall protects implemention. from client and vice versa. If client has access to representation, they can change values of underlying data structures

COSC 220: Data Structures in C++

The Vector Class

To use the *Vector class*, include the interface in your program: #include “vector.h”

Collection classes in next set of lectures draw much from:
C++ *Standard Template Library* (STL)
Advanced and powerful set of C++ classes

The Vector class is valuable and uses C++ array structure

Arrays in C++ difficult to work with. i.e. fixed size, size is ambiguous, cannot insert and delete elements, no error checking for array locations that exist or not etc.

COSC 220: Data Structures in C++

The Vector Class

C++ collection classes utilize a *base type*
e.g. Vector<int> ← rep's a vector whose elt's are int's
Use angle bracket notation for *base type*

Classes w/ base type called: parameterized classes.
In C++ paramet'd classes called *templates,* reflecting how C++ treats Vector<int>, Vector<char>, and Vector<double> as indep. classes with common structure.
Vector → acts as template for generating a family of classes.

Declaring a Vector Object:
Vector objects declared ~'ly to types:
Vector<int> vec; ← declares Vector vec with integer base

COSC 220: Data Structures in C++

The Vector Class

Vector Operations

declare Vector → empty Vector variable

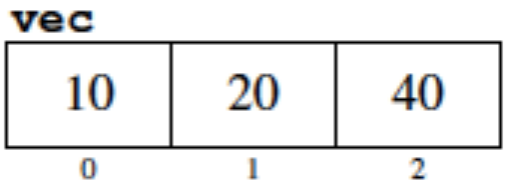
the *add* method adds new elt. to end of vector

vec.add(10);

vec.add(20);

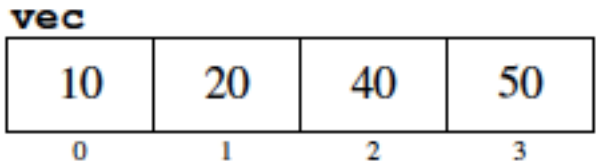
vec.add(40);

vec is now 3 elt. vector:

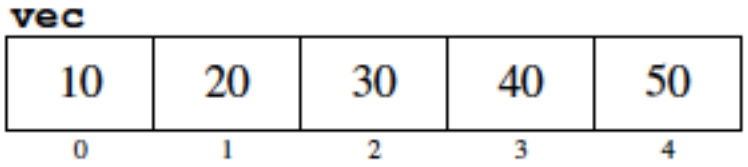


vec.add(50);

yields:



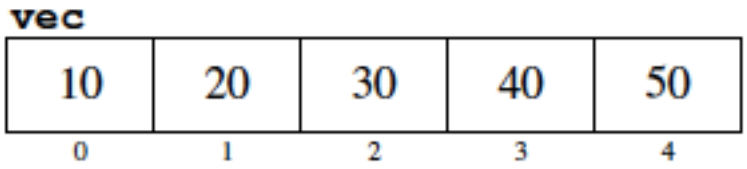
vec.insert(2, 30); ← index # and value, inserts value before that index

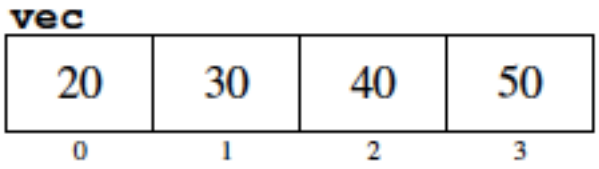


COSC 220: Data Structures in C++

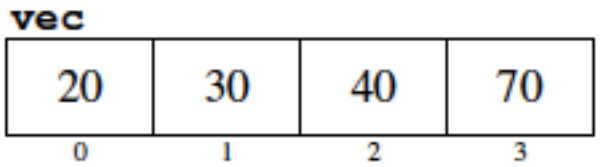
The Vector Class

removing elts: *remove* method
vec.remove(0)





Select and modify elts: *get* and *set*
get: takes index # and returns the value
set: takes index # and changes the value
vec.get(2) → returns value 40
vec.set(3, 70); →



Easier to use Vector class indexing like w/ arrays: vec[i]
vec[3] = 70; ← sets 3rd elt to value 70.

COSC 220: Data Structures in C++

The Vector Class

can loop through indices:

```
for (int i = 0 ; i < vec.size(); i++) {
       loop body, current vec elt is v[i]
}
```

Class Exercise: write out code that creates a vector vec of length 4 with elements 20, 30, 40, 70 and use a loop to print it out as a comma separated list.

