

Introductory Project



WATERLOO
ENGINEERING

Douglas Wilhelm Harder, M.Math. LEL
Vajihollah Montaghani, M.A.Sc.
Department of Electrical and Computer Engineering
University of Waterloo
Waterloo, Ontario, Canada

ece.uwaterloo.ca
dwharder@alumni.uwaterloo.ca
vajih.montaghani@gmail.com



Outline

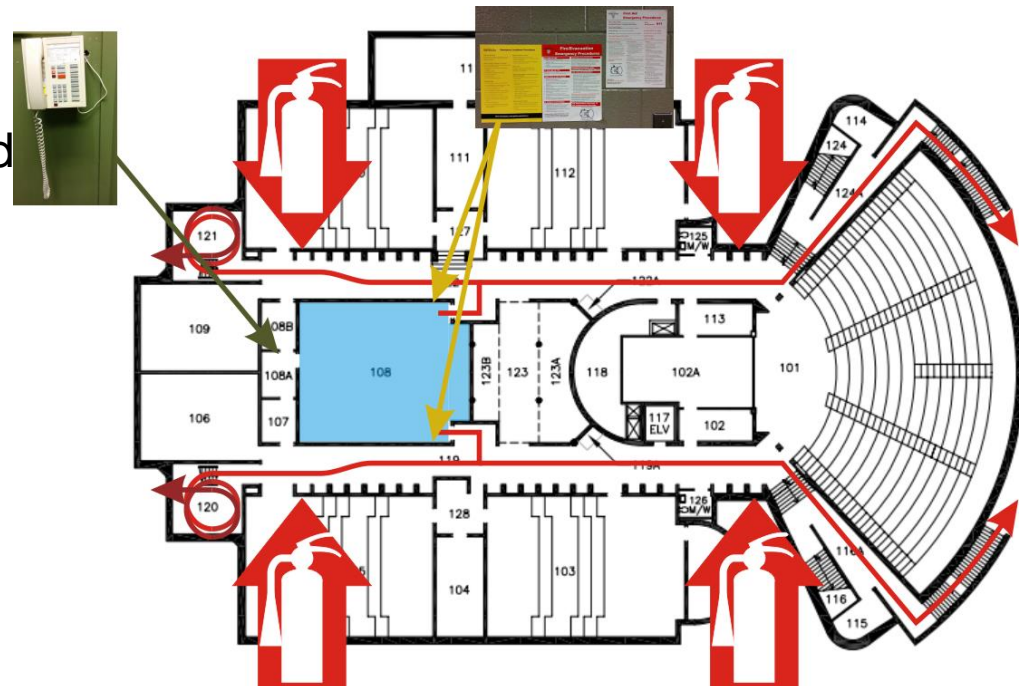
In this introductory project, we will:

- Review pass-by-reference and pass-by-value
- Describe remote logins
- Give an introduction to Unix
- Create a hello world program
- Create a header file for an Array class
- Discuss bug fixing
- Introduce five other functions and three statistical functions
- Other features of classes
- Templates
- Our testing environment

Safety first

This is RCH 108 and in case of an emergency:

- There two room exits and four available building exits
 - Exit promptly and orderly if a fire alarm goes off
- Emergency information posters are at both doors
 - The location of the first-aid kit is listed on the *First Aid Emergency Procedures* poster
- A phone is in the back room
 - Call 911 in an emergency
- Fire extinguishers are located in the hallways towards the building exits





Project requirements

Project requirements include:

- Submissions are individual
- You can work together, but the authoring of code must be done alone
- Projects 1, 2, 3, 4 are due at 22:00 (10:00 PM) on the Tuesday immediately following the laboratory corresponding with the project
- Project 5 is due at midnight on the last day of class
- Project submissions are through Learn
- The Drop box is open for late submissions until 6:00 AM the next morning
- All submissions are compiled and graded on `ecelinux.uwaterloo.ca`
- Project details at `ece.uwaterloo.ca/~dwharder/aads/Projects/`



Outline

In this laboratory, we will:

- Review pass-by-value versus pass-by-reference
- Get you logged into `ece1linux`
- Have you save, compile and execute a *Hello World!* program
- We will introduce you to C++ in Visual Studio
- The introduction will be through an Array class
 - We will implement and add additional functionality
 - We will create executable functions using this class
 - We will use this class in our testing framework
- We will then copy this class to `ece1linux` and test it there
- We will conclude with topics such as:
 - Recursion



Pass-by-value and pass-by-reference

Normally, when you pass an argument to a function, the parameter in the function is assigned the value of the argument, but the parameter is a different variable

- Changes to the parameters of a function do not affect the arguments

```
#include <iostream>

void swap( int x, int y ) {
    int tmp = x;
    x = y;
    y = tmp;
}

int main() {
    int m = 5, n = 17;
    swap( m, n );
    std::cout << "m = " << m << " and n = " << n << std::endl;
    return 0;
}
```



Pass-by-value and pass-by-reference

If you want changes in the functions to affect the original arguments, we must pass the arguments by reference:

```
#include <iostream>

void swap( int &x, int &y ) {
    int tmp = x;
    x = y;
    y = tmp;
}

int main() {
    int m = 5, n = 17;
    swap( m, n );
    std::cout << "m = " << m << " and n = " << n << std::endl;
    return 0;
}
```



Pass-by-value and pass-by-reference

In object-oriented programming, a pass-by-value of an object must make a complete copy of the object

- This may be prohibitively expensive
- Passing an Array object by value makes a complete copy of the array and that copy is deleted when the function exits...
- We will therefore often pass objects by reference
 - Problem: objects passed by reference may be changed—what if we don't want the function to change the object
 - Solution: *pass-by-constant-reference*

The notation for pass-by-constant-reference is:

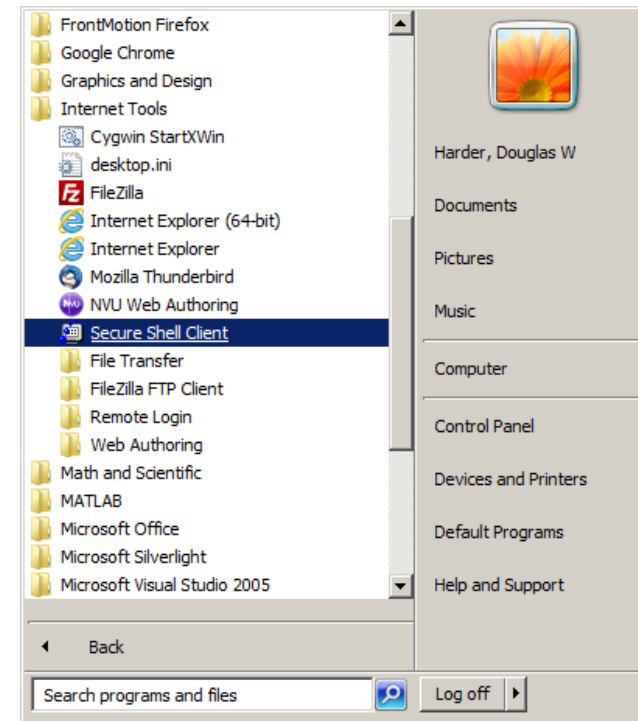
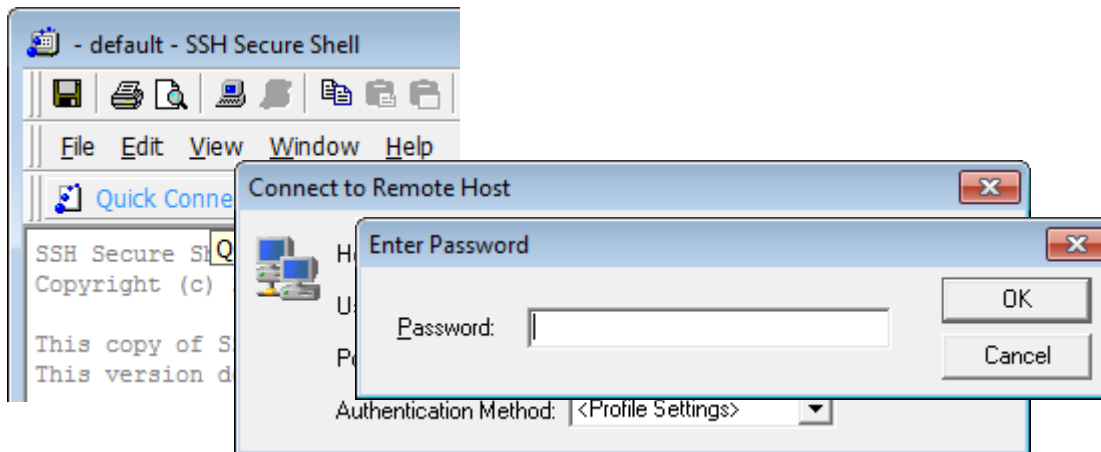
```
void f( int const &x ) {  
    // x refers to the argument, but you cannot change the value of x  
    x = y;  
    y = tmp;  
}
```

- We will discuss this later in relation to objects

Remote logins

You can log into a Unix machine by using a terminal program

- Launch the Secure Shell Client (SSh)
- All Nexus terminals have SSh installed under *Internet Tools*
- Select the *Quick Connect* button
- Set the Host Name to `ecelinux.uwaterloo.ca`
- Set the User Name to your uWaterloo User ID (e.g., `dwharder`)
- Select *Connect* and use your Nexus password

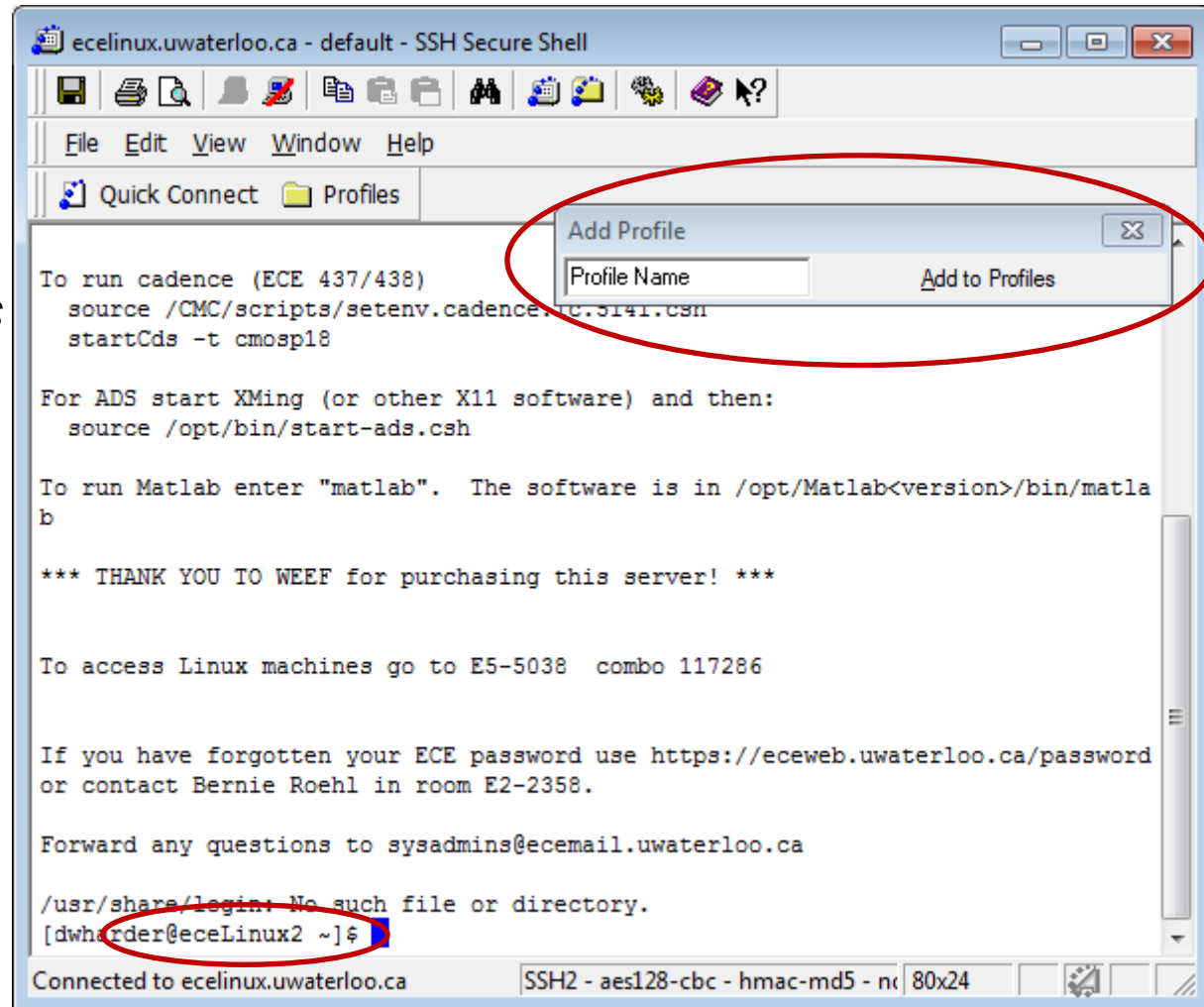


Remote logins

You have now established a remote *shell* user interface

You can optionally add a profile to save your Host and User names under *Profiles*

You will be randomly logged onto one of
 eceLinux1
 eceLinux2
 eceLinux3
 etc.





