Programming 8 : Things that Didn't Fit

Sam Meehan
University of Washington
21 March 2017

Introduction

- I'm a broken record
  - "We are only touching the surface"
  - Much more needs to be explored on your own
  - ... and with your mentor this summer
- Useful/advanced concepts that you should know
  - Reading in an external text file
  - Strings as arrays
  - Type-casting : float → int , string → float
  - Arguments to main()
  - Overloading functions : void AddStuff(int& a, int& b) vs. void AddStuff(int& a, int& b, int& c)
  - Macros : A very specific kind of one "__LINE__" and "__FILE__"
  - "maps" and how to use them
- This list may be shorter or longer depending on who you ask

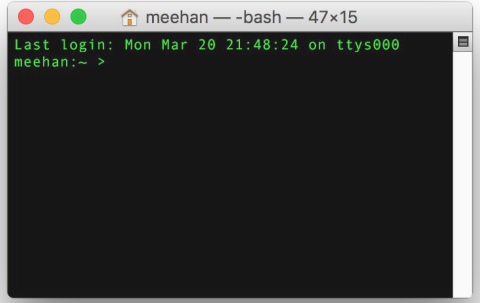
I/O (Input/Output) : Reading Files

- Data is NEVER [1] hardcoded into your programs or [2] input by user
  - Formats : ROOT files, XML files, text files ...
- Reading these efficiently (and robustly!) is important : Basic Tutorial
  - Example here for text file ➔ used for configuration of program in some cases

Program

std::ios::istream
std::ios::ostream

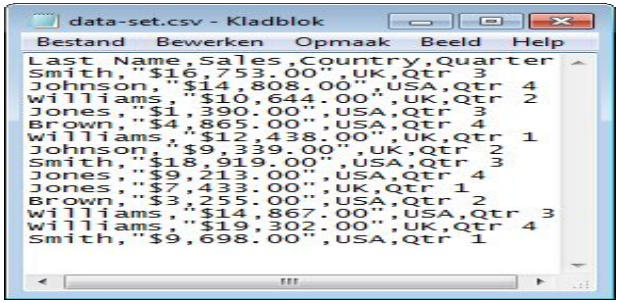
Program interacts with **terminal**



Program

std::ios::ifstream
std::ios::ofstream
std::ios::fstream

Program interacts with **external files**



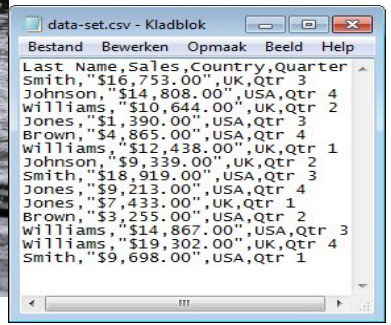
I/O (Input/Output) : Reading Files

- Reading files must be done through “streams”
  - ifstream : for the input of data from external file to program
  - ofstream : for the output of data from the program to an external file
  - fstream : for the input ***OR*** output of data to/from the program (specified expcitly)

Program

Guys in Boats :
string, int, float … “data”

The River :
std::ios::ifstream
std::ios::ofstream
std::ios::fstream



I/O Reading Files : Read

- It is important to check that things are “okay” when openning/reading
  - Corrupt files exist, empty files exist … ➔ fstream::is_open() , fstream::bad()
- By default, data read in stored as strings (next)
  - Can be automatically fed into other structures ➔ “do you want a text file?”

File Path : ~/UCFresno/Sessions/Session8_tuesday/filesread.cxx

(no symbol selected)

```cpp
//·basic·file·operations
#include·<iostream>
#include·<fstream>

using·namespace·std;

int·main·()·{
··std::cout<<"STARTING·PROGRAM"<<std::endl;

··std::ifstream·myfile;
··myfile.open·("data.txt",·std::ios::in);

··if(myfile.bad()·||·myfile.eof()){
····std::cout<<"This·file·did·not·open·...·"<<std::endl;
····std::cout<<"returning·from·main()"<<std::endl;
····return·0;
··}

··std::string·line;

··while(·get_line(myfile,line)·){
····std::cout<<line<<std::endl;
··}

··myfile.close();

··std::cout<<"========================"<<std::endl;
··std::cout<<"Finished·reading·file·-·exitting"<<std::endl;
··std::cout<<"========================"<<std::endl;

··return·0;
}
```

Line 1 Col 1 | C++ | Unicode (UTF-8) | Unix (LF) | 665 / 78 / 32

Session8_tuesday — -bash — 62×23

```
meehan:Session8_tuesday > g++ filesread.cxx
meehan:Session8_tuesday > ./a.out
STARTING PROGRAM
sam,29,Geneva
dan,34,StGenis
pat,31,Georgia
steve,21,NewYork
kate,16,Oklahoma
jannet,27,California
chase,26,Connecticut
========================
Finished reading file - exitting
========================
meehan:Session8_tuesday >
```

I/O Reading Files : Read

- It is important to check that things are “okay” when openning/reading
  - Corrupt files exist, empty files exist … ➔ fstream::is_open() , fstream::bad()
- By default, data read in stored as strings (next)
  - Can be automatically fed into other structures ➔ “do you want a text file?”

