Pointer & Reference & I/O

Computer Programming for Engineers (DASF003-41)

Instructor:

Sungjae Hwang (jason.sungjae.hwang@gmail.com)

This week!

- Textbook : Absolute C++ 6th edition (Walter Savitch)

- Review Pointer & Reference (10.1)
- Dynamic Memory Allocation (10.2)
- Console I/O (1.3)
- File I/O (12.1)

POINTERS AND REFERENCE

Review on Pointers

- **Pointer Definition**
  - Memory address of a variable
- **Pointers are "typed"**
  - We can store pointer in variables
  - Not int, double etc.
    - A pointer to int, double etc.

Review on Pointers

- C-style pointers are still intensively used:

```
int robert, william;
int *bob, *jason;
```

- robert, william are ordinary int variables
- bob and jason store pointers to int variables
- Pointers are still used for call-by-reference in functions.

Review on Pointers

- C-style pointers are still intensively used:

```
int robert, william;
int *bob = &robert, *jason = &william;
```

- &: “address of” operator
  - Obtain the address of a variable

Review on Pointers

- C-style pointers are still intensively used:

```
int robert, william;
int *bob = &robert, *jason = &william;

cout << *bob;   //same output
cout << robert; //same output
```

- * : “dereference” operator
  - Dereference the pointer variable
  - Obtain data that pointer variable points to

Review on Pointers

Important: why are pointers so confusing?

- Reason: usage of '*' is different between declaration and dereference.
- Declaration:

```
int *p1, *p2; // * is used for declaration
```

- Dereference

```
int  q = *p1; // * is used for dereference
```

Good practice to distinguish them

- So, use int* instead of int * (i.e., remove space in declaration)

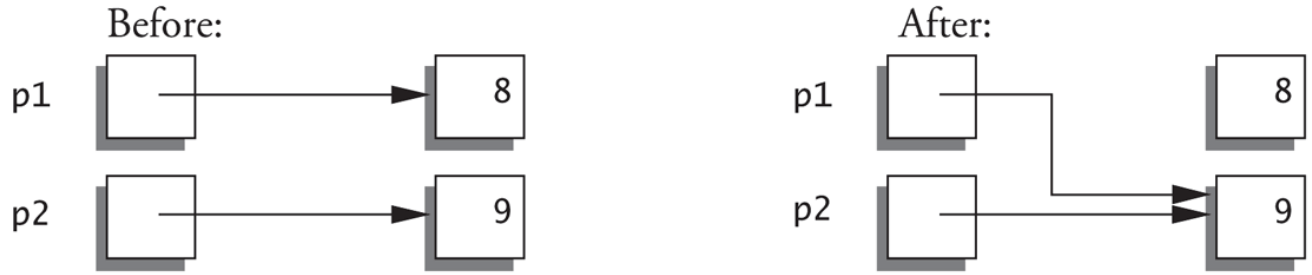
```
int* p1; // int* considered a type, avoiding confusion
int* p2;
```

Review on Pointers

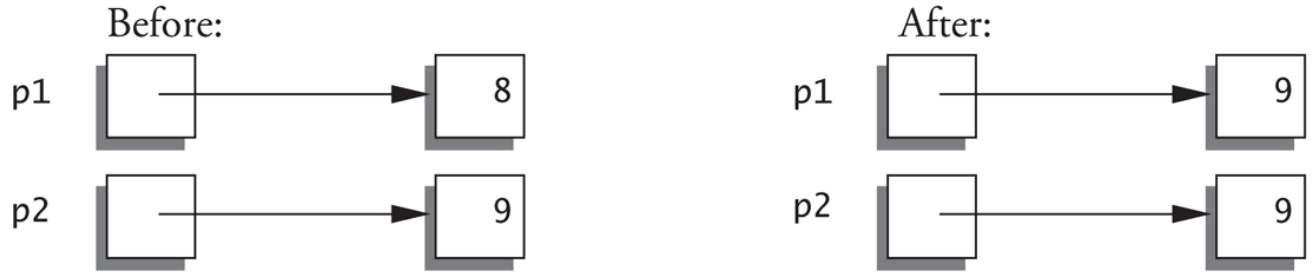
- Pointer assignment

Display 10.1 Uses of the Assignment Operator with Pointer Variables

```
p1 = p2;
```




```
*p1 = *p2;
```