Remote logins

A shell passes commands to the host machine

- You enter commands at the prompt (\$)
- The shell passes the commands to the host computer
- There, the shell converts your commands into system calls

```

To run cadence (ECE 437/438)
source /CMC/scripts/setenv.cadence.ic.5141.csh
startCds -t cmosp18

For ADS start Xming (or other X11 software) and then:
source /opt/bin/start-ads.csh

To run Matlab enter "matlab". The software is in /opt/Matlab<version>/bin/matlab

*** THANK YOU TO WEEF for purchasing this server! ***

To access Linux machines go to E5-5038 combo 117286

If you have forgotten your ECE password use https://eceweb.uwaterloo.ca/password
or contact Bernie Roehl in room E2-2358.

Forward any questions to sysadmins@ecemail.uwaterloo.ca

/usr/share/login: No such file or directory.
[dwharder@eceLinux2 ~]$ █
    
```

Connected to ecelinux.uwaterloo.ca SSH2 - aes128-cbc - hmac-md5 - nc 80x24



Remote logins

Aside:

- If you have Linux at home, you can log in remotely through the shell:

```
$ ssh dwharder@ecelinux.uwaterloo.ca
```



Remote logins

When you log onto `ece1linux`, the current (or *working*) directory you are viewing is your home directory, e.g.,

`/home/dwharder`

Windows uses *drives* to differentiate between different storage devices, e.g., `C:\Users\dwharder`

- Unix makes no such distinction
- Drives are subdirectories of the `/mnt` directory



Introduction to Unix

We can ask for a listing of files and directories in the working directory:

```
$ pwd
```

```
/home/dwharder
```

```
$ ls
```

```
$
```

pwd	P rint w orking d irectory
ls	L ist directory contents



Introduction to Unix

Let's create a directory:

```
$ mkdir ece250
```

```
$ ls
```

```
ece250
```

```
$ cd ece250
```

```
$ mkdir lab0
```

```
$ cd lab0
```

```
$ pwd
```

```
/home/dwharder/ece250/lab0
```

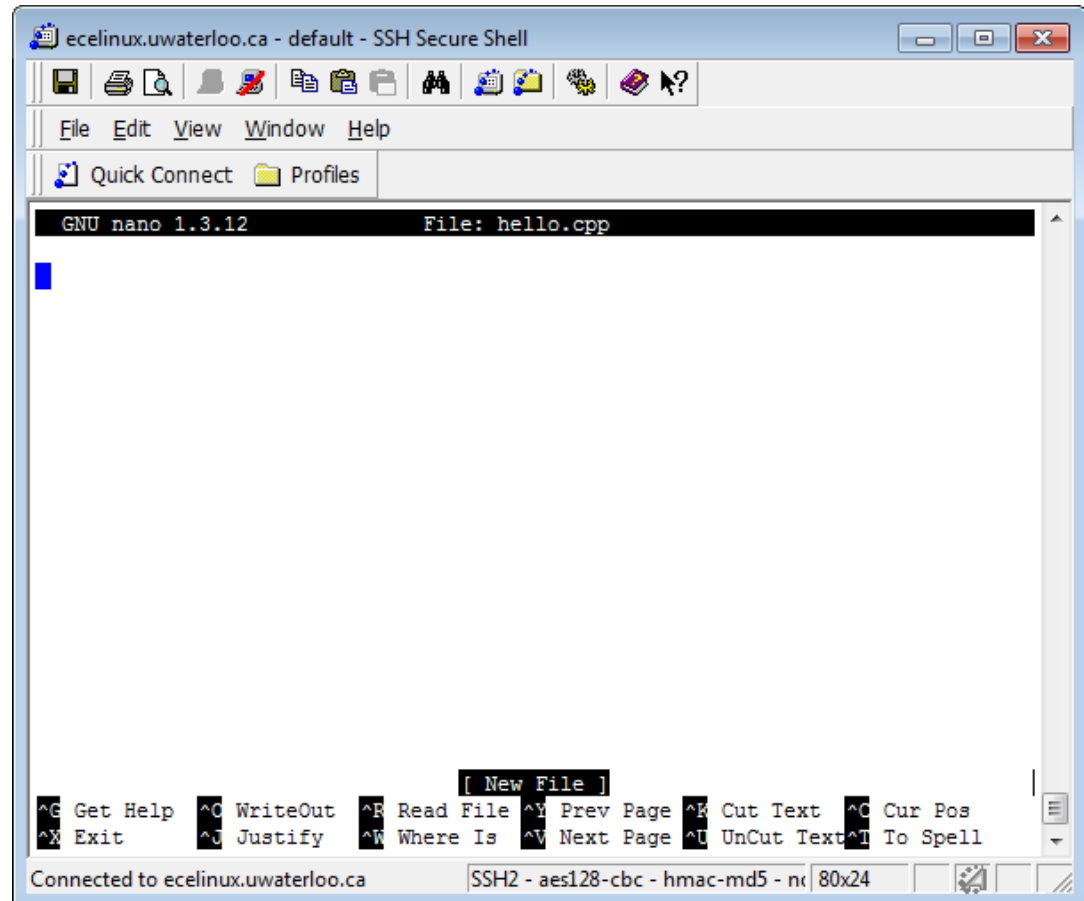
cd **Change directory**



Hello world!

The next step is to edit a file:

```
$ nano hello.cpp
```





Hello world!

Many of the important commands are listed at the bottom

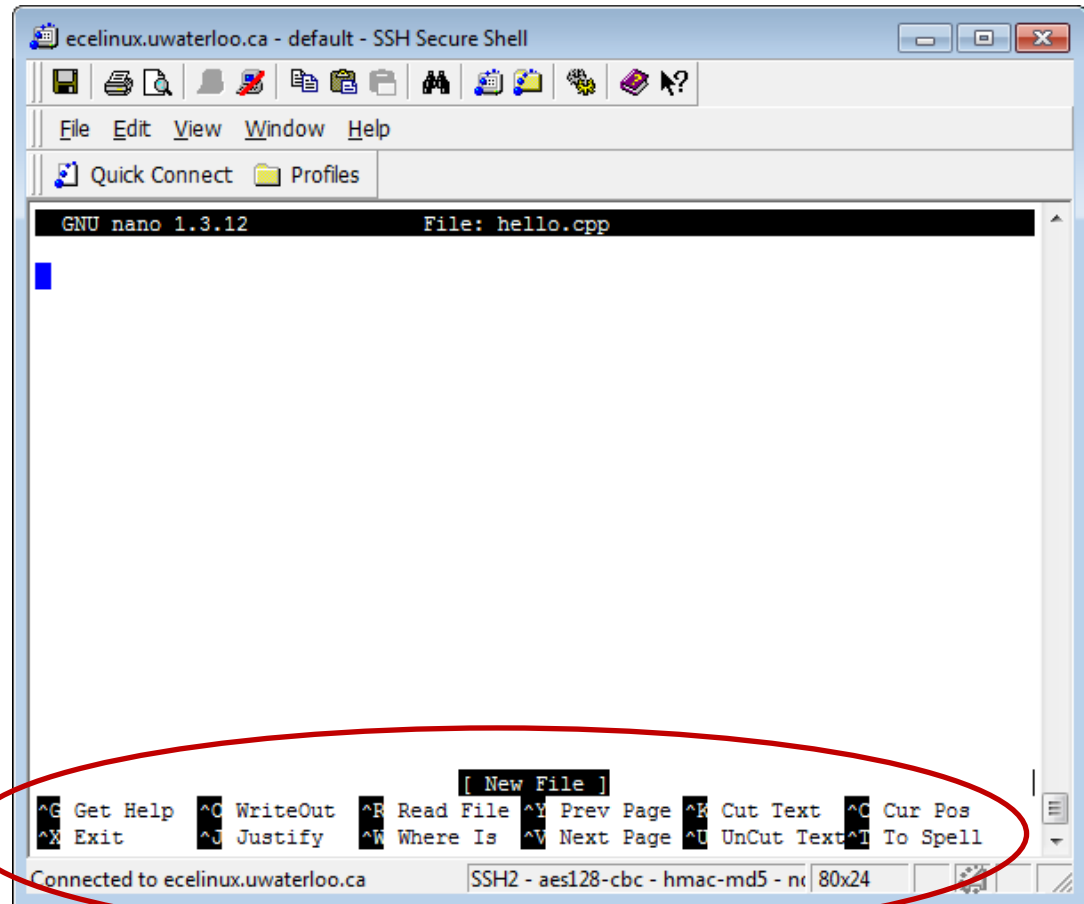
- Ctrl-g Get help

- Ctrl-o Write out
- Ctrl-r Read a file
- Ctrl-x Exit

- Ctrl-y Previous page
- Ctrl-v Next page

- Ctrl-k Cut text
- Ctrl-u Paste text

- Ctrl-c Cursor position
- Ctrl-j Justify
- Ctrl-w Search
- Ctrl-t Spell checker





Hello world!

If you are familiar with `vi` or `emacs`, you can use:

```
$ vi hello.cpp
```

```
$ emacs hello.cpp
```

- Do not use these if you are not familiar...
 - To quit `vi`, use `:q!`
 - To quit `emacs`, use `Ctrl-x Ctrl-c`



Hello world!

In your editor of choice, enter the text

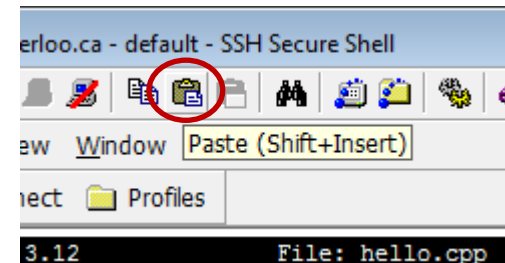
```
#include <iostream>
using namespace std;

int main() {
    cout << "Hello world!" << endl;

    return 0;
}
```

To cut-and-paste, you must first use Ctrl-c in Windows and then use *Edit*→*Paste* in the SSh window

Next, save the file (Ctrl-o) and exit (Ctrl-x)





Hello world!

At the shell, we can list the contents of the working directory:

```
$ ls  
hello.cpp
```

We now want g++ to compile the source code into an executable file or program

```
$ g++ hello.cpp  
$ ls  
a.out    hello.cpp
```

a.out **Assembler output**

Let us now execute the command

```
$ ./a.out  
Hello world!
```

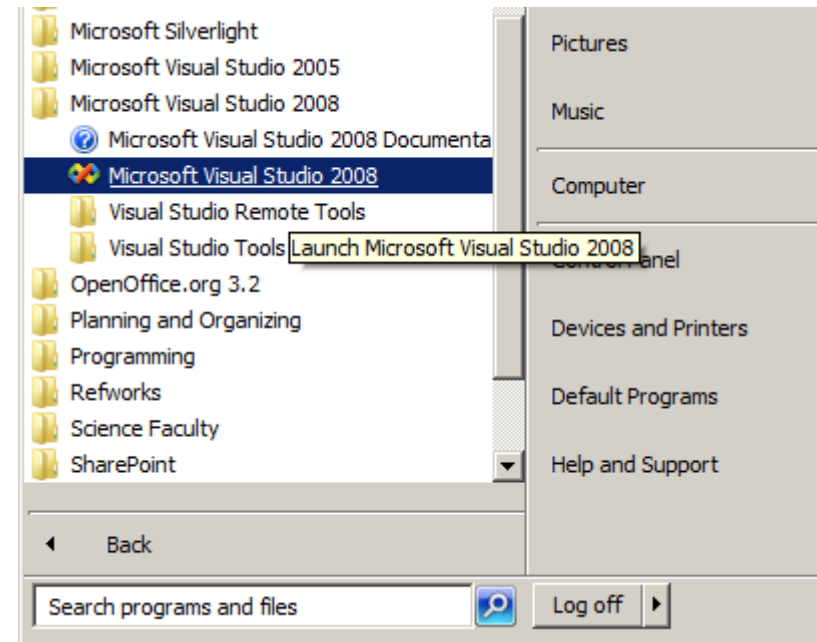



Creating a project

We will now leave Unix and go back to Windows Visual Studio

First, launch Visual Studio and create a new project

- Select a **General** project type
- Use the **Empty Project** installed template
- Use **Lab0** as the project name