File Path : ~/UCFresno/Sessions/Session8_tuesday/filesread.cxx

(no symbol selected)

```
///basic file operations
eam>
am>

using namespace std;

int main () {
  std::cout<<"STARTING PROGRAM"<<std::endl;

  std::ifstream myfile;
  myfile.open ("data.txt", std::ios::in);

  if(myfile.bad() || myfile.eof()){
    std::cout<<"This file did not open ... "<<std::endl;
    std::cout<<"returning from main()"<<std::endl;
    return 0;
  }

  std::string line;

  while( get_line(myfile,line) ){
    std::cout<<line<<std::endl;
  }

  myfile.close();

  std::cout<<"======================="<<std::endl;
  std::cout<<"Finished reading file - exitting"<<std::endl;
  std::cout<<"======================="<<std::endl;

  return 0;
}
```

Line 1 Col 1 | C++ | Unicode (UTF-8) | Unix (LF) | 665 / 78 / 32

Make the std::ifstream class instance

open() the file and specify
[1] the name of the file (“data.txt”)
[2] the way you will interact with it (ios::in, ios::out)

Session8_tuesday — -bash — 62×23

```
meehan:Session8_tuesday > g++ filesread.cxx
y > ./a.out

steve,21,NewYork
kate,16,Oklahoma
jannet,27,California
chase,26,Connecticut
=======================
Finished reading file - exitting
=======================
meehan:Session8_tuesday >
```

I/O Reading Files : Read

- It is important to check that things are “okay” when openning/reading
  - Corrupt files exist, empty files exist … ➔ fstream::is_open() , fstream::bad()
- By default, data read in stored as strings (next)
  - Can be automatically fed into other structures ➔ “do you want a text file?”

File Path : ~/UCFresno/Sessions/Session8_tuesday/filesread.cxx

(no symbol selected)

```
///basic file operations
eam>
am>

using namespace std;

int main () {
  std::cout<<"STARTING PROGRAM"<<std::endl;

  std::ifstream myfile;
  myfile.open ("data.txt", std::ios::in);

  if(myfile.bad() || myfile.eof()){
    std::cout<<"This file did not open ...."<<std::endl;
    std::cout<<"returning from main()"<<std::endl;
    return 0;
  }

  std::string line;

  while( get_line(myfile,line) ){
    std::cout<<line<<std::endl;
  }

  myfile.close();

  std::cout<<"========================"<<std::endl;
  std::cout<<"Finished reading file - exitting"<<std::endl;
  std::cout<<"========================"<<std::endl;

  return 0;
}
```

Line 1 Col 1 | C++ | Unicode (UTF-8) | Unix (LF) | 665 / 78 / 32

Make the std::ifstream class instance

open() the file and specify
[1] the name of the file (“data.txt”)
[2] the way you will interact with it (ios::in, ios::out)

Check if the stream is “useable”

Session8_tuesday — -bash — 62×23

```
meehan:Session8_tuesday > g++ filesread.cxx
meehan:Session8_tuesday > ./a.out
steve,21,NewYork
kate,16,Oklahoma
jannet,27,California
chase,26,Connecticut
========================
Finished reading file - exitting
========================
meehan:Session8_tuesday >
```

I/O Reading Files : Read

- It is important to check that things are “okay” when openning/reading
  - Corrupt files exist, empty files exist … ➔ fstream::is_open() , fstream::bad()
- By default, data read in stored as strings (next)
  - Can be automatically fed into other structures ➔ “do you want a text file?”

File Path : ~/UCFresno/Sessions/Session8_tuesday/filesread.cxx

(no symbol selected)

```
///basic file operations
eam>
am>

using namespace std;

int main () {
  std::cout<<"STARTING PROGRAM"<<std::endl;

  std::ifstream myfile;
  myfile.open ("data.txt", std::ios::in);

  if(myfile.bad() || myfile.eof()){
    std::cout<<"This file did not open ... "<<std::endl;
    std::cout<<"returning from main()"<<std::endl;
    return 0;
  }

  std::string line;

  while( get_line(myfile,line) ){
    std::cout<<line<<std::endl;
  }

  myfile.close();

  std::cout<<"======================="<<std::endl;
  std::cout<<"Finished reading file - exitting"<<std::endl;
  std::cout<<"======================="<<std::endl;

  return 0;
}
```

Line 1 Col 1 | C++ | Unicode (UTF-8) | Unix (LF) | 665 / 78 / 32

Make the std::ifstream class instance

open() the file and specify
[1] the name of the file (“data.txt”)
[2] the way you will interact with it (ios::in, ios::out)

Check if the stream is “useable”

Read in the data of this file

Session8_tuesday — -bash — 62×23

```
meehan:Session8_tuesday > g++ filesread.cxx
meehan:Session8_tuesday > ./a.out
steve,21,NewYork
kate,16,Oklahoma
jannet,27,California
chase,26,Connecticut
=======================
Finished reading file - exitting
=======================
meehan:Session8_tuesday >
```

WARNING : Skipping binary files

I/O Reading Files : Write

- Writing files is like reading … in reverse
- IMPORTANT : Close your files after you are finished writing
  - you may inadvertently mess with them later on

Session8_tuesday — -bash — 62×23

```
[meehan:Session8_tuesday > ls -l
total 96
-rw-r--r--@ 1 meehan  staff  120 Mar 20 22:08 data.txt
-rw-r--r--@ 1 meehan  staff  664 Mar 21 06:19 filesread.cxx
-rw-r--r--@ 1 meehan  staff  435 Mar 20 22:20 fileswrite.cxx
-rw-r--r--@ 1 meehan  staff  701 Mar 20 17:24 mainargs.cxx
-rw-r--r--@ 1 meehan  staff  712 Mar 21 05:51 strings1.cxx
-rw-r--r--@ 1 meehan  staff  197 Mar 21 05:53 strings1b.cxx
-rw-r--r--@ 1 meehan  staff  256 Mar 21 05:51 strings1c.cxx
-rw-r--r--@ 1 meehan  staff  450 Mar 21 05:51 strings1d.cxx
-rw-r--r--@ 1 meehan  staff  334 Mar 21 05:56 strings2.cxx
-rw-r--r--@ 1 meehan  staff  479 Mar 21 05:57 strings3.cxx
-rw-r--r--@ 1 meehan  staff  403 Mar 21 06:10 strings4.cxx
-rw-r--r--@ 1 meehan  staff  436 Mar 21 06:16 strings5.cxx
meehan:Session8_tuesday >
```