Pointer


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

int main(){
  int a = 3;
  int b = 5;
  int *poi_1, *poi_2, *poi_3;

  poi_1 = &a;
  poi_2 = &b;

  poi_3 = poi_1;

  cout << "*poi_1: " << *poi_1 << endl;
  cout << "*poi_3: " << *poi_3 << endl;
  cout << "*poi_2: " << *poi_2 << endl;

  cout << "poi_1: " << poi_1 << endl;
  cout << "poi_3: " << poi_3 << endl;
  cout << "poi_2: " << poi_2 << endl;

  cout << "&poi_1: " << &poi_1 << endl;
  cout << "&poi_3: " << &poi_3 << endl;
  cout << "&poi_2: " << &poi_2 << endl;

  *poi_2 = *poi_1;
  cout << "**********************" << endl;

  cout << "*poi_1: " << *poi_1 << endl;
  cout << "*poi_3: " << *poi_3 << endl;
  cout << "*poi_2: " << *poi_2 << endl;

  cout << "poi_1: " << poi_1 << endl;
  cout << "poi_3: " << poi_3 << endl;
  cout << "poi_2: " << poi_2 << endl;

  cout << "&poi_1: " << &poi_1 << endl;
  cout << "&poi_3: " << &poi_3 << endl;
  cout << "&poi_2: " << &poi_2 << endl;

  return 0;
}
```

Review on Pointers

Can perform arithmetic on pointers

- "Address" arithmetic

Example

```
int arr[5] = {1,2,3,4,5};
(arr) contains address of arr[0]
(arr+1) evaluates to address of arr[1]
(arr+2) evaluates to address of arr[2]
```

- Arithmetic results are different depending on the basic type

Pointer arithmetic example


```
 1 #include <iostream>
 2 using namespace std;
 3
 4 int main(){
 5   int arr[5] = {1,2,3,4,5};
 6   cout << "arr: " << arr << endl;
 7   cout << "arr+1: " << arr+1 << endl;
 8   cout << "arr+2: " << arr+2 << endl;
 9   return 0;
10 }
```

nullptr (C++11)

Ambiguity of NULL in C

- no distinction between integer 0 and null pointer

```
void func( int* p );
void func( int i );
```

- Which func is invoked given `func(NULL)`? Both are equally valid since NULL is merely 0.

C++11 resolved this problem by introducing nullptr

```
int* p_int = nullptr;
printf( "%s\n", typeid(nullptr).name() );
```

```
>> echo 'Dn' | c++filt -t
std::nullptr_t
```

- the type of `nullptr` is `std::nullptr_t`
- `std` is a namespace for standard, which will be explained later

Reference

Reference defined:

- conceptually similar to pointer, but is simpler
  - We can use it, as if it has the same type as the source has.
- Specified by ampersand (&) after type (e.x int& a;)
- Name of a storage location or alias to a variable
- Must be a valid reference! no null/invalid reference exist.
  - Pointer can have a null pointer, but is not for references.

Reference

Example of a standalone reference

```
int robert, william;
int& bob = robert, &bill=william;
```

- Changes made to bob will affect robert
- Multiple reference declaration is similar to those of pointers.
  - e.g., int &r1, &r2;

Reference vs pointer

- Cannot store NULL
- Cannot be re-assigned
- Initialization and declaration must come together
- Share the same memory location

Reference


```
 1 #include <iostream>
 2 using namespace std;
 3
 4 int main(){
 5
 6   int a = 3, b = 5;
 7   int* po1 = &a;
 8   int* po2 = po1;
 9
10   int& re1 = a;
11   int& re2 = re1;
12
13   cout << "po1: " << &po1 << endl;
14   cout << "po2: " << &po2 << endl;
15   cout << "re1: " << &re1 << endl;
16   cout << "re2: " << &re2 << endl;
```

```
19 #if 0
20   //Error 1: cannot be null
21   int& d = NULL;
22 #endif
23
24
25 #if 0
26   //Error 2: initialize when declaring it
27   int& d;
28   c = a;
29 #endif
30
31   return 0;
332 }
```

Array Pointer (Revisit)

Array Pointer

```
int arr[5] = {1,2,3,4,5};
int *ptr = arr; //ptr points to 0th element of arr array
int(*ptr2)[5] = &arr //ptr2 points to whole array
```

- data type (*variable name) [array size]

Array Reference

- data type (&variable name) [array size]