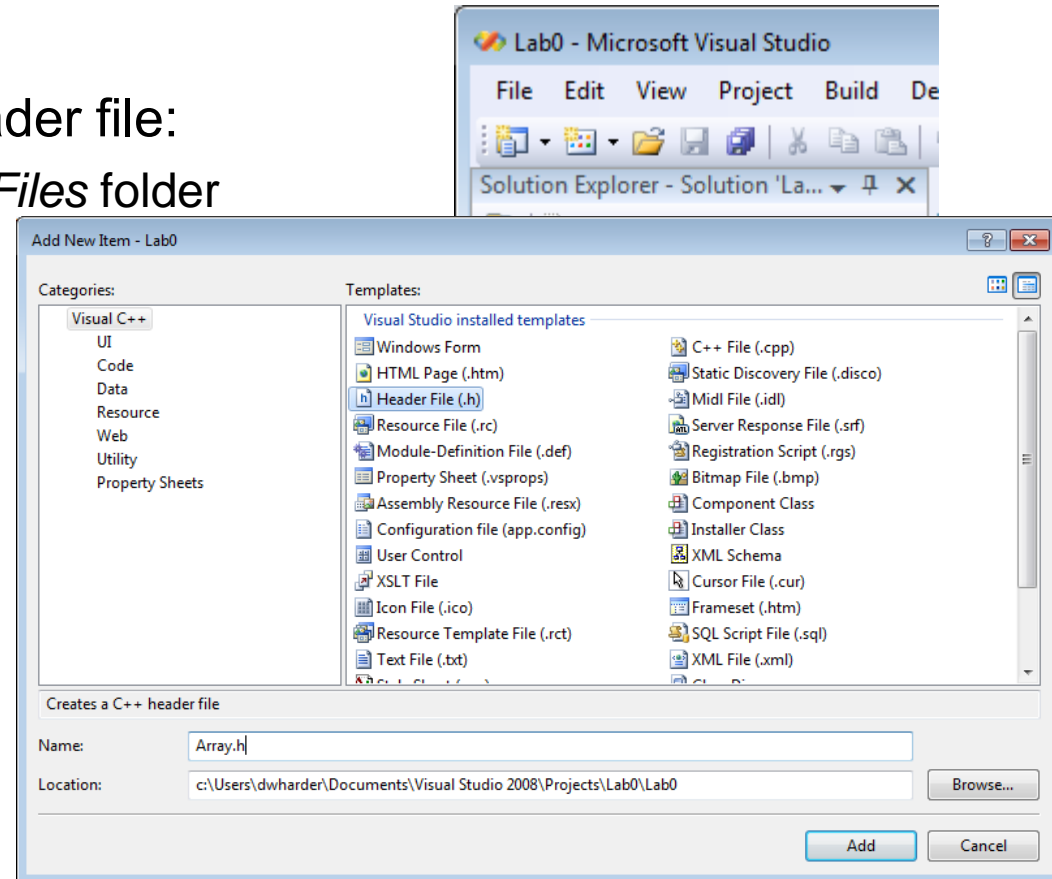
Adding a header file

This creates a project with locations for header, resource, and source files

- We'll use the first and third

First, let's create a new header file:

- Right-click on the *Header Files* folder
- Select *Add→New Item...*
 - This brings up the *Add New Item* dialog
- Select the *Header file (.h)* template, enter the name *Array.h* and select *Add*





Creating the header file

We will fill the array class:

```
#ifndef ARRAY_H
#define ARRAY_H

class Array {
private:
    int array_capacity;
    int *internal_array;
    int array_size;

public:
    Array( int );
    int size() const;
    void append( int );
};

// member function definitions go here...

#endif
```

The **capacity** is the number of things the container can hold, while the **size** is the number of objects the container is holding



Creating the header file

The member function definitions follow:

```
Array::Array( int n ):
    array_capacity( n ),
    internal_array( new int[array_capacity] ),
    array_size( 0 ) {
    // does nothing
}

int Array::size() const {
    return array_size;
}

void Array::append( int obj ) {
    // currently, entries 0, ..., array_size - 1 are occupied
    internal_array[array_size] = obj;
    ++array_size;
}
```



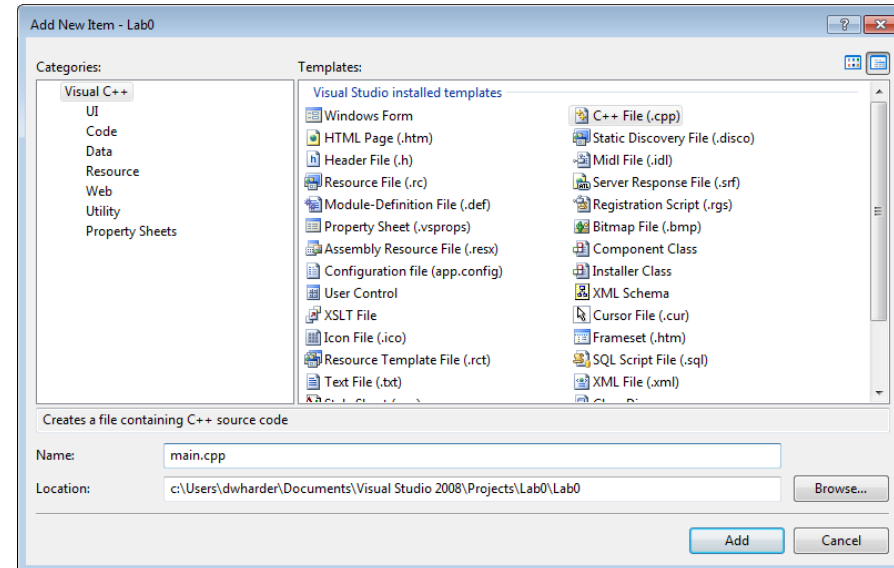
Creating an executable file

You can now save the header file

- We want to execute a program, but the header file only contains the description of the array class—it doesn't do anything

We have to create a .cpp file with a main function

- Right-click on the *Source Files* folder
- Select *Add*→*New Item...*
- Select the *C++ file (.cpp)* template, name it `main.cpp`, and select *Add*





Creating an executable file

We create an executable that adds entries into an array:

```
#include <iostream>
#include "Array.h"

int main() {
    // Create an array of size 10
    Array info( 10 );

    std::cout << "The size of the array is " << info.size() << std::endl;
    info.append( 42 );
    info.append( 91 );
    std::cout << "The size of the array is now " << info.size() << std::endl;

    return 0;
}
```



Creating an executable file

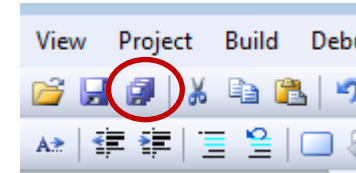
We must now:

- Save all the files:

File → *Save All* or Shift-Ctrl-s or

- Attempt to build the solution:

Build → *Build Solution* or F7 or



- Note: if the *Build Solution* icon does not appear in your toolbar, select *View* → *Toolbars* → *Build*



Creating an executable file

The compiler will now attempt to build a solution and the output appears in the lower output panel

- For a successful build, the output will be something like:

```
1>----- Build started: Project: Lab0, Configuration: Debug Win32 -----
1>Compiling...
1>main.cpp
1>Linking...
1>LINK : C:\Users\dwharder\Documents\Visual Studio 2008\Projects\Lab0\Debug\Lab0.exe not found
1>Embedding manifest...
1>Build log was saved at "file:///c:/Users/dwharder\Documents\Visual Studio 2008\Projects\Lab0\
1>Lab0 - 0 error(s), 0 warning(s)
===== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped =====
```

= Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped =



Creating an executable file

The compiler will now attempt to build a solution and the output appears in the lower output panel

- For an unsuccessful build, the compiler will try to tell you something about the error

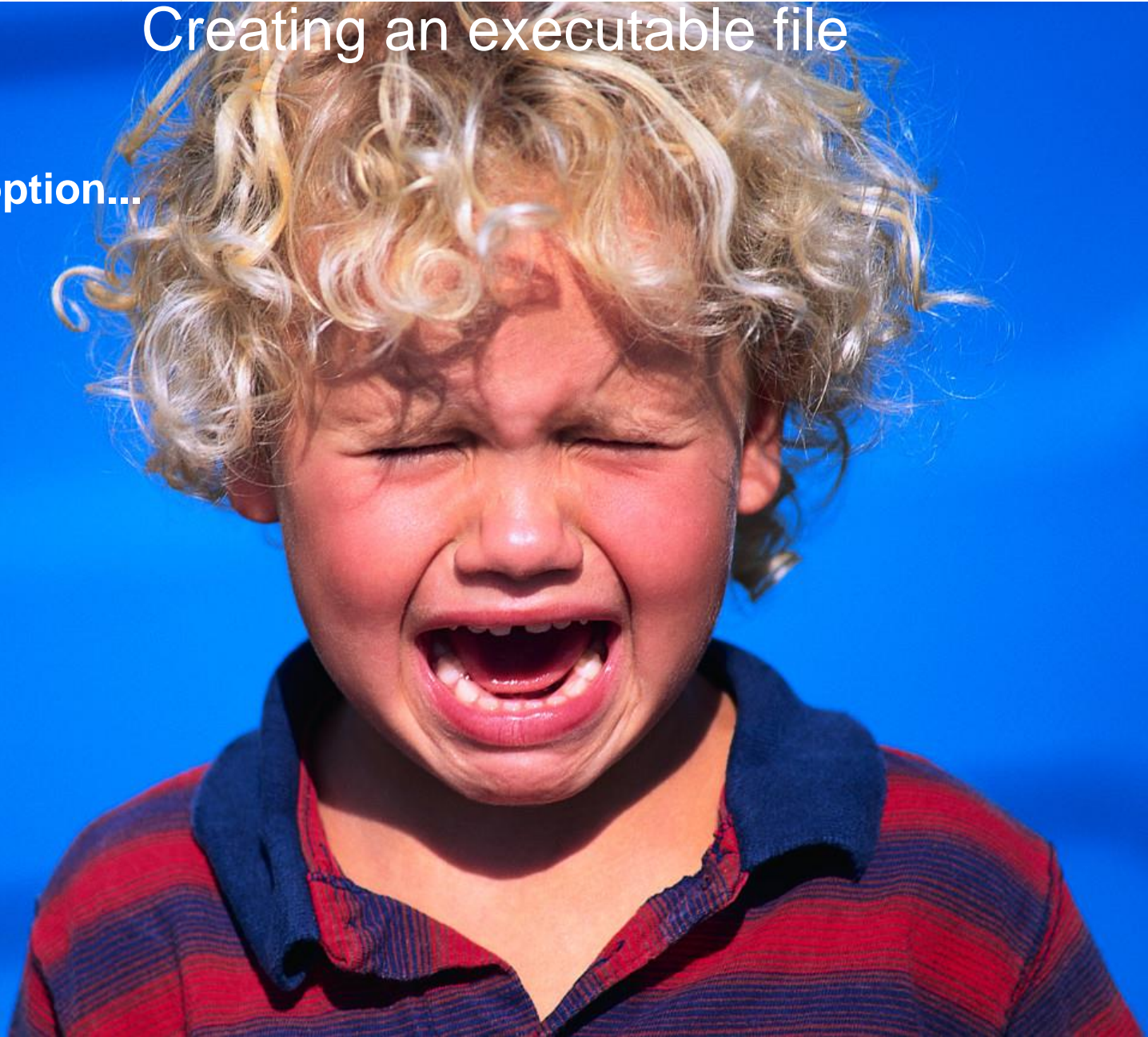
```
1>----- Build started: Project: Lab0, Configuration: Debug Win32 -----
1>Compiling...
1>main.cpp
1>c:\users\dwharder\documents\visual studio 2008\projects\lab0\lab0\main.cpp(8) : error C2065:
1>c:\users\dwharder\documents\visual studio 2008\projects\lab0\lab0\main.cpp(8) : error C2146:
1>c:\users\dwharder\documents\visual studio 2008\projects\lab0\lab0\main.cpp(8) : error C3861:
1>c:\users\dwharder\documents\visual studio 2008\projects\lab0\lab0\main.cpp(10) : error C2065
1>c:\users\dwharder\documents\visual studio 2008\projects\lab0\lab0\main.cpp(10) : error C2228 1>
type is ''unknown-type''
1>c:\users\dwharder\documents\visual studio 2008\projects\lab0\lab0\main.cpp(11) : error C2065
1>c:\users\dwharder\documents\visual studio 2008\projects\lab0\lab0\main.cpp(11) : error C2228 1>
type is ''unknown-type''
1>c:\users\dwharder\documents\visual studio 2008\projects\lab0\lab0\main.cpp(12) : error C2065
1>c:\users\dwharder\documents\visual studio 2008\projects\lab0\lab0\main.cpp(12) : error C2228
1>
type is ''unknown-type''
1>c:\users\dwharder\documents\visual studio 2008\projects\lab0\lab0\main.cpp(13) : error C2065
1>c:\users\dwharder\documents\visual studio 2008\projects\lab0\lab0\main.cpp(13) : error C2228
1>
type is ''unknown-type''
1>Build log was saved at "file:///c:/Users/dwharder/Documents/Visual Studio 2008/Projects/Lab0\
```

= Build: 0 succeeded, 1 failed, 0 up-to-date, 0 skipped =



Creating an executable file

One option...