File Path : ~/UCFresno/Sessions/Session8_tuesday/fileswrite.cxx

(no symbol selected)

```
1  #include <iostream>
2  #include <fstream>
3
4
5  int main () {
6    std::cout<<"STARTING PROGRAM"<<std::endl;
7
8    std::ofstream myfile ("example.txt");
9    if( !myfile.is_open() ){
10     std::cout << "Unable to open file"<<std::endl;
11     std::cout << "returning"<<std::endl;
12     return 0;
13   }
14
15   myfile << "This is a line.\n";
16   myfile << "This is another line.\n";
17
18   myfile.close();
19
20   std::cout<<"File written and closed"<<std::endl;
21
22   return 0;
23 }
```

Line 1 Col 1 | C++ | Unicode (UTF-8) | Unix (LF) | 435 / 5…

I/O Reading Files : Write

WARNING : Skipping binary files

- Writing files is like reading … in reverse
- IMPORTANT : Close your files after you are finished writing
  - you may inadvertently mess with them later on

Session8_tuesday — -bash — 62×23

```
[meehan:Session8_tuesday > ls -l
total 96
-rw-r--r--@ 1 meehan  staff  120 Mar 20 22:08 data.txt
-rw-r--r--@ 1 meehan  staff  664 Mar 21 06:19 filesread.cxx
-rw-r--r--@ 1 meehan  staff  435 Mar 20 22:20 fileswrite.cxx
-rw-r--r--@ 1 meehan  staff  701 Mar 20 17:24 mainargs.cxx
-rw-r--r--@ 1 meehan  staff  712 Mar 21 05:51 strings1.cxx
-rw-r--r--@ 1 meehan  staff  197 Mar 21 05:53 strings1b.cxx
-rw-r--r--@ 1 meehan  staff  256 Mar 21 05:51 strings1c.cxx
-rw-r--r--@ 1 meehan  staff  450 Mar 21 05:51 strings1d.cxx
-rw-r--r--@ 1 meehan  staff  334 Mar 21 05:56 strings2.cxx
-rw-r--r--@ 1 meehan  staff  479 Mar 21 05:57 strings3.cxx
-rw-r--r--@ 1 meehan  staff  403 Mar 21 06:10 strings4.cxx
-rw-r--r--@ 1 meehan  staff  436 Mar 21 06:16 strings5.cxx
meehan:Session8_tuesday >
```

File Path : ~/UCFresno/Sessions/Session8_tuesday/fileswrite.cxx

(no symbol selected)

```
1  #include <iostream>
2  #include <fstream>
3
4
5  int main () {
6    std::cout<<"STARTING PROGRAM"<<std::endl;
7
8    std::ofstream myfile ("example.txt");
9    if( !myfile.is_open() ){
10     std::cout << "Unable to open file"<<std::endl;
11     std::cout << "returning"<<std::endl;
12     return 0;
13   }
14
15   myfile << "This is a line.\n";
16   myfile << "This is another line.\n";
17
18   myfile.close();
19
20   std::cout<<"File written and closed"<<std::endl;
21
22   return 0;
23 }
```

Line 1 Col 1 | C++ | Unicode (UTF-8) | Unix (LF) | 435 / 5…

Can declare and open with std::fstream constructor
NOTE : fstreams come with default interaction method (e.g. ios::out for ofstream)

Push info (strings) to the file using “<<” operator

I/O Reading Files : Write

WARNING : Skipping binary files

- Writing files is like reading … in reverse
- IMPORTANT : Close your files after you are finished writing
  - you may inadvertently mess with them later on

Session8_tuesday — -bash — 62×23

```
meehan:Session8_tuesday > g++ fileswrite.cxx
meehan:Session8_tuesday > ./a.out
STARTING PROGRAM
File written and closed
meehan:Session8_tuesday > ls -l
total 184
-rwxr-xr-x  1 meehan  staff  38620 Mar 21 06:21 a.out
-rw-r--r--@ 1 meehan  staff    120 Mar 20 22:08 data.txt
-rw-r--r--  1 meehan  staff     38 Mar 21 06:21 example.txt
-rw-r--r--@ 1 meehan  staff    664 Mar 21 06:19 filesread.cxx
-rw-r--r--@ 1 meehan  staff    435 Mar 20 22:20 fileswrite.cxx
-rw-r--r--@ 1 meehan  staff    701 Mar 20 17:24 mainargs.cxx
-rw-r--r--@ 1 meehan  staff    712 Mar 21 05:51 strings1.cxx
-rw-r--r--@ 1 meehan  staff    197 Mar 21 05:53 strings1b.cxx
-rw-r--r--@ 1 meehan  staff    256 Mar 21 05:51 strings1c.cxx
-rw-r--r--@ 1 meehan  staff    450 Mar 21 05:51 strings1d.cxx
-rw-r--r--@ 1 meehan  staff    334 Mar 21 05:56 strings2.cxx
-rw-r--r--@ 1 meehan  staff    479 Mar 21 05:57 strings3.cxx
-rw-r--r--@ 1 meehan  staff    403 Mar 21 06:10 strings4.cxx
-rw-r--r--@ 1 meehan  staff    436 Mar 21 06:16 strings5.cxx
meehan:Session8_tuesday >
```

File Path : ~/UCFresno/Sessions/Session8_tuesday/fileswrite.cxx

(no symbol selected)

```
1  #include <iostream>
2  #include <fstream>
3
4
5  int main () {
6    std::cout<<"STARTING PROGRAM"<<std::endl;
7
8    std::ofstream myfile (
9    if( !myfile.is_open() )
10     std::cout << "Unable to open file"<<std::endl;
11     std::cout << "Returning"<<std::endl;
12     return 0;
13   }
14
15   myfile << "This is a line.\n";
16   myfile << "This is another line.\n";
17
18   myfile.close();
19
20   std::cout<<"File written and closed"<<std::endl;
21
22   return 0;
23 }
```