```
int arr[5] = {1,2,3,4,5};
int(&ptr3)[5] = arr //ptr3 is array reference
```

Array & Reference Pointer


```
 2 #include <iostream>
 3 using namespace std;
 4
 5 int main()
 6 {
 7 int arr[5] = { 1, 2, 3, 4, 5 };
 8 int *ptr = arr;
 9 int (*ptr2)[5] = &arr;
10 int (&ptr3)[5] = arr;
11
12 cout << "sizeof(ptr) = "<< sizeof(ptr) << ",
sizeof(*ptr) = " << sizeof(*ptr) << endl;
13 cout << "sizeof(ptr2) = "<< sizeof(ptr2) << ",
sizeof(*ptr2) = " << sizeof(*ptr2) << endl;
14
15 cout <<"\n"<< ptr << endl;
16 cout << ptr[0] << endl;
17 cout << ptr[1] << endl;
18
19 cout << "++++++++++++" << endl;
20 cout << (*ptr2)[0] << endl;
21 cout << (*ptr2)[1] << endl;
22
23 cout << "++++++++++++" << endl;
24 cout << ptr3[0] << endl;
25 cout << ptr3[1] << endl;
26 return 0;
27 }
```

Pointer pitfall

- Define pointer type

```
int* a, b;
```

- Any Problem here?

Pointer pitfall #1

Define pointer type

```
int* a, b;
```

- Only a is pointer variable

typedef

```
typedef int* IntPtr;
IntPtr a, b;
int *a, *b;
```

- Can use typedef to define an alias for any type

Pointer pitfall #2

Define pointer type

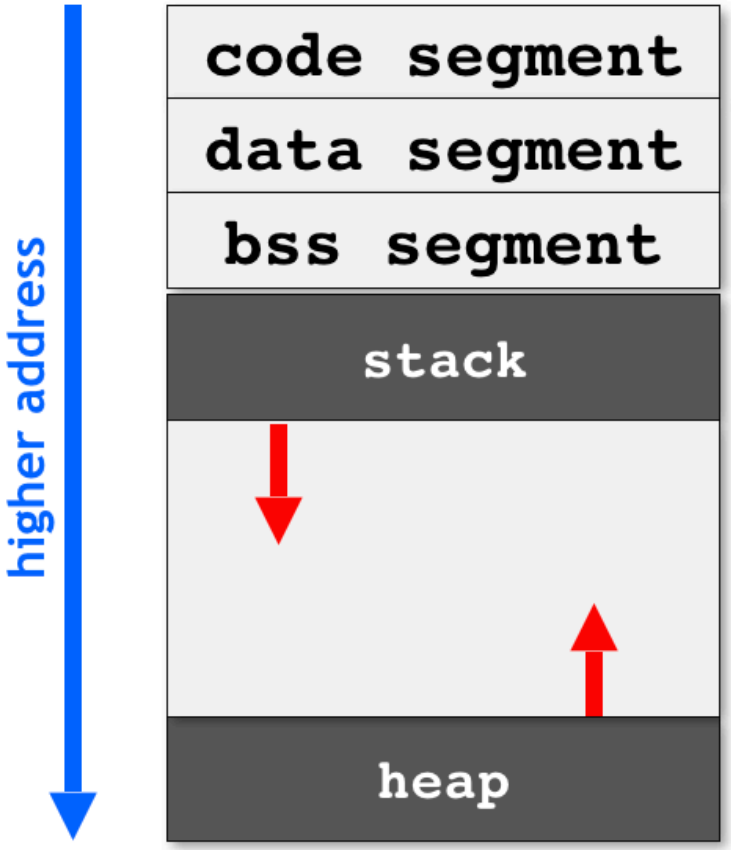
```
int a[10] = {1,2,3,4,5,6,7,8,9,0};
int* b;

b = a; //legal
a = b; //illegal
```

- Array pointer is **CONSTANT** pointer!

DYNAMIC MEMORY ALLOCATION

Review on Pointers: Memory Structure

- A computer program in memory consists of segments



- program instructions
- global/static variables with initialization
- global/static variables (uninitialized)
- local variables in functions
  grows downwards

- dynamically-allocated memory
  grows upwards

malloc and free (in C)

- malloc()/free() functions can be used to allocate/deallocate memory.
  - e.g., dynamically allocate/release a single integer