Creating an executable file

...or you start reading the first line of the errors:

```
1>c:\users\dwhr...projects\lab0\lab0\main.cpp(8) : error C2065: 'Arry' : undeclared identifier
```

```
main.cpp(8) : error C2065: 'Arry' : undeclared identifier
```

It tells you:

- The file name `main.cpp`
- The line number `8`
- The problem: `'Arry' is not declared`

If you double-click on the error, it highlights line 8 in the file `main.cpp`



Creating an executable file

The compiler does not immediately stop when it finds an error

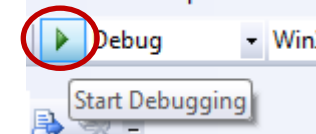
- It attempts to continue, perhaps giving further information
- In others, the first error causes subsequent errors
- In this case, fixing the first error corrects all the remaining errors



Creating an executable file

We now want to execute the program

- Select *Debug* → *Start Debugging* or F5 or
- A black window appears and disappears
 - We need to get it to pause before exiting
- At the end of your main function, add:

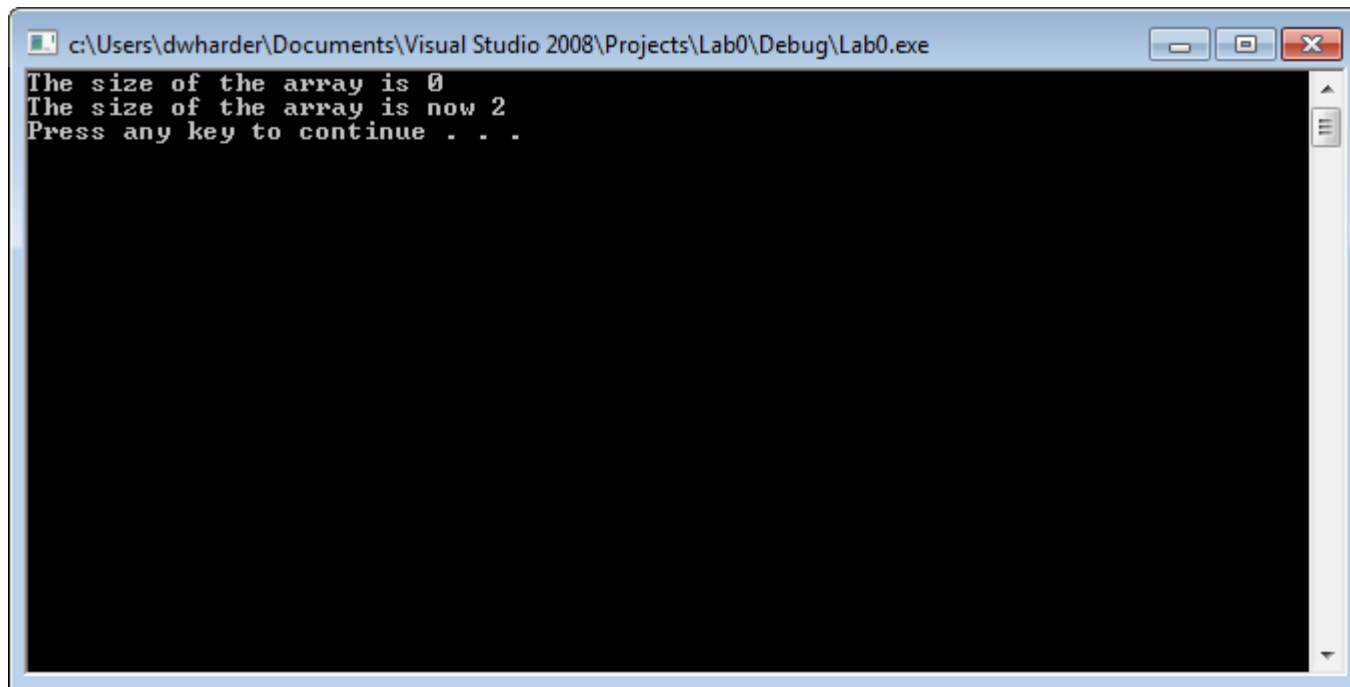


```
        system( "pause" );  
        return 0;  
    }
```



Creating an executable file

Now we see:



```
c:\Users\dwharder\Documents\Visual Studio 2008\Projects\Lab0\Debug\Lab0.exe
The size of the array is 0
The size of the array is now 2
Press any key to continue . . .
```

The `system("pause")` generates the
Press any key to continue . . .



Default values of parameters

Currently, explicitly pass the capacity of the array

```
Array info( 10 );
```

What happens if the user attempts to call

```
Array info;
```

- Currently, this causes a compilation error:

```
1>c:\users\dwharder\documents\visual studio 2008\projects\lab0\  
lab0\main.cpp(8) : error C2512:
```

```
'Array' : no appropriate default constructor available
```

This says, there is no constructor that takes no arguments



Default values of parameters

We have two options:

- Consider this a user error—the user failed to provide an array capacity
- Declare a default value for the parameter

To specify the default value, we do so in the class definition:

```
class Array {  
    private:  
        int array_capacity;  
        int *internal_array;  
        int array_size;  
  
    public:  
        Array( int = 10 );  
        int size() const;  
        void append( int );  
};
```



Correcting a possible error

What do we do if the user does:

```
Array info( 0 );  
Array info( -5 );
```

It makes sense that the capacity should be at least 1:

```
Array::Array( int n ):  
array_capacity( std::max( 1, n ) ),  
internal_array( new int[array_capacity] ),  
array_size( 0 ) {  
    // does nothing  
}
```

- We must now also include a library with `std::max`
 `#include <algorithm>`



Correcting another possible error

Consider:

```
void Array::append( int obj ) {  
    // currently, entries 0, ..., array_size - 1 are occupied  
    internal_array[array_size] = obj;  
    ++array_size;  
}
```

and the function

```
int main() {  
    Array info( 10 );  
  
    for ( int i = 0; i < 20; ++i ) {  
        info.append( i );  
    }  
  
    std::cout << "The size is " << info.size() << std::endl;  
}
```



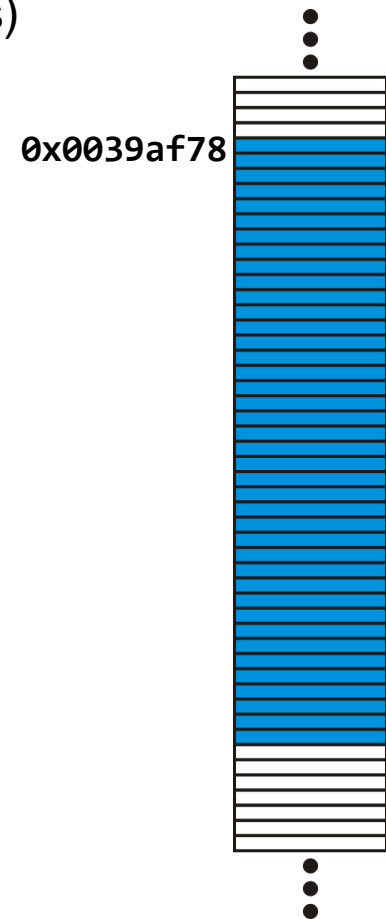
Correcting another possible error

Neither C nor C++ care if you go outside the bounds of your array

- Suppose you allocate an array of 10 ints (40 bytes)

```
int *ptr = new int[10];
```

- Suppose ptr is now assigned 0x0039af78





Correcting another possible error

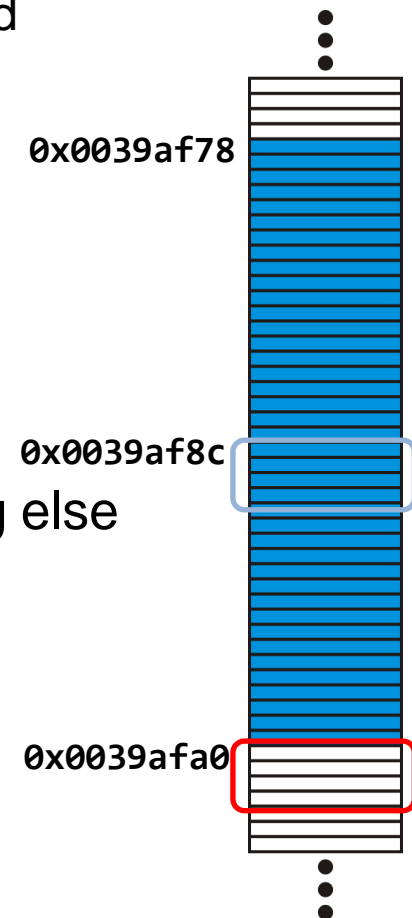
Suppose you access `ptr[5]`

- $0x0039af78 + 5 \times 4 = 0x0039af8c$ is calculated
- Whatever is there is accessed

Suppose you access `ptr[10]`

- $0x0039af78 + 10 \times 4 = 0x0039afa0$
- Whatever is there is accessed

Unfortunately, this may be allocated to something else





Correcting another possible error

What do we do if we append to an array that is full?

- We could ignore the append
- We could change the return value to `bool` which signals whether or not an append was successful
- We could *throw an exception*

We will use the second solution

- In a future function, we will use exceptions



Correcting another possible error

It is necessary to update the return value in both the class definition and the member function definition

```
class Array {
    // ...
    bool append( int );
    // ...
};

bool Array::append( int obj ) {
    if ( array_size == array_capacity ) {
        return false;
    }

    // currently, entries 0, ..., array_size - 1 are occupied
    internal_array[array_size] = obj;
    ++array_size;
    return true;
}
```



Updating the class

Let's add some more functionality:

- A function that returns the capacity
- Two Boolean-valued functions determining if the array is empty or full

To do this, we must:

1. Add three member function declarations to the class definition
2. Add three member functions definitions below the class



Updating the class

The member function declarations in the class definition could be:

```
int capacity() const;  
bool empty() const;  
bool full() const;  
void clear();
```

The first three member functions return queries based on the current state of the array object—nothing is changed

- Thus, we declare these as `const` or read-only
- If we accidentally try to change a member variable, the compiler will issue an error



Updating the class

In the class definition, we should separate the accessors from the mutators

```
class Array {  
    private:  
        int array_capacity;  
        int *internal_array;  
        int array_size;  
    public:  
        Array( int = 10 );  
  
        // Accessors  
        int size() const;  
        int capacity() const;  
        bool empty() const;  
        bool full() const;  
  
        // Mutators  
        void append( int );  
        void clear();  
};
```



`bool empty() const`

For the first Boolean-valued function, we could try:

```
bool Array::empty() const {  
    if ( array_size == 0 ) {  
        return true;  
    } else {  
        return false;  
    }  
}
```

Note, however, that `==` is a Boolean-valued operator

- It returns either `true` or `false`
- Why not just return the value generated by `==`?



`bool empty() const`

A better implementation is:

```
bool Array::empty() const {  
    return ( array_size == 0 );  
}
```

Note: the parentheses are not necessary—they give clarity
return is a statement, not a function!



bool empty() const

Another solution:

```
return ( size() == 0 );
```

- If you accidentally use `size() = 0`, the compiler will signal an error
 - You can't assign to the return value of a function!

Some programmers prefer:

```
return ( 0 == array_size );
```

- If they accidentally use `0 = array_size`, the compiler will signal an error
 - You can't assign a value to a number!
- If you accidentally use `array_size = 0`, it assigns `array_size` the value of zero and then returns 0



void clear()

The last empties the array object:

```
void clear();
```

In this case, the function is not declared const

- It will change the object

It will simply set the size member variable to zero

- It does not have to zero out the array—as new objects arrive, they will replace what currently exists in the array



void clear()

Question:

- What error do you get if you have

```
void clear();
```

in the class definition, but

```
void Array::clear() const {  
    // ...  
}
```

in the member function definition?



void clear()

Answers:

- In the first case, the member function definition does not match any of the member function declarations in the class definition:

```
'void Array::clear(void) const' : overloaded member  
function not found in 'Array'
```

- The error message in g++ is actually more helpful:

```
Array.h:36: error: prototype for 'void Array::clear() const'  
does not match any in class 'Array'  
Array.h:17: error: candidate is: void Array::clear()
```



void clear()

Question:

- What error do you get if you have **const** in both?

```
void clear() const ;
```

```
void Array::clear() const {  
    // ...  
}
```




void clear()

Answers:

- The signatures match up, but now you are trying to assign to a member variable in a read-only function:

`l-value specifies const object`

- Again, the error message in g++ is actually more helpful:

`Array.h:37: error: assignment of data-member
'Array::array_size' in read-only structure`

The term *read-only* is synonymous with the concept of `const` in C++

Incidentally, Stroustrup wanted to use the keyword `readonly` in the first version of C++



void clear()

Question:

- What happens if you call a non-read-only member function from within a read-only (const) member function?

```
bool Array::empty() const {  
    clear();  
    return ( size() == 0 );  
}
```



void clear()

Answers:

- The error message in g++ is

```
Array.h: In member function 'bool Array::empty() const':  
Array.h:178: error: passing 'const Array' as 'this'  
argument of 'void Array::clear()' discards qualifiers
```

- You cannot call a non-read-only member function from a read-only (const) member function



Updating the class

Implement these functions...



Four statistical functions

Finally, let us add four statistical functions:

```
int sum() const;  
double average() const;  
double variance() const;  
double std_dev() const;
```

These will return formulas where the sample standard deviation is the square root of the variance

$$\text{sample average} = \bar{a} = \frac{\sum_{k=1}^n a_k}{n}$$

$$\text{sample variance} = \frac{\sum_{k=1}^n (a_k - \bar{a})^2}{n-1}$$



Four statistical functions

Some provisos:

- The sum of an empty list is \emptyset
- The sample average is not defined if the size is zero
- The sample variance and standard deviations are not defined if the size is zero or one

In both cases, we must notify the user



Four statistical functions

Let's declare a class

```
class exception {  
    // empty class  
};  
  
class underflow : public exception {  
    // empty class  
};
```

We can *throw* an instance of this exception:

```
double Array::average() const {  
    if ( empty() ) {  
        throw underflow();  
    }  
  
    // find the average  
}
```



Four statistical functions

The class underflow is defined in the file `Exception.h` found at
<http://ece.uwaterloo.ca/~dwharder/aads/Projects/src/>

Copy this file into the same directory as your `Array.h` file and then include this header file into your project in Visual Studio

At the top of the `Array.h` file, include the line:

```
#include "Exception.h"
```



Four statistical functions

The use of `empty()` is also better than

```
int Array::min() const {
    if ( 0 == array_size ) {
        throw underflow();
    }

    // find the minimum entry
}
```

Someone reading this must have to try to figure out the significance of the Boolean check...