Push info (strings) to the file using “<<” operator

Line 1 Col 1 | C++ | Unicode (UTF-8) | Unix (LF) | 435 / 5…

I/O Reading Files : Write

WARNING : Skipping binary files

- Writing files is like reading … in reverse
- IMPORTANT : Close your files after you are finished writing
  - you may inadvertently mess with them later on

Session8_tuesday — -bash — 62×23

```
meehan:Session8_tuesday > g++ fileswrite.cxx
meehan:Session8_tuesday > ./a.out
STARTING PROGRAM
File written and closed
meehan:Session8_tuesday > l
total 184
-rwxr-xr-x  1 meehan  staff
-rw-r--r--@ 1 meehan  staff    120 Mar 20 22:08 data.txt
-rw-r--r--  1 meehan  staff     38 Mar 21 06:21 example.txt
-rw-r--r--@ 1 meehan  staff    664 Mar 21 06:19 filesread.cxx
-rw-r--r--@ 1 meehan  staff    435 Mar 20 22:20 fileswrite.cxx
-rw-r--r--@ 1 meehan  staff    701 Mar 20 17:24 mainargs.cxx
-rw-r--r--@ 1 meehan  staff    712 Mar 21 05:51 strings1.cxx
-rw-r--r--@ 1 meehan  staff    197 Mar 21 05:53 strings1b.cxx
-rw-r--r--@ 1 meehan  staff    256 Mar 21 05:51 strings1c.cxx
-rw-r--r--@ 1 meehan  staff    450 Mar 21 05:51 strings1d.cxx
-rw-r--r--@ 1 meehan  staff    334 Mar 21 05:56 strings2.cxx
-rw-r--r--@ 1 meehan  staff    479 Mar 21 05:57 strings3.cxx
-rw-r--r--@ 1 meehan  staff    403 Mar 21 06:10 strings4.cxx
-rw-r--r--@ 1 meehan  staff    436 Mar 21 06:16 strings5.cxx
meehan:Session8_tuesday >
```

New file is created

File Path : ~/UCFresno/Sessions/Session8_tuesday/fileswrite.cxx

(no symbol selected)

```
1  #include <iostream>
2  #include <fstream>
3
4
5  int main () {
6    std::cout<<"STARTING PROGRAM"<<std::endl;
7
8    std::ofstream myfile ("example.txt");
9    if( !myfile.is_open() ){
10     std::cout << "Unable to open file"<<std::endl;
11     std::cout << "returning"<<std::endl;
12     return 0;
13   }
14
15   myfile << "This is a line.\n";
16   myfile << "This is another line.\n";
17
18   myfile.close();
19
20   std::cout<<"File written and closed"<<std::endl;
21
22   return 0;
23 }
```

Always close your files

Line 1 Col 1 | C++ | Unicode (UTF-8) | Unix (LF) | 435 / 5...

Strings : concept

- Many parts of our programs do not *ONLY* deal with numerical data
  - e.g. labels for histogram, readable/printable statements to terminal, text input files ...
- Dealing with this data efficiently is important → strings (Link Here)
- Strings = arrays of characters (e.g. char [];)

char literal (Link Here) : specified by single quotes

char myString[]

'H'	'e'	'l'	...	'a'	m'	'\0'

escape character ('\0') : indicated the end of the string

Group	Type names*	Notes on size / precision
Character types	**char**	Exactly one byte in size. At least 8 bits.
	char16_t	Not smaller than `char`. At least 16 bits.
	char32_t	Not smaller than `char16_t`. At least 32 bits.
	wchar_t	Can represent the largest supported character set.
Integer types (signed)	**signed char**	Same size as `char`. At least 8 bits.
	signed **short** *int*	Not smaller than `char`. At least 16 bits.
	signed **int**	Not smaller than `short`. At least 16 bits.
	signed **long** *int*	Not smaller than `int`. At least 32 bits.
	signed **long long** *int*	Not smaller than `long`. At least 64 bits.
Integer types (unsigned)	**unsigned char**	(same size as their signed counterparts)
	unsigned short *int*	
	unsigned *int*	
	unsigned long *int*	
	unsigned long long *int*	
Floating-point types	**float**	
	double	Precision not less than `float`
	long double	Precision not less than `double`
Boolean type	**bool**	
Void type	**void**	no storage
Null pointer	**decltype(nullptr)**	

Strings : concept

- Many parts of our programs do not *ONLY* deal with numerical data
  - e.g. labels for histogram, readable/printable statements to terminal, text input files ...
- Dealing with this data efficiently is important ➔ strings
- Strings = arrays of characters (e.g. char [];)

Session8_tuesday — -bash — 62×23

```
meehan:Session8_tuesday > g++ strings1.cxx
meehan:Session8_tuesday > ./a.out
STARTING PROGRAM
FullPrint : HelloSam
SinglePrint 0 : H
SinglePrint 1 : e
SinglePrint 2 : l
SinglePrint 3 : l
SinglePrint 4 : o
SinglePrint 5 : S
SinglePrint 6 : a
SinglePrint 7 : m
SinglePrint 8 :
meehan:Session8_tuesday >
```

1.cxx — Disk Browser 2 (Session8_tuesday — ~/UCFresno/Sessions)