COSC 220: Data Structures in C++

The Vector Class

```
cout << “[”;
for (int i = 0; i < vec.size(); i++) {
if (i > 0) cout << “, ”;
cout << vec[i];
}
cout << “]” << endl;
```

yields: `[20, 30, 40, 70]`

COSC 220: Data Structures in C++

The Vector Class

Passing a vector object as a parameter:
In general, pass by reference. Saves data space and simplifies coding.

```
void printVector(Vector<int> & vec ) {
   cout << "[";
   for (int i = 0; i < vec.size(); i++) {
      if (i > 0) cout << ", ";
      cout << vec[i];
   }
   cout << "]" << endl;
}
```

COSC 220: Data Structures in C++

The Vector Class

Class Exercise:
Write a function `removeZeroElements(Vector<int> & vec)` that removes all zeros elts from a vector

hint, use the remove method

COSC 220: Data Structures in C++

The Vector Class

Class Exercise:
Write a function `removeZeroElements(Vector<int> & vec)` that removes all zeros elts from a vector

hint, use the remove method

```
void removeZeroElements (Vector<int> & vec ) {
   cout << "[";
   for (int i = vec.size() - 1; i >=0; i--) {
      if (vec[i] == 0) vec.remove(i);
   }
}
```

loop starts at end to ensure that removing an elt doesn't change order of unchecked elts.
Other ways to do this as well.

COSC 220: Data Structures in C++

The Vector Class

vector of predefined size

Method 1: use loop to create vect of zeros of len 18

```
const int N_HOLES = 18;

Vector<int> golfscores;
for (int i = 0; i < N_HOLES; i++) {
  golfScores.add(0);
}
```

Method 2: pass size as a parameter

```
const int N_HOLES = 18;

Vector<int> golfscores(N_HOLES);
```

COSC 220: Data Structures in C++

The Vector Class

Program to display lines in a file in reverse order

see: ReverseFile.cpp from fredpark.com/teaching

run it on the Palindromes.txt file

COSC 220: Data Structures in C++

The Vector Class

Program to count the letter frequencies in a file
see: LetterFrequency.cpp

use it on George Eliot's *Middlemarch*
can be found on my website as Middlemarch.txt

COSC 220: Data Structures in C++

The Vector Class

breakdown of the letter frequency code

see code and explanation in code

Letter ‘A’ has ASCII code 65. see pg 22 in book.
ch-‘A’ → maps to index
e.g. if ch == ‘A’, ch-‘A’ → 0, 1st index in Vector
e.g. if ch == ‘B’, ch-‘A’ → 1, 2nd index in Vector etc.

COSC 220: Data Structures in C++

The Vector Class

Constructors for the vector class

ReverseFile program defines empty vector like so:

```
Vector<string> lines;
```

LetterFrequency program does this:

```
Vector<int> letterCounts(26);
```

Normal types not auto'ly initialized but for instances of a class they are. Invoke special method: ***constructor***

Default constructor: case no arguments supplied
Type of constructor depends on type used e.g. string, bool, int etc.
default values for int = 0, bool = false, char ASCII value = 0.

COSC 220: Data Structures in C++

The Vector Class

constructors table:

TABLE 5-1 Entries in the `vector.h` interface

Constructors

`Vector<`*type*`>()`	Creates an empty vector.
`Vector<`*type*`>(`*n*`, `*value*`)`	Creates a vector with *n* elements, each of which has the specified value. If *value* is missing, each element is initialized to its default value.

Methods

`size()`	Returns the number of elements in the vector.
`isEmpty()`	Returns **true** if the vector is empty.
`get(`*index*`)`	Returns the element at the specified index position. Attempting to get the value of an element outside the vector bounds generates an error.
`set(`*index*`, `*value*`)`	Sets the element at the specified index to the new value. Attempting to set the value of an element outside the vector bounds generates an error.
`add(`*value*`)`	Adds a new element at the end of the vector.
`insertAt(`*index*`, `*value*`)`	Inserts the new value before the specified index position.
`removeAt(`*index*`)`	Deletes the element at the specified index position.
`clear()`	Removes all elements from the vector.

Operators

vec`[`*index*`]`	The `Vector` class extends the square bracket operators so that clients can use array notation to select elements. Attempting to select an element outside the vector bounds generates an error.
v_1 `+` v_2	Concatenates v_1 and v_2 and returns a new vector containing the combined elements.
vec `+=` e_1, e_2, . . .	Adds the elements e_1, e_2, and so on, to the end of the vector `vec`.