Four statistical functions

A function calling `average()` can try to *catch* the thrown exception:

```
double av;  
Array info( 10 );  
  
try {  
    av= info.average();  
} catch ( underflow excpt ) {  
    // the array 'info' is empty--use a default value  
    av = 0.0;  
}
```



Four statistical functions

In implementing these functions:

- Use `sum()` in `average()`
- Use `average()` in `variance()`
- Use `variance()` in `std_dev()`

$$\text{sample average} = \bar{a} = \frac{\sum_{k=1}^n a_k}{n}$$

$$\text{sample variance} = \frac{\sum_{k=1}^n (a_k - \bar{a})^2}{n - 1}$$



Four statistical functions

Why can't we use the following implementation?

```
double Array::average() const {  
    if ( empty() ) {  
        throw underflow();  
    }  
  
    // The average is the sum of the entries divided by the size  
    return sum()/size();  
}
```

$$\text{sample average} = \bar{a} = \frac{\sum_{k=1}^n a_k}{n}$$



Four statistical functions

Answer: the division operator sees two integers

```
double Array::average() const {
    if ( empty() ) {
        throw underflow();
    }

    // The average is the sum of the entries divided by the size
    return sum()/size();
}
```

$$\text{sample average} = \bar{a} = \frac{\sum_{k=1}^n a_k}{n}$$

It will use integer division before converting the result into a double



Four statistical functions

Solution: tell the compiler you want to convert each to a double

```
double Array::average() const {  
    if ( empty() ) {  
        throw underflow();  
    }  
  
    // The average is the sum of the entries divided by the size  
    return static_cast<double>( sum() ) /  
           static_cast<double>( size() );  
}
```

$$\text{sample average} = \bar{a} = \frac{\sum_{k=1}^n a_k}{n}$$

Now the compiler will use double-precision floating-point division



Four statistical functions

Why can't we use the following implementation?

```
double Array::variance() const {
    if ( size() <= 1 ) {
        throw underflow();
    }
```

```
double av = average();
```

```
double ssdiff = 0.0;
```

```
for ( int i = 0; i < size(); ++i ) {
    ssdiff += (internal_array[i] - av)^2;
}
```

```
return ssdiff/(size() - 1);
```

```
}
```

$$\text{sample variance} = \frac{\sum_{k=1}^n (a_k - \bar{a})^2}{n-1}$$



Four statistical functions

Answer: ^ is the binary exclusive-or (xor) operator

```
double Array::variance() const {
    if ( size() <= 1 ) {
        throw underflow();
    }
```

```
double av = average();
```

$$\text{sample variance} = \frac{\sum_{k=1}^n (a_k - \bar{a})^2}{n-1}$$

```
double ssdiff = 0.0;
```

```
for ( int i = 0; i < size(); ++i ) {
    ssdiff += (internal_array[i] - av)^2;
}
```

```
return ssdiff/(size() - 1);
```

```
}
```



Four statistical functions

Solution: explicitly perform the squaring

```
double Array::variance() const {
    if ( size() <= 1 ) {
        throw underflow();
    }
```

```
double av = average();
```

$$\text{sample variance} = \frac{\sum_{k=1}^n (a_k - \bar{a})^2}{n-1}$$

```
double ssdiff = 0.0;
```

```
for ( int i = 0; i < size(); ++i ) {
    ssdiff += (internal_array[i] - av)*(internal_array[i] - av);
}
```

```
return ssdiff/(size() - 1);
```

```
}
```



Four statistical functions

Note: the compiler knows that `ssdiff` is of type double

```
double Array::variance() const {
    if ( size() <= 1 ) {
        throw underflow();
    }
```

```
double av = average();
```

```
double ssdiff = 0.0;
```

```
for ( int i = 0; i < size(); ++i ) {
    ssdiff += (internal_array[i] - av)*(internal_array[i] - av);
}
```

```
return ssdiff/(size() - 1);
```

```
}
```

the compiler calculates `size() - 1` and converts the result to a double

$$\text{sample variance} = \frac{\sum_{k=1}^n (a_k - \bar{a})^2}{n-1}$$



Four statistical functions

In calculating the standard deviation, we must now include the `cmath` library

```
#include <cmath>
```

The square root function is within the `std` namespace

As we have already written a function to calculate the variance, we should simply call this function

- It already throws an exception if the size is less than two



Updating `main()`

Now, update your `main()` function to print the results of some of the other functions

- First, add a few more integers and then print the output of some of the other functions that we have just written



Memory deallocation

One issue we haven't addressed is memory deallocation

- The constructor made an explicit request to the operating system for some memory using the `new` operator
- The `new` operator returns the address of the first location allocated
- The operating system will keep that memory allocated until that same address is returned using the `delete` command



Memory deallocation

What happens here?

```
void f() {
    Array a( 100 );
    // do nothing else...
}

int main() {
    for ( int i = 0; i < 10000; ++i ) {
        f();
    }

    return 0;
}
```

This is said to form a *memory leak*



Memory deallocation

When a local variable goes out of scope, the memory allocated for it on the call stack gets reused

- Unfortunately, in the constructor, a call to the new operator was made
 - That memory has not been deallocated
- We need to also deallocate that additional memory when the object goes out of scope



Memory deallocation

To perform this clean-up, C++ uses a member function called the destructor:

```
class Array {  
    private:  
        // private member variables and private member functions  
  
    public:  
        Array( int = 10 );  
        ~Array();  
};
```



Memory deallocation

The implementation deletes the contents of the array:

```
// Constructor
Array::Array( int n ):
array_capacity( std::max( 1, n ) ),
internal_array( new int[array_capacity] ),
array_size( 0 ) {
    // does nothing
}

// Destructor
// - deallocate the memory for the array
Array::~~Array() {
    delete [] internal_array;
}
```



Memory deallocation

Note:

- If the memory was allocated with

```
Class *ptr = new Class( args... );
```

you must call

```
delete ptr;
```

- If the memory was allocated with

```
Class *ptr = new Class[ARRAY_SIZE];
```

you must call

```
delete [] ptr;
```




Accessing an entry

So, we're storing objects in an array—but we can't get at them

- We've added 20 things into an array; how do we access the 15th?

It might be reasonable to write some sort of 'at' function:

```
int Array::at( int n ) const {
    if ( n < 0 || n >= size() ) {
        throw out_of_range();
    }

    return internal_array[n];
}
```



Accessing an entry

Thus, our code might look something like:

```
for ( int i = 0; i < info.size(); ++i ) {  
    cout << info.at( i ) << " " << endl;  
}
```

This works...but wouldn't it be nice if you could do the following?

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

Problem: indexing only works on C++ arrays, and `info` is an instance of the `Array` class



Operator overloading

Solution: let the compiler know what to do if you call `info[15]`

```
int Array::operator[]( int n ) const {  
    return internal_array[n];  
}
```

Now, if the user ever calls `info[i]`, it will call the above function with the argument `i`

- Essentially, `info[i]` is identical to `info.operator[](i)`
- Think of `operator[]` as the name of a function that happens to be called if the user ever calls `info[i]`



Further functionality

Now we're going to go more into the gory details of C++

- Swapping two instances of the Array class

```
Array a( 3 ), b( 5 );
b.append( 52 );
// now swap 'a' and 'b' so that 'a' is of size 5
// and holds the entry 52, and
// 'b' is empty and size 3
```

- Creating a copy of an instance of the Array class

```
Array a( 5 );
a.append( 35 );
a.append( 42 );
Array b( a ); // make b a copy of 'a'
```

- Assigning an instance of the Array class to a variable already storing

```
Array a( 5 );
a.append( 35 );
Array b( 7 );
b.append( 42 );
a = b; // 'a' is now of size 7 containing 42 while the array
// of size 5 with 35 is gone
```



Swapping two instances

Suppose we want to swap the contents of two variables storing instances of our Array class

- We want to swap all of the member variables
- We can use the `std::swap` function

```
void Array::swap( Array &other ) {  
    std::swap( array_capacity, other.array_capacity );  
    std::swap( internal_array, other.internal_array );  
    std::swap( array_size,      other.array_size );  
}
```

- Note that we must use pass-by-reference: we want to swap the argument that is being passed with this object



Swapping two instances

Now we can call:

```
Array a( 3 ), b( 5 );  
a.append( 54 );  
a.append( 25 );  
a.append( 37 );  
b.append( 92 );  
b.append( 82 );  
a.swap( b );
```



Making a copy

Recall that for swap, we used pass-by-reference

- What happens if we pass an object by value?
- Answer: by default, a new instance is created and its member variables are assigned the member variables of the argument
- This may be okay for a complex number class, but does it work with the Array class?

What happens here?

```
void f( Array second ) {  
    // do something  
}
```

```
int main() {  
    Array first( 5 );  
    first.append( 13 );  
    first.append( 17 );  
    first.append( 39 );  
    first.append( 666 );  
  
    f( first );  
  
    return 0;  
}
```




Making a copy

Having executed the first five statements, this is our state:

```

int main() {
    Array first( 6 );
    first.append( 13 );
    first.append( 17 );
    first.append( 39 );
    first.append( 666 );

    f( first );

    cout << first[0];

    return 0;
}

```

first		
array_capacity		6
internal_array	0x000b3820	
array_size		4

0x000b3820	13
0x000b3824	17
0x000b3828	39
0x000b382c	666
0x000b3830	?
0x000b3834	?



Making a copy

Calling `f`, a new Array is allocated on the stack and all member variables are copied over—including the array

```
int main() {
    Array first( 6 );
    first.append( 13 );
    first.append( 17 );
    first.append( 39 );
    first.append( 666 );

    f( first );

    cout << first[0];

    return 0;
}

void f( Array second ) {
    // do something
}
```

first	array_capacity	6		
	internal_array	0x000b3820		
	array_size	4		
second	array_capacity	6	0x000b3820	13
	internal_array	0x000b3820	0x000b3824	17
			0x000b3828	39
			0x000b382c	666
			0x000b3830	?
			0x000b3834	?



Making a copy

When f returns, the destructor is called on the instance second

- This deallocates the memory of the array

```
int main() {
    Array first( 6 );
    first.append( 13 );
    first.append( 17 );
    first.append( 39 );
    first.append( 666 );
```

first		
array_capacity		6
internal_array	0x000b3820	
array_size		4

```
f( first );
```

```
cout << first[0];
```

second

array_capacity		6
internal_array	0x000b3820	
array_size		4

0x000b3820	13
0x000b3824	17
0x000b3828	39
0x000b382c	666
0x000b3830	?
0x000b3834	?

```
}
```

```
void f( Array second ) {
    // do something
}
```



Making a copy

Now `first` is referring still to memory that has been deallocated!

```
int main() {
    Array first( 6 );
    first.append( 13 );
    first.append( 17 );
    first.append( 39 );
    first.append( 666 );
```

first

array_capacity	6
internal_array	0x000b3820
array_size	4

```
f( first );
```

```
cout << first[0];
```

second

array_capacity	6
internal_array	0x000b3820
array_size	4

0x000b3820	13
0x000b3824	17
0x000b3828	39
0x000b382c	666
0x000b3830	?
0x000b3834	?

```
}
```

```
void f( Array second ) {
    // do something
}
```



Making a copy

When the values of the member variables are all simply copied over, this is said to be a *shallow copy*

first

```
array_capacity      6
internal_array      0x000b3820
array_size          4
```

second

```
array_capacity      6
internal_array      0x000b3820
array_size          4
```

```
0x000b3820      13
0x000b3824      17
0x000b3828      39
0x000b382c     666
0x000b3830      ?
0x000b3834      ?
```



Making a copy

What we require is that the copy allocates memory for a new array and copies over the values

- This is called a *deep copy*

first

```
array_capacity      6
internal_array      0x000b3820
array_size          4
```

```
0x000b3820      13
0x000b3824      17
0x000b3828      39
0x000b382c     666
0x000b3830      ?
0x000b3834      ?
```

second

```
array_capacity      6
internal_array      0x00176a48
array_size          4
```

```
0x00176a48      13
0x00176a4c      17
0x00176a50      39
0x00176a54     666
0x00176a58      ?
0x00176a5c      ?
```



Making a copy

We then need to copy over the values from the first array

- Now, when the function exits, the object **second** is destroyed and its copy of the array is deleted
- The array allocated to **first** is unaffected

first

```
array_capacity      6
internal_array      0x000b3820
array_size          4
```

```
0x000b3820  13
0x000b3824  17
0x000b3828  39
0x000b382c  666
0x000b3830  ?
0x000b3834  ?
```

second

```
array_capacity      6
internal_array      0x00176a48
array_size          4
```

```
0x00176a48  13
0x00176a4c  17
0x00176a50  39
0x00176a54  666
0x00176a58  ?
0x00176a5c  ?
```




Making a copy

The copy constructor must create a new array and copy over all the values

- The default copy constructor (a *shallow copy*) is identical to

```
Array::Array( Array const &other ):  
array_capacity( other.array_capacity ),  
internal_array( other.internal_array ),  
array_size( other.array_size ) {  
    // empty  
}
```



Making a copy

Instead, we require a *deep copy*

- We must allocate memory for an array and copy the values over

```
Array::Array( Array const &other ):  
array_capacity( other.array_capacity ),  
internal_array( new int[array_capacity] ),  
array_size( other.array_size ) {  
    // copy over the values from one array to the other  
    for ( int i = 0; i < size(); ++i ) {  
        internal_array[i] = other.internal_array[i];  
    }  
}
```



Pass by value

When an instance of a class is passed-by-value, if a copy constructor is declared, it will be called

– Example:

```
int init( Array data, ... ) {  
    // 'data' is passed-by-value and, because a  
    // copy constructor is declared, it will be  
    // initialized with the first argument of init(...)  
    // passed as the argument to the copy constructor  
  
    // Append 42 to the copy  
    data.append( 42 );  
  
    return 0;  
    // Just before the function returns,  
    //     the destructor is called on 'data'  
}
```