File Path : ~/UCFresno/Sessions/Session8_tuesday/strings1.cxx

(no symbol selected)

```
#include <iostream>

int main ()
{
  std::cout<<"STARTING PROGRAM"<<std::endl;

  char helloArray[] = "HelloSam";

  std::cout<<"FullPrint : "<<helloArray<<std::endl;
  std::cout<<"SinglePrint 0 : "<<helloArray[0]<<std::endl;
  std::cout<<"SinglePrint 1 : "<<helloArray[1]<<std::endl;
  std::cout<<"SinglePrint 2 : "<<helloArray[2]<<std::endl;
  std::cout<<"SinglePrint 3 : "<<helloArray[3]<<std::endl;
  std::cout<<"SinglePrint 4 : "<<helloArray[4]<<std::endl;
  std::cout<<"SinglePrint 5 : "<<helloArray[5]<<std::endl;
  std::cout<<"SinglePrint 6 : "<<helloArray[6]<<std::endl;
  std::cout<<"SinglePrint 7 : "<<helloArray[7]<<std::endl;
  std::cout<<"SinglePrint 8 : "<<helloArray[8]<<std::endl;

  return 0;
}
```

Strings : concept

- The concept of “arrays of characters” is not terribly useful
  - Cannot refill the entire array at once
  - Requires inspection/treatment of each individual element ➔ tedious

Session8_tuesday — -bash — 62×23

```
[meehan:Session8_tuesday > g++ strings1b.cxx
strings1b.cxx:9:14: error: array type 'char [9]' is not
      assignable
  helloArray = "HelloSam";
  ~~~~~~~~~~ ^
1 error generated.
meehan:Session8_tuesday >
```

... NOPE!

.cxx — Disk Browser 2 (Session8_tuesday — ~/UCFresno/Sessions)

File Path : ~/UCFresno/Sessions/Session8_tuesday/strings1b.cxx

(no symbol selected)

```
1  #include <iostream>
2
3  int main ()
4  {
5    std::cout<<"STARTING PROGRAM"<<std::endl;
6
7    char helloArray[9];
8
9    helloArray = "HelloSam";
10
11   std::cout<<"FullPrint : "<<helloArray<<std::endl;
12
13   return 0;
14 }
```

Strings : concept

- The concept of “arrays of characters” is not terribly useful
  - Cannot refill the entire array at once
  - Requires inspection/treatment of each individual element ➔ tedious

Session8_tuesday — -bash — 62×23

```
meehan:Session8_tuesday > g++ strings1c.cxx
strings1c.cxx:10:14: error: array type 'char [9]' is not
      assignable
  helloArray = {'H','e','l','l','o','S','a','m','\0'};
  ~~~~~~~~~~ ^
1 error generated.
meehan:Session8_tuesday >
```

... NOPE!

.cxx — Disk Browser 2 (Session8_tuesday — ~/UCFresno/Sessions)

File Path : ~/UCFresno/Sessions/Session8_tuesday/strings1c.cxx

(no symbol selected)

```
1  #include <iostream>
2
3  int main ()
4  {
5    std::cout<<"STARTING PROGRAM"<<std::endl;
6
7    char helloArray[9];
8
9  //   helloArray = "HelloSam";
10   helloArray = {'H','e','l','l','o','S','a','m','\0'};
11
12   std::cout<<"FullPrint : "<<helloArray<<std::endl;
13
14
15   return 0;
16 }
```

Strings : concept

- The concept of “arrays of characters” is not terribly useful
  - Cannot refill the entire array at once
- Requires inspection/treatment of each individual element ➜ tedious

Session8_tuesday — -bash — 62×23

```
meehan:Session8_tuesday > g++ strings1d.cxx
meehan:Session8_tuesday > ./a.out
STARTING PROGRAM
FullPrint : HelloSam
meehan:Session8_tuesday >
```

That’s tedious …

d.cxx — Disk Browser 2 (Session8_tuesday — ~/UCFresno/Sessions)

File Path : ~/UCFresno/Sessions/Session8_tuesday/strings1d.cxx

(no symbol selected)

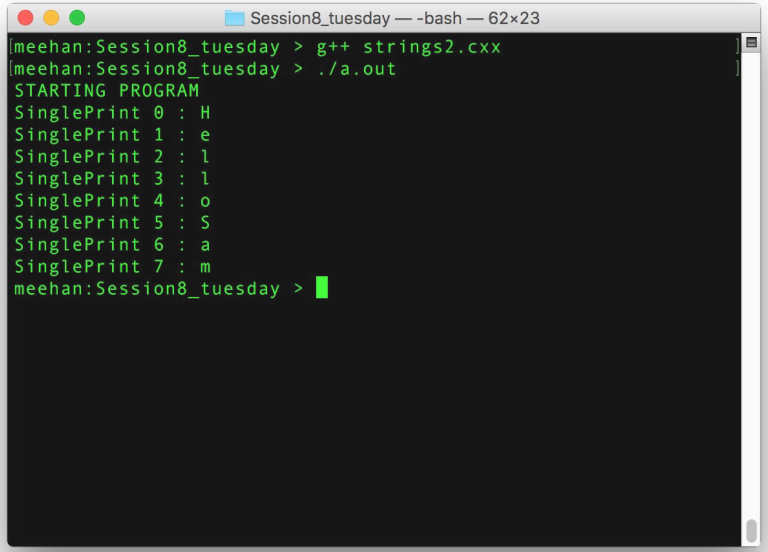
```
1  #include <iostream>
2
3  int main ()
4  {
5    std::cout<<"STARTING PROGRAM"<<std::endl;
6
7    char helloArray[9];
8
9    //   helloArray = "HelloSam";
10   //   helloArray = {'H','e','l','l','o','S','a','m','\0'};
11
12   helloArray[0]='H';
13   helloArray[1]='e';
14   helloArray[2]='l';
15   helloArray[3]='l';
16   helloArray[4]='o';
17   helloArray[5]='S';
18   helloArray[6]='a';
19   helloArray[7]='m';
20   helloArray[8]='\0';
21
22   std::cout<<"FullPrint : "<<helloArray<<std::endl;
23
24
25   return 0;
26 }
```