```
int* ptr = (int*) malloc( sizeof(int)*1 );
if(ptr) free( ptr );
ptr = NULL;
```

  - e.g., dynamically allocate/release 10 doubles

```
double* ptr = (double*) malloc( sizeof(double)* 10 );
if(ptr) free( ptr );
ptr = NULL;
```

new and delete operators (C++)

- **new/delete can replace malloc/free with typed sizes.**
  - e.g., dynamically allocate/release a single integer

```
int* ptr = new int;
if(ptr!=nullptr) delete ptr;
ptr=nullptr;
```

- **new[]/delete[] for arrays**
  - e.g., dynamically allocate/release 10 doubles

```
double* ptr = new double[10];
if(ptr!=nullptr) delete[] ptr;
ptr=nullptr;
```

    - delete[] indicates the ptr is an array.

new and delete operators (C++)

new[]/delete[] for arrays

- delete [] for each call to new

```
typedef int* IntArrayPtr;

IntArrayPtr *m = new IntArrayPtr[5];
for (int i=0; i<5; i++)
   m[i] = new int[3];
//m is now a 5 by 3 array.

for (int i=0; i<5; i++)
    delete[] m[i];
delete[] m;
```

Dangling Pointers

- **delete p;**
  - Destroys dynamic memory
  - But p still points there! (called dangling pointer)
  - If p is dereferenced (*p) : unpredictable results!

- **Avoid dangling pointers**
  - Assign pointer to NULL after delete

```
int* ptr = new int;
if(ptr!=nullptr) delete ptr;
ptr=nullptr;
```

new operators example

Display 10.2 Basic Pointer Manipulations

```
1   //Program to demonstrate pointers and dynamic variables.
2   #include <iostream>
3   using std::cout;
4   using std::endl;


5   int main( )
6   {
7       int *p1, *p2;


8       p1 = new int;
9       *p1 = 42;
10      p2 = p1;
11      cout << "*p1 == " << *p1 << endl;
12      cout << "*p2 == " << *p2 << endl;


13      *p2 = 53;
14      cout << "*p1 == " << *p1 << endl;
15      cout << "*p2 == " << *p2 << endl;
```

new operators example

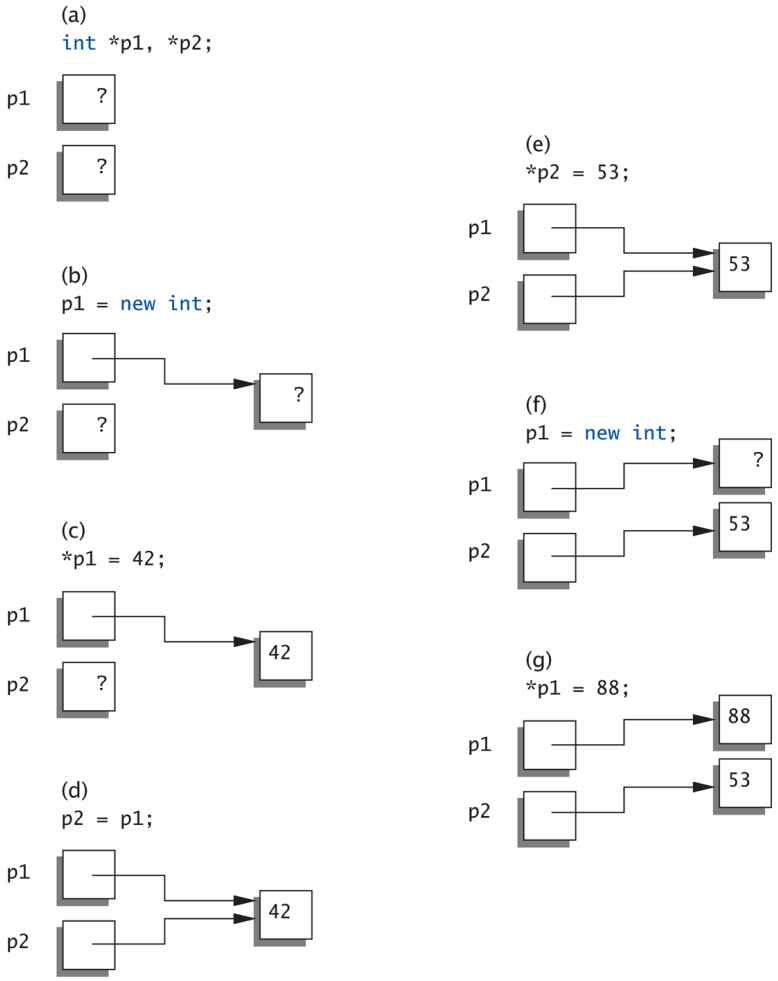
```
16        p1 = new int;
17        *p1 = 88;
18        cout << "*p1 == " << *p1 << endl;
19        cout << "*p2 == " << *p2 << endl;