COSC 220: Data Structures in C++

The Vector Class

Vector Operations:

+: concatenation

+=: adds what's on RHS to end of vector

```
Vector<char> v1;
v1 += 'A', 'B', 'C';

//C++11
Vector<char> v1 = { 'A' , 'B' , 'C' };
```

COSC 220: Data Structures in C++

The Grid Class

2D structures → Grid Class

TABLE 5-2 Entries exported by the `grid.h` interface

Constructors

`Grid<`*type*`>()`	Creates an empty grid. Clients who use the default constructor must specify the grid dimensions by calling `resize`.
`Grid<`*type*`>(`*rows*`, `*cols*`)`	Creates a grid with the specified number of rows and columns. Each element is initialized to the default value for the type.

Methods

`numRows()` `numCols()`	These methods return the number of rows and the number of columns, respectively.
`inBounds(`*row*`, `*col*`)`	Returns `true` if the specified row and column coordinates are inside the grid.
`get(`*row*`, `*col*`)`	Returns the element of the grid that appears at the specified row and column.
`set(`*row*`, `*col*`, `*value*`)`	Sets the element at the specified grid coordinates to the new value.
`resize(`*rows*`, `*cols*`)`	Changes the dimensions of the grid as specified by the *rows* and *cols* parameters. Any previous contents of the grid are discarded.

Operators

grid`[`*row*`][`*col*`]`	The `Grid` class extends the square bracket operators so that clients can use array notation to select individual elements of the grid.

COSC 220: Data Structures in C++

The Grid Class

2D structures → Grid Class

example: magic squares

see magic squares code

code is not perfect. what can be improved? what cases will it possibly not work. what's the fix?

COSC 220: Data Structures in C++

The Stack Class

The Stack Class

Simplest collection class
stack → storage for collection of data values
restriction: values removed in opposite order they were added
Last item added → first one removed

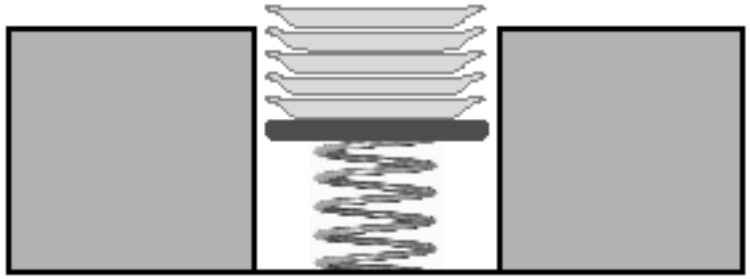
pushing: adding new value
popping: removing most recent item (not a dance move in this context!!!)
LIFO: last in first out

COSC 220: Data Structures in C++

The Stack Class

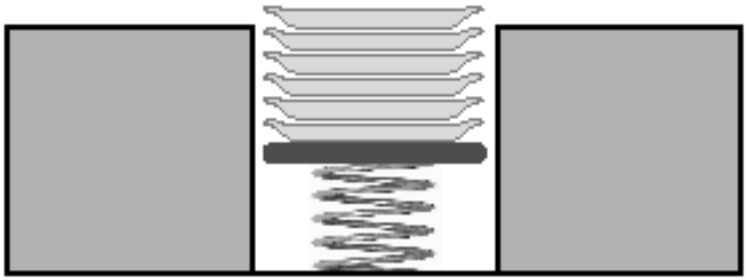
The Stack Class

like adding dishes to a stack of them in the CI



add a plate and the stack pushes down the spring

someone grabs a plate, it's the last one added



COSC 220: Data Structures in C++

The Stack Class

The Stack Class

The Stack Class important since nested function calls form a stack frame.
cf. COSC 120 lecture
stack frame function call

similar to Vector class
create instance of stack class via: Stack<int>, Stack<string> etc

can create a stack of self defined classes:
Stack<your_class> i.e. Stack<Vector<int>> etc.

COSC 220: Data Structures in C++

The Stack Class

The Stack Class

TABLE 5-3 Entries exported by the `stack.h` interface

Constructor

`Stack<`*type*`>()`	Creates an empty stack capable of holding values of the specified type.

Methods

`size()`	Returns the number of elements currently on the stack.
`isEmpty()`	Returns `true` if the stack is empty.
`push(`*value*`)`	Pushes *value* on the stack so that it becomes the topmost element.
`pop()`	Pops the topmost value from the stack and returns it to the caller. Calling `pop` on an empty stack generates an error.
`peek()`	Returns the topmost value on the stack without removing it. Calling `peek` on an empty stack generates an error.
`clear()`	Removes all the elements from a stack.

COSC 220: Data Structures in C++

The Stack Class

The Stack Class

Exercise 1: Write a program that uses the stack class to reverse a sequence of integers read from the console one per number line. Use a sentinel to stop input.
Repeat for a list of words entered one per line in the console.

Exercise 2: Use the grid class to create a tic-tac toe game. start with simple output of the grid as text based graphics

Exercise 3: Improve the magic squares code.