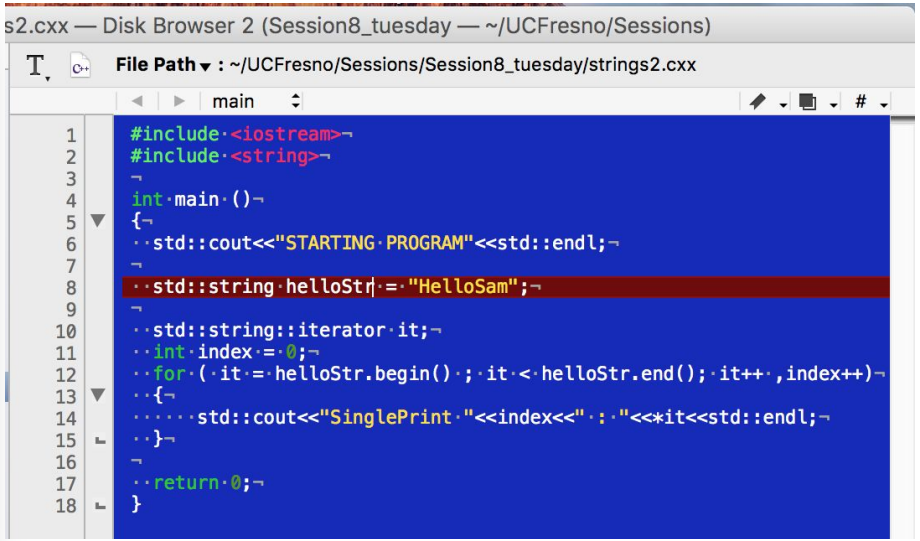
Modification of each individual character element

Line 1 Col 1 | C++ | Unicode (UTF-8) | Unix (LF) | 450 / 60 / 26

Strings : inspecting

- Strings exist as a separate std::string class - Link Here
  - Included in the standard library : #include <string>
- Because of this the usage must be expanded
  - Cons : Inspecting them takes more work → they are a data/functionality structure (i.e. a class)
  - Pros : Significant gains in functionality that “just makes sense” (because they are a class)





Strings : manipulating

- Strings exist as a separate std::string class - Link Here
  - Included in the standard library : #include <string>
- Because of this the usage must be expanded
  - Cons : Inspecting them takes a bit more work because they are a data/functionality structure
  - Pros : Significant gains in functionality that “just makes sense”

Session8_tuesday — -bash — 62×23

```
meehan:Session8_tuesday > g++ strings3.cxx
meehan:Session8_tuesday > ./a.out
STARTING PROGRAM
INIT : Testing 1,2 : Hello Harinder
INIT : Testing 3   : ZZZ
FINAL : Testing 1,2 : Hello Harinder
FINAL : Testing 3   : HelloHarinder
meehan:Session8_tuesday >
```

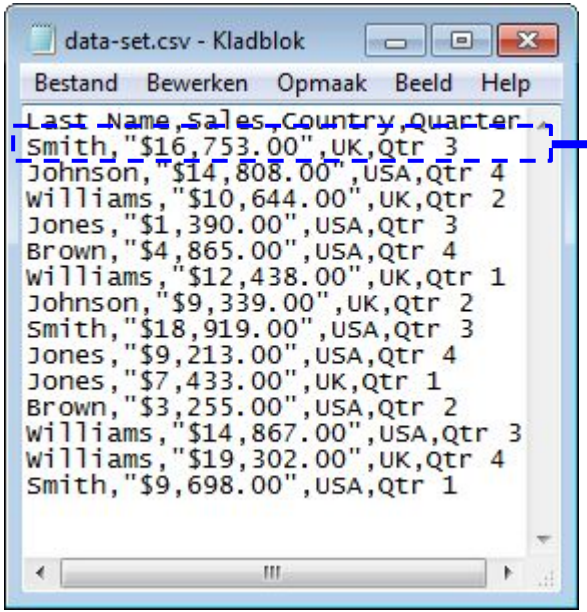
main

```
#include <iostream>
#include <string>

int main ()
{
  std::cout<<"STARTING PROGRAM"<<std::endl;

  std::string str1;
  std::string str2 = "Harinder";
  std::string str3 = "ZZZ";

  str1 = "Hello";

  std::cout<<"INIT : Testing 1,2 : "<<str1<<" "<<str2<<std::endl;
  std::cout<<"INIT : Testing 3   : "<<str3<<std::endl;

  str3 = str1+str2;

  std::cout<<"FINAL : Testing 1,2 : "<<str1<<" "<<str2<<std::endl;
  std::cout<<"FINAL : Testing 3   : "<<str3<<std::endl;

  return 0;
}
```

Concatenate = “adding strings”

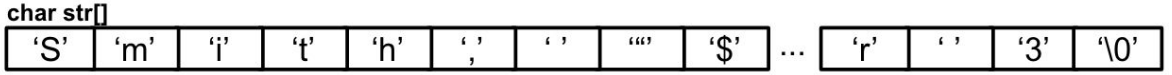
Strings : parsing (1)

- Parsing : to divide something into parts (Merriam-Webster)
- Common usage : Fixed data format that you want to read in and divide up
  - How can we do this with strings?



std::ifstream

char str[];
string str;



Strings : parsing (1)

- How can we do this with std::string? ➔ [std::string::strtok()]
  - This is one “standard” way of doing it
- Necessary to adapt usage to functionality of class
  - Arguments : the string you want to split + the list of “delimiter” characters
  - Splits off single tokens (substrings) of the text between the characters

Session8_tuesday — -bash — 58×20

```
[meehan:Session8_tuesday > g++ strings4.cxx
[meehan:Session8_tuesday > ./a.out
STARTING PROGRAM
Splitting string - This, a sample string. into tokens:
Split : 0  This
Split : 1  a
Split : 2  sample
Split : 3  string
meehan:Session8_tuesday >
```

s4.cxx — Disk Browser 2 (Session8_tuesday — ~/UCFresno/Sessions)