Pass by reference

With pass-by-reference, no copy is made

– Example:

```
int init( Array &data, ... ) {  
    // Here, the parameter 'data' refers to the original  
    // first argument passed to the init(...) function  
  
    // Append 42 to the original argument  
    data.append( 42 );  
  
    return 0;  
    // No destructor is called on 'data'  
    //     the original continues to exist  
}
```



Pass by constant reference

With pass-by-constant-reference, no copy is made but also, the function `init(...)` can only call accessors of the Array class

– Example:

```
int init( Array const &data, ... ) {  
    // Here, the parameter 'data' refers to the original  
    // first argument passed to the init(...) function  
  
    // We can call data.size(), data.sum(), data.average(), etc.,  
    // but we cannot call data.append(...) or data.clear()  
  
    return 0;  
    // No destructor is called on 'data'  
    //     the original continues to exist  
}
```



Assigning an object

What happens if we execute the following?

```
Array a( 3 );  
a.append( 35 );  
Array b( 5 );  
b.append( 42 );  
a = b;
```

We must:

- Delete the memory allocated to **a**
- Create a copy of **b**

This can be done with:

```
Array &Array::operator=( Array rhs ) {  
    swap( rhs );  
    return *this;  
}
```

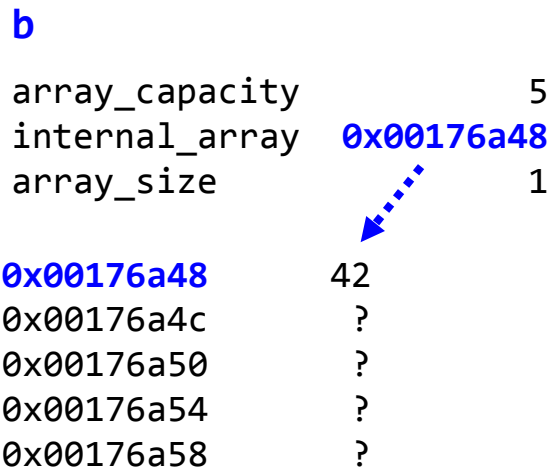
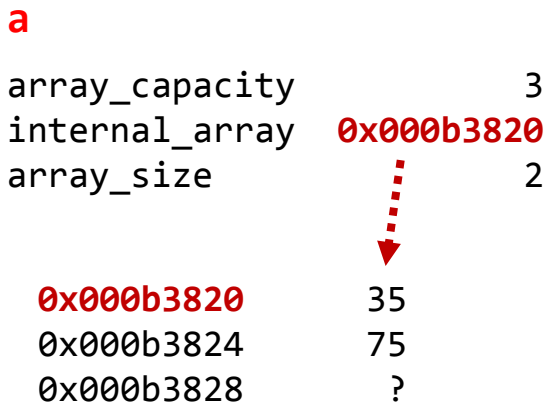


Assigning an object

What happens here is really slick:

```
Array a( 3 );
a.append( 35 );
a.append( 75 );
Array b( 5 );
b.append( 42 );
a = b;
```

```
Array &Array::operator=( Array rhs ) {
    swap( rhs );
    return *this;
}
```





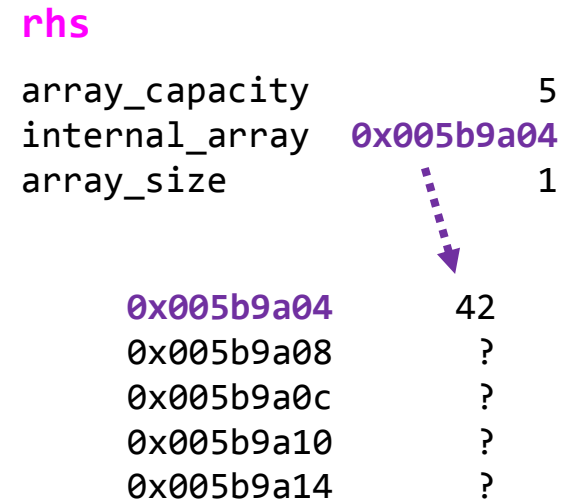
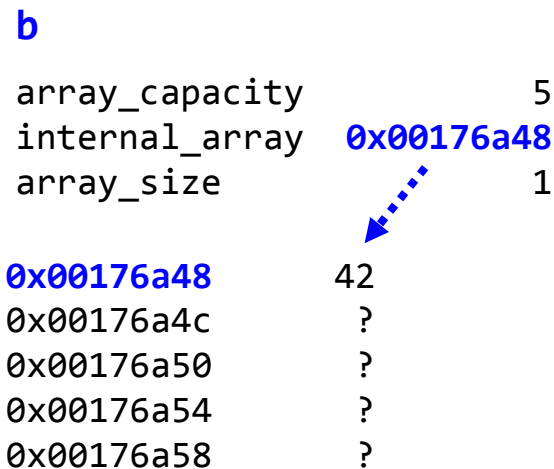
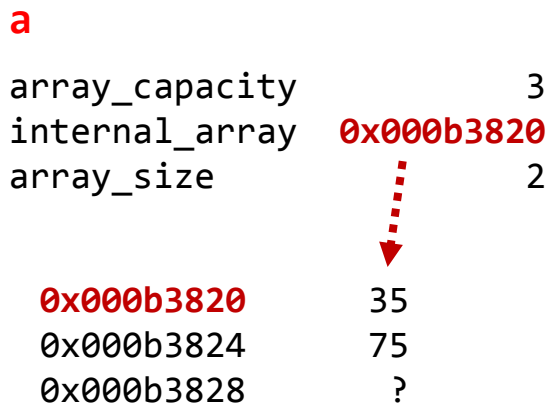
Assigning an object

When operator= is called on **a**, **b** is the argument

- The copy constructor is called and **b** is passed by value as **rhs**

```
Array a( 3 );
a.append( 35 );
a.append( 75 );
Array b( 5 );
b.append( 42 );
a = b;
```

```
Array &Array::operator=( Array rhs ) {
    swap( rhs );
    return *this;
}
```





Assigning an object

Next, we swap the member variables of **a** and **rhs**

```
Array a( 3 );
a.append( 35 );
a.append( 75 );
Array b( 5 );
b.append( 42 );
a = b;
```

```
Array &Array::operator=( Array rhs ) {
    swap( rhs );
    return *this;
}
```

a

array_capacity 5
internal_array 0x005b9a04
array_size 1

0x000b3820 35
0x000b3824 75
0x000b3828 ?

b

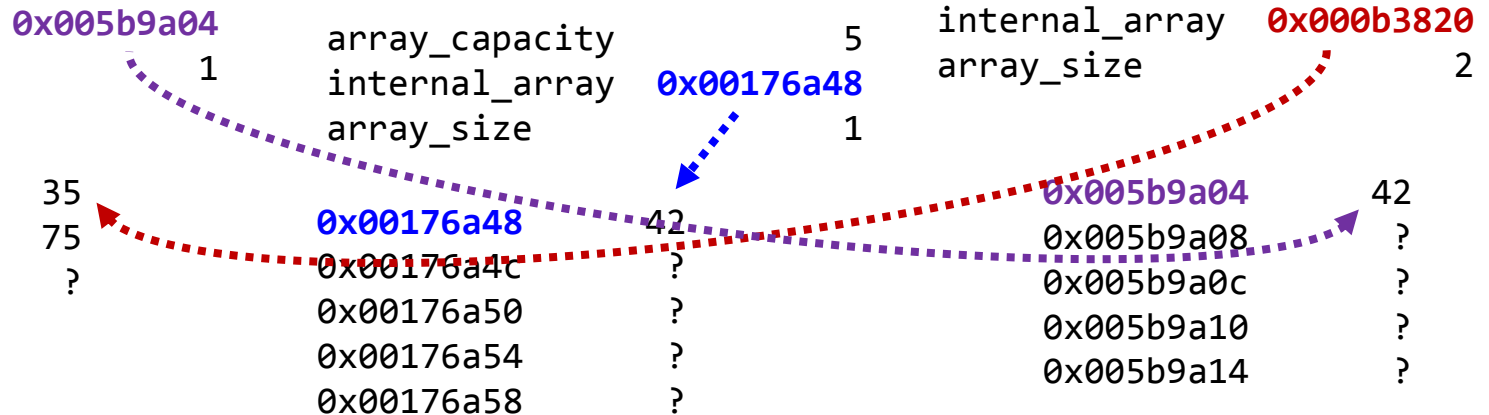
array_capacity 5
internal_array 0x00176a48
array_size 1

0x00176a48 42
0x00176a4c ?
0x00176a50 ?
0x00176a54 ?
0x00176a58 ?

rhs

array_capacity 3
internal_array 0x000b3820
array_size 2

0x005b9a04 42
0x005b9a08 ?
0x005b9a0c ?
0x005b9a10 ?
0x005b9a14 ?





Assigning an object

Now, when operator= exits, the object **rhs** is destroyed

```
Array a( 3 );
a.append( 35 );
a.append( 75 );
Array b( 5 );
b.append( 42 );
a = b;
```

```
Array &Array::operator=( Array rhs ) {
    swap( rhs );
    return *this;
}
```

a

array_capacity 5
internal_array **0x005b9a04**
array_size 1

~~0x000b3820 35
0x000b3824 75
0x000b3828 ?~~

b

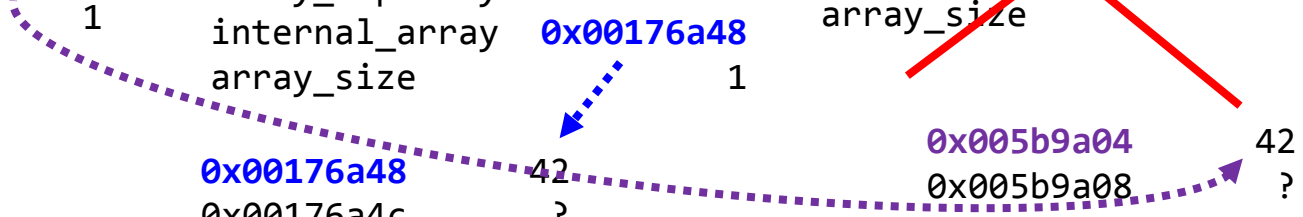
array_capacity 5
internal_array **0x00176a48**
array_size 1

0x00176a48 42
0x00176a4c ?
0x00176a50 ?
0x00176a54 ?
0x00176a58 ?

rhs

~~array_capacity 3
internal_array **0x000b3820**
array_size 2~~

~~0x005b9a04 42
0x005b9a08 ?
0x005b9a0c ?
0x005b9a10 ?
0x005b9a14 ?~~





Assigning an object

a is now a deep copy of **b** and the old memory is cleaned up

```
Array a( 3 );
a.append( 35 );
a.append( 75 );
Array b( 5 );
b.append( 42 );
a = b;
```

```
Array &Array::operator=( Array rhs ) {
    swap( rhs );
    return *this;
}
```

a

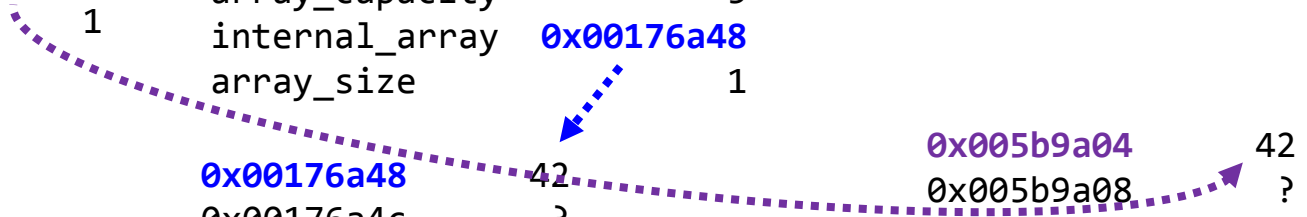
array_capacity 5
internal_array 0x005b9a04
array_size 1

b

array_capacity 5
internal_array 0x00176a48
array_size 1

0x00176a48 42
0x00176a4c ?
0x00176a50 ?
0x00176a54 ?
0x00176a58 ?

0x005b9a04 42
0x005b9a08 ?
0x005b9a0c ?
0x005b9a10 ?
0x005b9a14 ?





Printing an array

What happens if we try to print an instance of an Array class?

```
Array a( 5 );  
a.append( 35 );  
a.append( 42 );  
std::cout << a << std::endl;
```

This will result in a compile-time error—how do you print an array?

We can, however, define a function that says how to print an array

- We must declare this function to be a *friend* of our class
- This function must then describe how to print the array



Printing an array

We must declare the function to be a friend

```
class Array {  
    // private and public member functions and variables  
  
    // A friend to print the array  
    friend std::ostream &operator<<( std::ostream &, Array const & );  
};
```



Printing an array

That function must now print the object...

```
std::ostream &operator<<( std::ostream &out, Array const &para ) {
    if ( para.empty() ) {
        out << "-";
    } else {
        out << para.internal_array[0];
    }

    for ( int i = 1; i < para.size(); ++i ) {
        out << " " << para.internal_array[i];
    }

    for ( int i = para.size(); i < para.capacity(); ++i ) {
        out << " -";
    }

    return out;
}
```

Here, we print out those items that were appended and for everything else (up to the capacity), we print a hyphen



Update

- It's 5:30, you've just finished your implementation of Array
- Your boss walks up and says “I need an array of doubles, now!”