20        cout << "Hope you got the point of this example!\n";
21        return 0;
22    }
```

SAMPLE DIALOGUE

```
*p1 == 42
*p2 == 42
*p1 == 53
*p2 == 53
*p1 == 88
*p2 == 53
Hope you got the point of this example!
```

new and delete operators (C++)

Display 10.3 Explanation of Display 10.2



Array pitfall

Deleting dynamic array

```
delete [] arrayPtr; //legal
delete arrayPtr[];  //illegal
```

- Position of the square brackets in delete statement is confusing

Function return returns an array

```
int[] foo(); //illegal
int*  Foo(); //legal
```

- Array type is not allowed as the return type of function in C++

More on new and delete operators (C++)

- **Classes/Structures**
  - new = allocation + invocation of constructor
  - delete = invocation of destructor + deallocation
  - constructors/destructors will be explained later.

CONSOLE I/O

Console I/O

I/O objects cin, cout

- Defined in the C++ library called <iostream>
- Must have these lines (called preprocessor directives) near start of file:

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

- Tells C++ to use library so we can use the I/O objects cin, cout, cerr

Console Output

What can be outputted?

- Any data can be outputted to display screen
  - Variables
  - Constants
  - Literals
  - Expressions (which can include all of above)

```
cout << numberOfGames << " games played.";
```

Cascading: multiple values in one cout

```
cout << welcomeMsg << numberOfGames << " games played.";
```

Separating Lines of Output

New lines in output

- Recall: "\n" is escape sequence for the char "newline"

```
cout << "Hello World\n";
```

A second method: object std::endl

```
cout << "Hello World" << endl;
```

- Sends string "Hello World" to display and "\n", skipping to next line
- Same result as above

Formatting Output

Formatting numeric values for output

- Values may not display as you'd expect!

```
cout << "The price is $" << price << endl;
```

  - If price (declared double) has value 78.5, you might get:
  - The price is $78.500000 or:
  - The price is $78.5

- We must explicitly tell C++ how to output numbers in our programs!

Formatting Numbers

- "Magic Formula" to force decimal sizes:

```
cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout.precision(2);
```

- These statements force all future cout'ed values:
  - To have exactly two digits after the decimal place

```
cout << "The price is $" << price << endl;
```

  - Now results in the following: The price is $78.50

formatting output


```
#include <iostream>

using namespace std;

int main()
{
#if 1
  cout.setf(ios::fixed);
  cout.setf(ios::showpoint);
  cout.precision(3);
#endif

  double price = 78.5909309283;
  cout << "The price is $" << price << endl;

  return 0;
}
```

Input cin

- **cin:**
  - ">>" (extraction operator) points opposite
    - Think of it as "pointing toward where the data goes"
  - No literals allowed for cin
    - Must input "to a variable"
- **cin >> num;**
  - Waits on-screen for keyboard entry
  - Value entered at keyboard is "assigned" to num

Prompting for Input: cin and cout

Always "prompt" user for input

```
cout << "Enter number of dragons: ";
cin >> numOfDragons;
```

- Note no "\n" in cout. Prompt "waits" on same line for input as follows:
  - `Enter number of dragons: ____`
  - Underscore above denotes where keyboard entry is made

In general, every cin should have cout prompt

- Maximizes user-friendly input/output

Console I/O with Class string

Just like other types!