File Path : ~/UCFresno/Sessions/Session8_tuesday/strings4.cxx

main

```
1  #include <iostream>
2  #include <string>
3
4  int main ()
5  {
6    std::cout<<"STARTING PROGRAM"<<std::endl;
7
8    char str[] ="- This, a sample string.";
9    char * pch;
10
11   std::cout<<"Splitting string "<<str<<" into tokens:"<<std::endl;
12
13   pch = strtok (str," ,.-");
14
15   int i=0;
16
17   while (pch != NULL)
18   {
19     std::cout<<"Split : "<<i<<"  "<<pch<<std::endl;
20     pch = strtok (NULL," ,.-");
21     i++;
22   }
23
24   return 0;
25 }
```

Line 21 Col 9 | C++ | Unicode (UTF-8) | Unix (LF) | 403 / 51 / 25

Strings : parsing (1)

- How can we do this with std::string? ➔ [std::string::strtok()]
  - This is one “standard” way of doing it
- Necessary to adapt usage to functionality of class
  - Arguments : the string you want to split + the list of “delimiter” characters
  - Splits off single tokens (substrings) of the text between the characters

Session8_tuesday — -bash — 58×20

```
[meehan:Session8_tuesday > g++ strings4.cxx
[meehan:Session8_tuesday > ./a.out
STARTING PROGRAM
Splitting string - This, a sample string. into tokens:
Split : 0  This
Split : 1  a
Split : 2  sample
Split : 3  string
meehan:Session8_tuesday >
```

s4.cxx — Disk Browser 2 (Session8_tuesday — ~/UCFresno/Sessions)

File Path : ~/UCFresno/Sessions/Session8_tuesday/strings4.cxx

main

```
1  #include <iostream>
2  #include <string>
3
4  int main ()
5  {
6    std::cout<<"STARTING PROG
7
8    char str[] ="- This, a sam
9    char * pch;
10
11   std::cout<<"Splitting string "<<str<<" into tokens:"<<std::endl;
12
13   pch = strtok (str," ,.-");
14
15   int i=0;
16
17   while (pch != NULL)
18   {
19     std::cout<<"Split : "<<i<<"  "<<pch<<std::endl;
20     pch = strtok (NULL," ,.-");
21     i++;
22   }
23
24   return 0;
25 }
```

std::string::strtok() :
[1] Divides str by “ ,.-”
[2] Store location of each substring in pch

Line 21 Col 9 | C++ | Unicode (UTF-8) | Unix (LF) | 403 / 51 / 25

Strings : parsing (1)

- How can we do this with std::string? ➔ [std::string::strtok()]
  - This is one “standard” way of doing it
- Necessary to adapt usage to functionality of class
  - Arguments : the string you want to split + the list of “delimiter” characters
  - Splits off single tokens (substrings) of the text between the characters

Session8_tuesday — -bash — 58×20

```
[meehan:Session8_tuesday > g++ strings4.cxx
[meehan:Session8_tuesday > ./a.out
STARTING PROGRAM
Splitting string - This, a sample string. into tokens:
Split : 0  This
Split : 1  a
Split : 2  sample
Split : 3  string
meehan:Session8_tuesday >
```

s4.cxx — Disk Browser 2 (Session8_tuesday — ~/UCFresno/Sessions)

File Path : ~/UCFresno/Sessions/Session8_tuesday/strings4.cxx

main

```
1  #include <iostream>
2  #include <string>
3
4  int main ()
5  {
6    std::cout<<"STARTING PROGRAM"<<std::endl;
7
8    char str[] ="- This, a sample string.";
9    char * pch;
10
11   std::cout<<"Splitting string "<<str<<" into tokens:"<<std::endl;
12
13   pch = strtok (str," ,.-");
14
15   int i=0;
16
17   while (pch != NULL)
18   {
19     std::cout<<"Split : "<<i<<"  "<<pch<<std::endl;
20     pch = strtok (NULL," ,.-");
21     i++;
22   }
23
24   return 0;
25 }
```

Line 21 Col 9 | C++ | Unicode (UTF-8) | Unix (LF) | 403 / 51 / 25

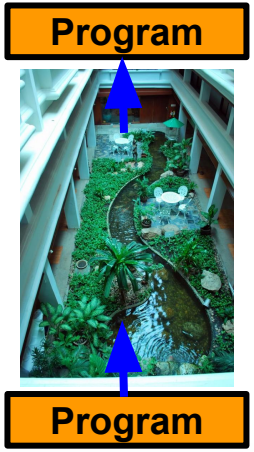
Strings - parsing (1)

- Personal take : This is confusing!
  - I only want to split my data by a single fixed type of character/string
- *“There’s more than one way to skin a cat”*

"Just how many ways <u>are</u> there to skin a cat?"

Strings : parsing (2)

- This is not the only way to do it ➔ [std::stringstream](#) , [Link Here](#)
  - Leverage the “array accessibility” of a string ➔ Parse by spaces with stringstream
- Two step process
  - [1] Make current string delimited by whitespace
  - [2] Use stringstream::operator>> to load in piecewise between whitespace




```
#include <iostream>
#include <string>
#include <sstream>

int main ()
{
  std::cout<<"STARTING PROGRAM"<<std::endl;

  std::string str = "102:330:3133:76531:451:000:12:44412";
  std::cout<<"Str INIT : "<<str<<std::endl;
  for (int i=0; i<str.length(); i++)
  {
    if (str[i] == ':')
        str[i] = ' ';
  }

  std::cout<<"Str SLICED : "<<str<<std::endl;

  std::stringstream ss(str);
  std::string temp;
  int i=0;

  while (ss >> temp){
    std::cout<<"Parsing "<<i<<" : "<<temp<<std::endl;
    i++;
  }
}
```