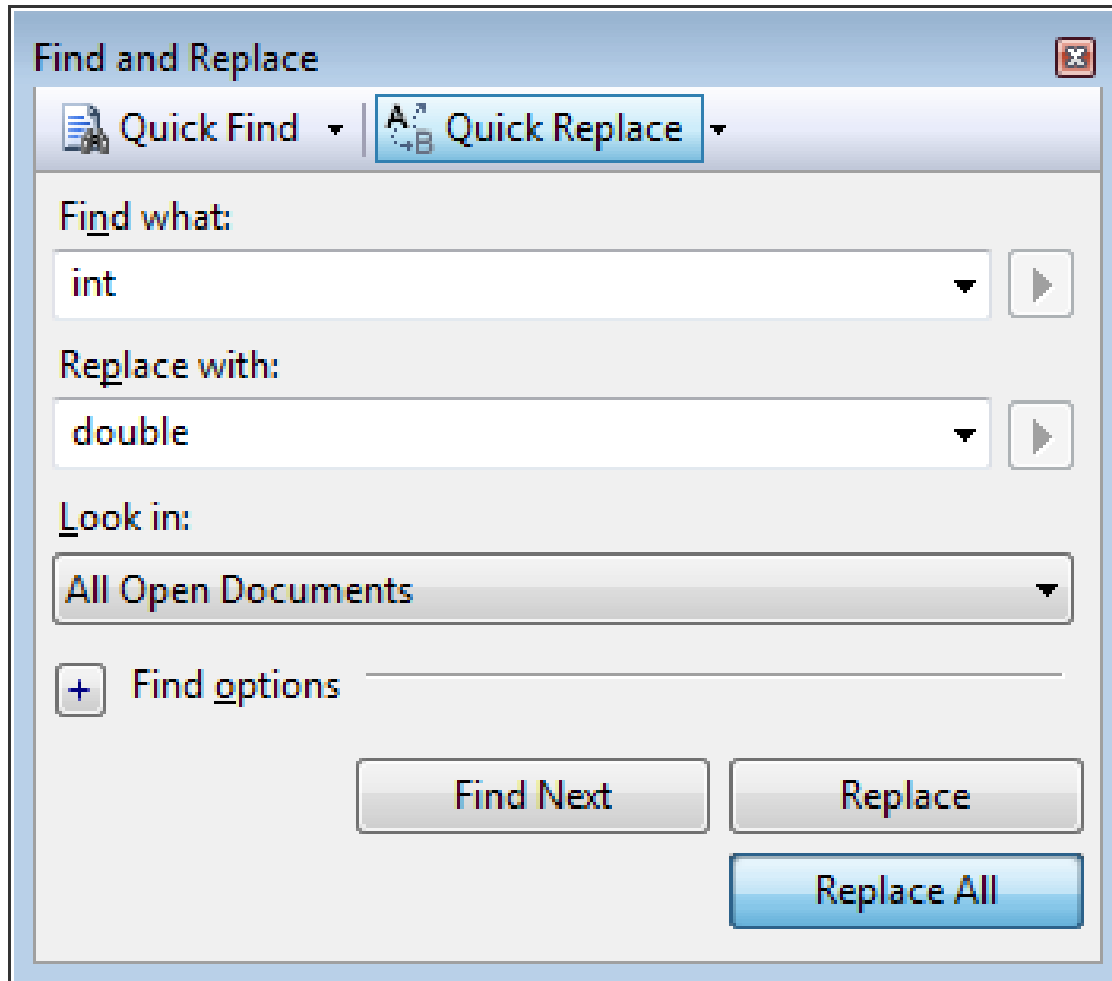


The Office (U.S. TV series), NBC Universal Television



Update

Your first thought:





Update

Then you realize this might not be a good idea...

```
// A friend to prdouble the array
friend std::ostream &operator<<( std::ostream &, Array const & );

double Array::size() const {
    return array_size;
}

double Array::variance() const {
    if ( size() <= 1 ) {
        throw underflow();
    }

    double av = average();

    double ssdiff = 0;

    for ( double i = 0; i < size(); ++i ) {
        ssdiff += (internal_array[i] - av)*(internal_array[i] - av);
    }

    return ssdiff/(size() - 1);
}
```



Templates

Instead of using an array of int, let's just define an arbitrary symbol

```

#include <algorithm>

class Array {
private:
    int array_capacity;
    Type *internal_array;
    int array_size;

public:
    Array( int = 10 );
    int size() const;
    bool append( Type );
    // etc.
};

Array::Array( int n ):
array_capacity( std::max( 1, n ) ),
internal_array( new Type[array_capacity] ),
array_size( 0 ) {
    // does nothing
}

int Array::size() const {
    return array_size;
}

bool Array::append( Type obj ) {
    if ( full() ) {
        return false;
    }

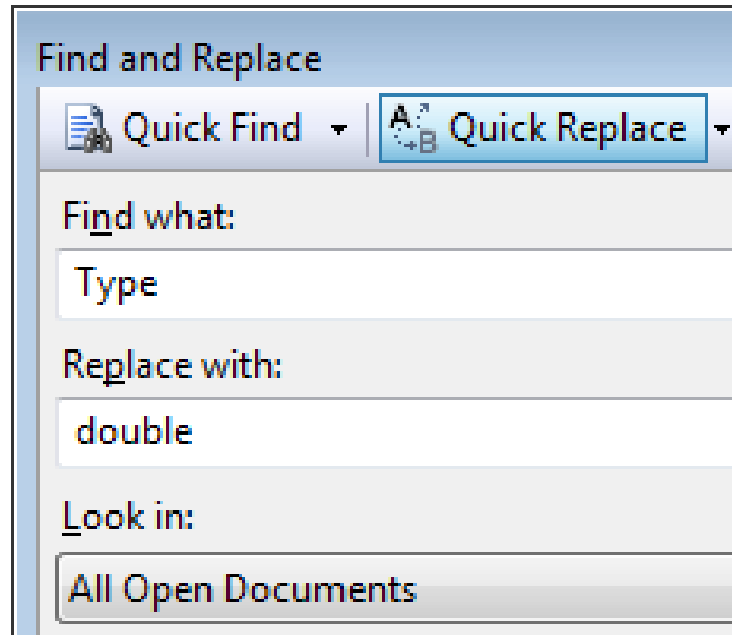
    // currently, entries 0, ..., array_size - 1
    // are occupied
    internal_array[array_size] = obj;
    ++array_size;
    return true;
}

```



Templates

Now, you can do your own find-and-replace of the type whenever you need a new Array structure



On the other hand,

if this is so obvious, why can't C++ do this for you????



Templates

Fortunately, it can, with a concept called *templates*

All you have to do is tell the compiler that Type is meant to be dictated by the programmer using the array class



Templates

Instead of using an array of int, let's just define an arbitrary type

```
#ifndef ARRAY_H
#define ARRAY_H

#include <algorithm>

template <typename Type>
class Array {
private:
    int array_capacity;
    Type *internal_array;
    int array_size;

public:
    Array( int = 10 );
    int size() const;
    bool append( Type );
    // etc.
};

template <typename Type>
Array<Type>::Array( int n ):
array_capacity( std::max( 1, n ) ),
internal_array( new Type[array_capacity] ),
array_size( 0 ) {
    // does nothing
}
```

```
template <typename Type>
int Array<Type>::size() const {
    return array_size;
}

template <typename Type>
bool Array<Type>::append( Type obj ) {
    if ( full() ) {
        return false;
    }

    // currently, entries 0, ..., array_size - 1
    // are occupied
    internal_array[array_size] = obj;
    ++array_size;
    return true;
}
```




Templates

Note that `template <typename Type>` is simply a modifier for each structure, be it a function or class

- Just like it is necessary that each function have a return type, any function or class declaration or definition must be prefixed by this statement if the structure uses templates



Templates

There is just one subtlety with friends:

```
template <typename Type>
class Array {
    // private and public member variables and member functions
    // Friends
    template <typename T>
    friend std::ostream &operator<<( std::ostream &, Array<T> const & );
};

template <typename T>
std::ostream &operator<<( std::ostream &out, Array<T> const &rhs ) {
    if ( rhs.empty() ) {
        out << "-";
    } else {
        out << rhs.internal_array[0];
    }

    for ( int i = 1; i < rhs.size(); ++i ) {
        out << " " << rhs.internal_array[i];
    }

    for ( int i = rhs.size(); i < rhs.capacity(); ++i ) {
        out << " -";
    }

    return out;
}
```



Templates

Now, in the main function, we would specify the type of the array:

```
#include <iostream>
#include "Array.h"

int main() {
    // Create an array of size 10
    Array<int> info( 10 );

    std::cout << "The size of the array is " << info.size() << std::endl;
    info.append( 42 );
    info.append( 91 );
    info.append( 35 );
    info.append( 83 );
    std::cout << "The size of the array is now " << info.size() << std::endl;
    std::cout << "The average and variance are " << info.average()
              << " and " << info.variance() << std::endl;

    return 0;
}
```



Templates

If we had an array of doubles:

```
#include <iostream>
#include "Array.h"

int main() {
    // Create an array of size 10
    Array<double> info( 10 );

    std::cout << "The size of the array is " << info.size() << std::endl;
    info.append( 42.52 );
    info.append( 91.41 );
    info.append( 35.91 );
    info.append( 83.19 );
    std::cout << "The size of the array is now " << info.size() << std::endl;
    std::cout << "The average and variance are " << info.average()
              << " and " << info.variance() << std::endl;

    return 0;
}
```



Testing

Our testing environment includes a program called a *driver* and we provide numerous testing input

The driver instantiates an instance of the data structure and the testing input indicates how to manipulate it

We provide one set of test cases—you will need to generate your own test cases, as well



Testing

The testing environment is made up of:

Feature	Sample names	Description
Testing environment	<code>Array_tester.h</code>	The testing environment, framework, and interpreter
Testing executable	<code>Array_driver.cpp</code>	Contains an executable which sets up the testing environment
Test inputs	<code>int.in.txt</code> <code>double.in.txt</code>	The input commands
Test outputs	<code>int.out.txt</code> <code>double.out.txt</code>	The expected output



Testing

Download `Array_tester.h` and `Array_driver.cpp` from
<http://ece.uwaterloo.ca/~dwharder/aads/Projects/src/>
and download `Tester.h` and `ece250.h` from
<http://ece.uwaterloo.ca/~dwharder/aads/Projects/src/>

Place them all in the same directory as `Array.h`

- Right-click on *Header Files*, select *Add→Existing Item...* and select the header (.h) files `Array_tester.h`, `Tester.h` and `ece250.h`
- Right-click on *Source Files*, select *Add→Existing Item...* and select the C++ (.cpp) file `Array_driver.cpp`



Testing

Warning!

IDEs such as Visual Studio, Eclipse, etc., only allow any one project to have exactly ONE source file that contains an

```
int main()
```

function

You must right-click on main.cpp under *Source Files* and select *Exclude From Project*

You will forget this and you will run into a frustrating error!!!!

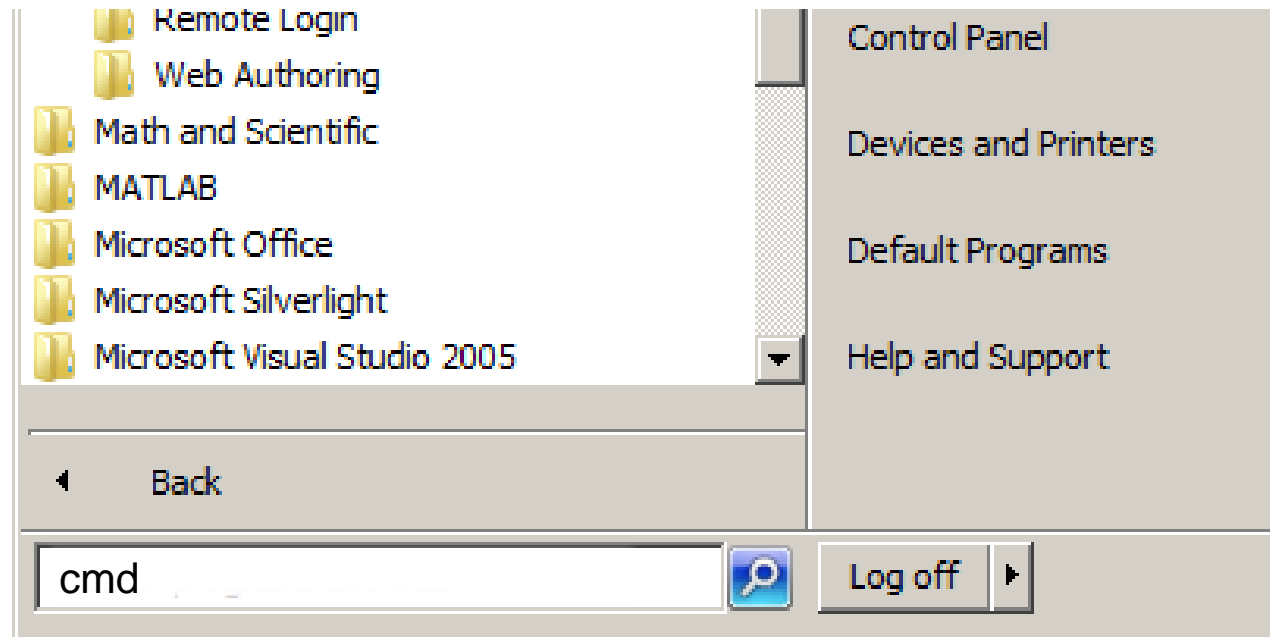


Testing

Now when build your project, it will, again create an executable, but now we have pass the file `int.in.txt` as input

Open a Command-line Window

- The easiest way is to type `cmd` and press Enter in the Start menu:





Testing

This launches a Command-line Window

```
cmd.exe C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.
C:\Users\dwharder>
```

Change directory (cd) into the directory containing your source files:
C:\Users\dwharder\Documents\Visual Studio 2008\Projects\Lab0



Testing

In the Debug sub-directory, there is an executable Lab0.exe

When you execute this function, you are met with a prompt

```
> Lab0.exe int
1 % |
```

You can now type in various commands

Note: the command-line argument `int` indicates we should be generating `Array<int>`

- Alternatively, if you want `Array<double>`, use

```
> Lab0.exe double
```



Testing

Two possible commands:

```
new
```

```
new: n
```

When you execute this function, you are met with a prompt

```
> Lab0.exe int
```

```
1 % new: 3
```

```
Okay
```

```
2 % |
```

This creates a new instance of your Array class with the value 3 passed to the constructor

- The same as calling `Array *object = new Array(3);`



Testing

Two other commands are:

```
size n
```

```
capacity n
```

You can check that

```
2 % size 0
```

```
Okay
```

```
3 % capacity 3
```

```
Okay
```

```
4 % |
```

These test:

```
object->size() == 0
```

```
object->capacity() == 3
```



Testing

Another command is
`append n b`

You can check that

```
4 % append 42 1
```

```
Okay
```

```
5 % append 37 1
```

```
Okay
```

```
6 % append 51 1
```

```
Okay
```

```
7 % append 68 0
```

```
Okay
```

```
8 % |
```

These test:

```
object->append( 42 ) == 1
```

```
object->append( 37 ) == 1
```

```
object->append( 51 ) == 1
```

```
object->append( 68 ) == 0
```

Recall that 1 is true and 0 is false...