```
string s1, s2;
cin >> s1;
cin >> s2;

cin >> s1 >> s2;

cin >> s1
    >> s2;
```

Results:

- User types in: May the hair on your toes grow long and curly!

Extraction still ignores whitespace:

- s1 receives value "May"
- s2 receives value “the"
- Will skip over any number of whitespace

Input/Output Example

- Using cin and cout with a string

```
//Program to demonstrate cin and cout with strings
#include <iostream>
#include <string>
using namespace std;

int main( )
{
    string dogName;
    int actualAge;
    int humanAge;
    cout << "How many years old is your dog? ";
    cin >> actualAge;
    humanAge = actualAge * 7;
    cout << "What is your dog's name? ";
    cin >> dogName;
    cout << dogName << "'s age is approximately " <<
     "equivalent to a " << humanAge << " year old human."
     << endl;
    return 0;
}
```

Input/Output Example 2

Using cin and cout with a string

```
How many years old is your dog? 5
What is your dog's name? Rex
Rex's age is approximately equivalent to a 35 year old human.
```

```
How many years old is your dog? 10
What is your dog's name? Mr. Bojangles
Mr.'s age is approximately equivalent to a 70 year old human.
```

- "Bojangles" is not read into dogName because cin stops input at the space.

cin example


```
1 #include <iostream>
 2 using namespace std;
 3
 4 int main(){
 5   char a[80], b[80];
 6   cout << "Enter some input:\n";
 7   //Do be do to you
 8
 9 #if 1
10   cin >> a;
11   cout << a << "END OF OUTPUT\n";
12 #endif
13
14 #if 0
15   cin >> a >> b;
16   cout << a << b << "END OF OUTPUT\n";
17 #endif
18   return 0;
19 }
```

get function

- Read one character of input

```
char c1, c2, c3, c4;
cin.get(c1);
cin.get(c2);
cin.get(c3);
cin.get(c4);

cout << “Output: ” << c1 << c2 << c3 << c4 << endl;
```

- Output

```
ABCD
Output: ABCD
```

get example


```
 1 #include <iostream>
 2 using namespace std;
 3 int main(){
 5 #if 1
 6   char c1, c2, c3, c4;
 7   cin.get(c1); cin.get(c2); cin.get(c3); cin.get(c4);
11   cout << "Output: " << c1 << c2 << c3 << endl;
12 #endif
13
14 #if 0
15   cout << "Enter a line of input and I will echo it:\n";
16   char symbol;
17   do{
18       cin.get(symbol);
19       cout << symbol;
20     } while (symbol != '\n');
21   cout << "That's all for this demonstration.\n";
22 #endif
23   return 0;}
```

getline() with C-string

- For complete lines:

```
char a[80];
cout << "Enter some input:\n”;
cin.getline(a, 80);
cout << a << "END OF OUTPUT";
```

- Dialogue produced:

```
Enter a line of input: Do be do to you!
Do be do to you! END OF INPUT
```

getline() with C-string

- For complete lines:

```
char shortString[5];
cout << "Enter some input:\n";
//abcd vs abcds vs a b c
cin.getline(shortString, 5);
```

- What would be the output?

Get line example


```
 1 #include <iostream>
 2 using namespace std;
 3
 4 int main(){
 5   char shortString[5];
 6   cout << "Enter some input:\n";
 7   //abcd vs abcds vs a b c
 8   cin.getline(shortString, 5);
 9   cout << shortString << "END OF OUTPUT\n";
10   return 0;
11 }
```

getline() with Class string

For complete lines:

```
string line;
cout << "Enter a line of input: ";
getline(cin, line);
cout << line << " END OF OUTPUT";
```

Dialogue produced:

```
Enter a line of input: Do be do to you!
Do be do to you! END OF INPUT
```

- Similar to c-string's usage of getline()

getline example


```
//Program to demonstrate cin and cout with strings
#include <iostream>
#include <string>
using namespace std;

int main( )
{
  string dogName;
  int actualAge;
  int humanAge;
  cout << "How many years old is your dog? ";
  cin >> actualAge;
  humanAge = actualAge * 7;
  cout << "What is your dog's name? ";
  // Is it enough?
  getline(cin, dogName);
  cout << dogName << "'s age is approximately " <<
    "equivalent to a " << humanAge << " year old human."
    << endl;
  return 0;
}
```

FILE I/O

File I/O Steps

1. Include file I/O library
2. Declare streams
3. Connect file to stream object
4. Check connection fail
5. Read / write file
6. Close file

Include File I/O Libraries

- **Classes ifstream and ofstream**
  - Defined in library <fstream>
  - Named in std namespace

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

Or

#include <fstream>
using std::ifstream;
using std::ofstream
```

Declare Streams

- For read file
  - ifstream
- For write file
  - ofstream

```
ifstream inStream;
ofstream outStream;
```