Testing

Other examples:

```
at i n
```

```
at! i
```

You can check that

```
8 % at 1 37
```

```
Okay
```

```
9 % at 2 51
```

```
Okay
```

```
10 % at! 3
```

```
Okay
```

```
11 % |
```

These test:

```
(*object)[1] == 37
```

```
(*object)[2] == 51
```

and that `(*object)[3]` throws an exception



Testing

We can end with

```
11 % delete
```

```
Okay
```

```
12 % details
```

```
SUMMARY OF MEMORY ALLOCATION:
```

```
Memory allocated: 64
```

```
Memory deallocated: 64
```

```
INDIVIDUAL REPORT OF MEMORY ALLOCATION:
```

Address	Using	Deleted	Bytes
0x49860a0	new	Y	24
0x49860d0	new[]	Y	12

```
13 % exit
```

```
Finishing Test Run
```

```
> |
```

The first calls

```
delete object;
```



Testing

A list of all possible commands is always found at the top of the `Array_tester.h` file in the comments

```

* new          new Array()      create a new array with default capacity
* new: n       new Array( n )   create a new array with capacity n
* size n      size             the size equals n
* capacity n  capacity         the capacity equals n
* empty b     empty            empty() returns the Boolean value b
* full b      full             full() returns the Boolean value b
* sum n       sum              the sum of the entries is n
* prod n      prod             the product of the entries is n
* min n       min              the minimum entry is n
* min!        min              an underflow exception is thrown
* max n       max              the maximum entry is n
* max!        max              an underflow exception is thrown
* average d   average          the average of the entries is d
* average!    average          an underflow exception is thrown
* variance d  variance         the variance of the entries is d
* variance!   variance        an underflow exception is thrown
* std_dev d   std_dev          the standard deviation of the entries is d
* std_dev!    std_dev          an underflow exception is thrown
* at i m      operator[]       object[i] returns m
* at! i       operator[]       object[i] throws an out_of_range
* append n b  append           attempting to append n returns the Boolean value b
* clear       clear            empties the array--always succeeds as a test
* cout        cout << Array    print the Array (for testing only)
* assign      operator =       assign this Array to a new Array object
* summary     summary          prints the amount of memory allocated
*             *                minus the memory deallocated
* details     details          prints a detailed description of which
*             *                memory was allocated with details
* !!          !!               use the previous command, e.g. 5 append 3
*             *                6 !! 7           // same as append 7
* !n         !n               use the command used in line n 7 append 6

```



Testing

If you ask for something incorrect, you get an error message:

```
1 % new
```

```
Okay
```

```
2 % append 52 1
```

```
Okay
```

```
3 % at 0 53
```

```
Failure in instance[0]: expecting the value '53' but got '52'
```

Suppose you forgot to implement an error condition:

```
4 % at! 1
```

```
Failure in instance[1]: expecting to catch an exception but nothing  
was raised.
```



Testing

Now, you could enter these commands over and over again, or you could save these instructions in a text file and then redirect them as input to the interpreter

- Our tests are provided to you as text files
- Move the provided `.txt` files to the project Debug directory

At the command line, type:

```
> Lab0.exe int < int.in.txt  
> type int.out.txt  
> Lab0.exe double < double.in.txt  
> type double.out.txt
```

Your output should be identical to the corresponding `*.out.txt`



Testing

Note: If your code does not work, you are welcome to examine the sample solution at

<http://ece.uwaterloo.ca/~dwharder/aads/Projects/0/src/Array.h>

The solution contains what would be reasonably considered to be the minimum amount of acceptable comments

Note: If you do not make a serious attempt to write your own implementation before you check the solution, you will have wasted your time... ☹️

- It would be much better for you and your friends if you asked your friends for help...

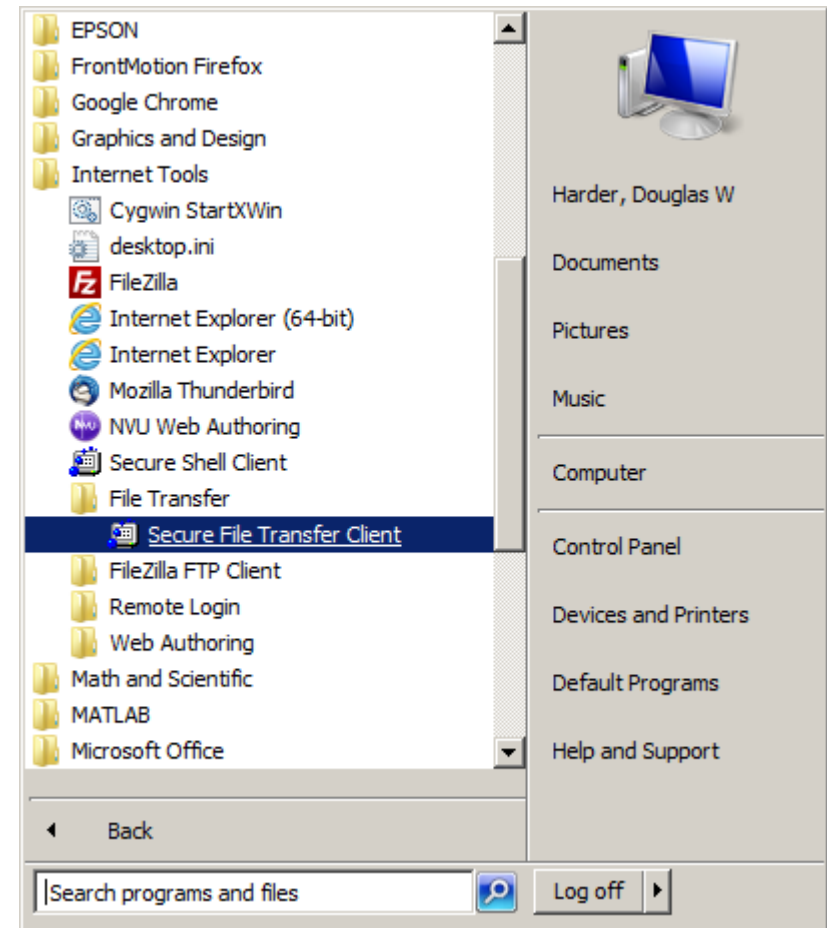


Going back to ecelinux

So, now we must try compiling this in Unix

- First, we must move the files over

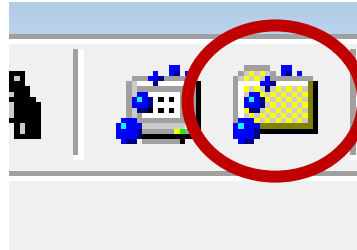
To copy files between operating systems, we must use the file-transfer protocol (ftp)





Going back to ecelinux

The easiest is if you still have the SSh window open, just select the ftp icon in the toolbar:



This opens an SSh Secure File Transfer window

- Don't worry about the left-hand *local* panel
- Navigate to ece250/lab0 in the right-hand *remote* panel

Now you can open a Windows Explorer window and drag-and-drop all of the .h, .cpp, and .txt files into the right-hand remote panel



Going back to ecelinux

In your SSh shell, you can now see these files:

```
$ pwd
```

```
/home/dwharder/ece250/lab0
```

```
$ ls
```

```
a.out          Array.h        Array_driver.cpp
```

```
Array_tester.h double.in.txt  double.out.txt
```

```
ece250.h       hello.cpp     int.in.txt
```

```
int.out.txt    Tester.h     main.cpp
```

```
$ g++ Array_driver.cpp
```

g++ will overwrite the a.out file



Going back to ecelinux

We can test the files, too:

```
$ ./a.out int < int.in.txt
```

```
output...
```

```
$ cat int.out.txt
```

```
it should appear the same...
```

If you want to be certain they're the same, diff should have no output:

```
$ ./a.out int < int.in.txt > output.txt
```

```
$ diff int.out.txt output.txt
```

```
$
```



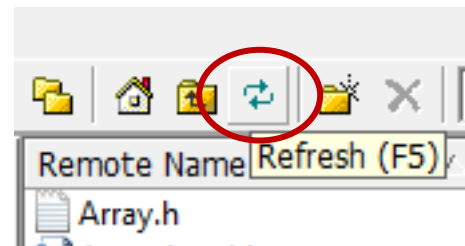
Project submission

You are now ready to create your submission:

```
$ tar -cvf uwuserid_p0.tar Array.h
$ ls
... uwuserid_p0.tar ...
$ gzip uwuserid_p0.tar
$ ls
... uwuserid_p0.tar.gz ...
```

You can now copy the files back to Windows by dragging and dropping them from the right-hand panel in ftp to your Windows file system

- You might have to refresh to see the newly created file





Recursion

Recursion is defined as when a the value of a function is defined in terms of other values of the function

- The value of the function will be known for at least one point

We will look at the factorial function and the Fibonacci numbers



Factorial function

There are two definitions of the factorial function:

Explicit:
$$n! = \prod_{k=1}^n k$$

Recursive:
$$n! = \begin{cases} 1 & n = 0 \\ n \cdot (n-1)! & n > 0 \end{cases}$$



Factorial function

In this case, the two implementations are:

```
double factorial( int n ) {
    double result = 1;

    for ( double i = 2; i <= n; ++i ) {
        result *= i;
    }

    return result;
}

double factorial_r( int n ) {
    if ( n <= 1 ) {
        return 1;
    } else {
        return n*factorial_r( n - 1 );
    }
}
```




Factorial function

Why double and not int or long?

$$2^{31} - 1 = 2147483647$$

$$13! = 6227020800$$

$$2^{63} - 1 = 9223372036854775807$$

$$21! = 51090942171709440000$$



Fibonacci numbers

There are two definitions of the Fibonacci numbers:

Explicit:
$$\phi \stackrel{\text{def}}{=} \frac{\sqrt{5} + 1}{2}$$

$$F(n) = \frac{\phi^n}{\sqrt{5}} - \frac{(1-\phi)^n}{\sqrt{5}}$$

Recursive:
$$F(n) = \begin{cases} 1 & n = 1, 2 \\ F(n-1) + F(n-2) & n > 2 \end{cases}$$



Fibonacci numbers

In this case, the two implementations are:

```
double fibonacci( int n ) {
    double phi = (std::sqrt(5.0) + 1.0)/2.0;

    double result = (
        std::pow( phi, n ) - std::pow( 1.0 - phi, n )
    )/std::sqrt( 5.0 );

    std::floor( result + 0.5 );
}
```

$$F(n) = \frac{\phi^n}{\sqrt{5}} - \frac{(1-\phi)^n}{\sqrt{5}}$$

```
double fibonacci_r( int n ) {
    if ( n <= 2 ) {
        return 1;
    } else {
        return fibonacci_r( n - 1 ) + fibonacci_r( n - 2 );
    }
}
```



Fibonacci numbers

Why double and not int or long?

$$2^{31} - 1 = 2147483647$$

$$F(47) = 2971215073$$

$$2^{63} - 1 = 9223372036854775807$$

$$F(93) = 12200160415121876738$$



Trying it out...

In this case, the two implementations are:

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

int main() {
    // print 17 digits of precision in the output
    cout.precision( 17 );

    for ( int i = 0; i <= 100; i += 1 ) {
        cout << "Explicit: " << i << "! = " << factorial( i ) << endl;
        cout << "Recursive: " << i << "! = " << factorial_r( i ) << endl;
        cout << "Explicit: F(" << i << ") = " << fibonacci( i ) << endl;
        cout << "Recursive: F(" << i << ") = " << fibonacci_r( i ) << endl;
    }

    return 0;
}
```



Trying it out...

How long does this take for you to run?



Usage Notes

- These slides are made publicly available on the web for anyone to use
- If you choose to use them, or a part thereof, for a course at another institution, I ask only three things:
 - that you inform me that you are using the slides,
 - that you acknowledge my work, and
 - that you alert me of any mistakes which I made or changes which you make, and allow me the option of incorporating such changes (with an acknowledgment) in my set of slides

Sincerely,

Douglas Wilhelm Harder, MMath

dwharder@alumni.uwaterloo.ca