Connect File to Stream Object

Must connect stream to file

```
ifstream inStream;
ofstream outStream;

inStream.open("cpe_r.txt");
outStream.open("cpe_w.txt")
```

Filename at declaration

```
ifstream inStream("cpe_r.txt");
ofstream outStream("cpe_w.txt");
```

- Standard open operation begins with empty file
- Even if file exists -> contents lost

Connect File to Stream Object

Appending to a file

```
ofstream outStream;
outStream.open("cpe_w.txt", ios::app);
```

- If file doesn't exist -> creates it
- If file exists -> appends to end

Check Connection Fail

- **File opens could fail**
  - If file doesn't exist
  - No permission
  - Unexpected result

- **Check connection fail**

```
ifstream inStream;
inStream.open("cpe_r.txt");
if (inStream.fail()){
   cout << "File open failed" << end;
   exit(1);
}
```

  - fail() is member function of stream object

Read/Write & Close File

Display 12.1 Simple File Input/Output

```
1  //Reads three numbers from the file infile.txt, sums the numbers,
2  //and writes the sum to the file outfile.txt.
3  #include <fstream>
4  using std::ifstream;
5  using std::ofstream;
6  using std::endl;

7  int main( )
8  {
9      ifstream inStream;
10     ofstream outStream;

11     inStream.open("infile.txt");
12     outStream.open("outfile.txt");

13     int first, second, third;
14     inStream >> first >> second >> third;
15     outStream << "The sum of the first 3\n"
16               << "numbers in infile.txt\n"
17               << "is " << (first + second + third)
18               << endl;
```

A better version of this program is given in Display 12.3.

You can use inStream.get(second) here

You can use outStream.put('a') here

Read/Write & Close File

```
19        inStream.close( );
20        outStream.close( );

21        return 0;
22    }
```

SAMPLE DIALOGUE

There is no output to the screen and no input from the keyboard.

infile.txt
(Not changed by program)

```
1
2
3
4
```

outfile.txt
(After program is run)

```
The sum of the first 3
numbers in infile.txt
is 6
```

File I/O example


```
 1 #include <iostream>
 2 #include <fstream>
 3 #include <cstdlib>
 4
 5 using std::ifstream; using std::ofstream; using std::cout;
 8
 9 int main(){
10   ifstream fin; ofstream fout;
13   fin.open("input.txt");
14   if (fin.fail()){
15     cout << "Input file opening failed.\n";
16     exit(1); }

19   fout.open("output.txt");
20   if(fout.fail()){
21     cout << "Ouput file opening failed\n";
22     exit(1); }

25   std::string a,b,c;
26   fin >> a >> b >> c;
27   fout << a << "," << b << "," << c << std::endl;
28
29   fin.close(); fout.close();
32   return 0;
33 }
```

Check End of File

- **Use loop to process file until end**
  - Typical approach

- **Two ways to test for end of file**
  - Using member function eof()

```
inStream.get(next);
while(!inStream.eof()){
  cout << next;
  inStream.get(next);
}
```

  - Read each character until file ends
  - eof() member function returns bool

Check End of File

Second method

- Read operation return bool value

```
while(inStream >> next){
  cout << next;
}
```

- Expression return true if read successful
- Return false if attempts to read beyond end of file

File read and write


```
 1 #include <iostream>
 2 #include <fstream>
 3 #include <cstdlib>
 4
 5 using std::ifstream;
 6 using std::ofstream;
 7 using std::cout;
 8
 9 int main(){
10   ifstream fin;
11   ofstream fout;
12
13   fin.open("story.txt");
14   if (fin.fail()){
15     cout << "Input file opening failed.\n";
16     exit(1);
17   }
18
19   fout.open("numstory.txt");
20   if(fout.fail()){
21     cout << "Ouput file opening failed\n";
22     exit(1);
23   }
25   char next;
26   int n = 1;
27   fin.get(next);
28   fout << n << " ";
29   while(!fin.eof()){
30     fout << next;
31     if(next == '\n'){
32       n++;
33       fout << n << ' ';
34     }
35     fin.get(next);
36   }
37
38   fin.close();
39   fout.close();
40
41   return 0;
42 }
```

Summary

- **ifstream, ofstream**
  - 6 steps in processing file
- **Don’t forget to check file connection result**
- **Don’t forget to close file**
- **Loop until meets eof**
  - get, put, <<, >>