New header needed from standard

Replace the char/string of interest with a space

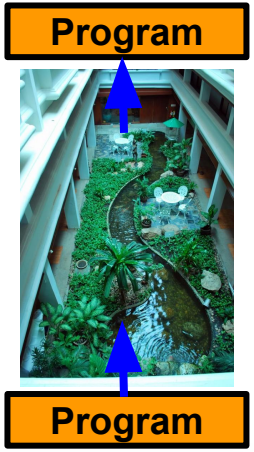
Convert the string to a stream (i.e. the stringsteam)

Re-Read in the data from the

```
meehan:Session8_tuesday > g++ strings5.cxx
meehan:Session8_tuesday > ./a.out
STARTING PROGRAM
Str INIT : 102:330:3133:76531:451:000:12:44412
Str SLICED : 102 330 3133 76531 451 000 12 44412
Parsing 0 : 102
Parsing 1 : 330
Parsing 2 : 3133
Parsing 3 : 76531
Parsing 4 : 451
Parsing 5 : 000
Parsing 6 : 12
Parsing 7 : 44412
meehan:Session8_tuesday >
```

Strings : parsing (2)

- This is not the only way to do it ➔ std::stringstream , Link Here
  - Leverage the “array accessibility” of a string ➔ Parse by spaces with stringstream
- Two step process
  - [1] Make current string delimited by whitespace
  - [2] Use stringstream::operator>> to load in piecewise between whitespace




```
#include<iostream>
#include<string>
#include<sstream>

int main ()
{
  std::cout<<"STARTING PROGRAM"<<std::endl;

  std::string str = "102:330:3133:76531:451:000:12:44412";
  std::cout<<"Str INIT : "<<str<<std::endl;
  for (int i=0; i<str.length(); i++)
  {
    if (str[i] == ':')
      str[i] = ' ';
  }

  std::cout<<"Str SLICED : "<<str<<std::endl;

  std::stringstream ss(str);
  std::string temp;
  int i=0;

  while (ss >> temp){
    std::cout<<"Parsing "<<i<<" : "<<temp<<std::endl;
    i++;
  }
}
```

New header needed from standard

Replace the char/string of interest with a space

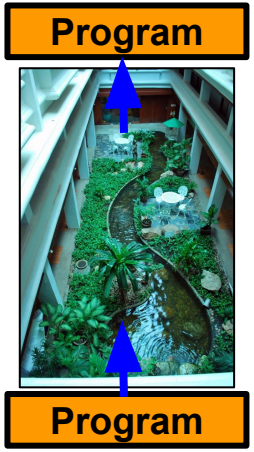
Convert the string to a stream (i.e. the stringsteam)

Re-Read in the data from the

```
meehan:Session8_tuesday > g++ strings5.cxx
meehan:Session8_tuesday > ./a.out
STARTING PROGRAM
Str INIT : 102:330:3133:76531:451:000:12:44412
Str SLICED : 102 330 3133 76531 451 000 12 44412
Parsing 0 : 102
Parsing 1 : 330
Parsing 2 : 3133
Parsing 3 : 76531
Parsing 4 : 451
Parsing 5 : 000
Parsing 6 : 12
Parsing 7 : 44412
meehan:Session8_tuesday >
```

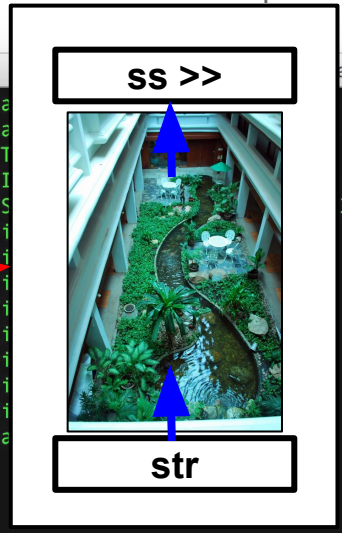
Strings : parsing (2)

- This is not the only way to do it ➔ [std::stringstream](#) , [Link Here](#)
  - Leverage the “array accessibility” of a string ➔ Parse by spaces with stringstream
- Two step process
  - [1] Make current string delimited by whitespace
  - [2] Use stringstream::operator>> to load in piecewise between whitespace




```
#include <iostream>
#include <string>
#include <vector>
#include <sstream>

int main ()
{
  std::cout<<"STARTING PROGRAM"<<std::endl;

  std::string str = "102:330:3133:76531:451:000:12:44412";
  std::cout<<"Str INIT : "<<str<<std::
  for (int i=0; i<str.length(); i++)
  {
    if (str[i] == ':')
        str[i] = ' ';
  }

  std::cout<<"Str SLICED : "<<str<<std::endl;

  std::stringstream ss(str);
  std::string temp;
  int i=0;

  while (ss >> temp){
    std::cout<<"Parsing "<<i<<" : "<<temp<<std::endl;
    i++;
  }
}
```

Convert the string to a stream
(i.e. the stringsteam)

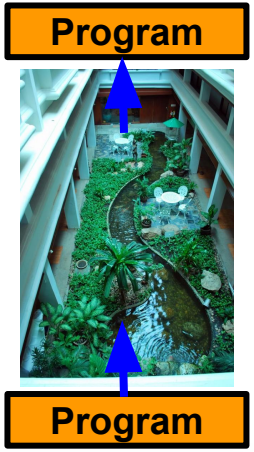



```
[meeha
[meeha
START
Str I
Str S
Parsi
Parsi
Parsi
Parsi
Parsi
Parsi
Parsi
Parsi
meeha
```

```
ay — -bash — 58×20
 strings5.cxx
.out

:451:000:12:44412
31 451 000 12 44412
```

Strings : parsing (2)

- This is not the only way to do it ➔ std::stringstream , Link Here
  - Leverage the “array accessibility” of a string ➔ Parse by spaces with stringstream
- Two step process
  - [1] Make current string delimited by whitespace
  - [2] Use stringstream::operator>> to load in piecewise between whitespace



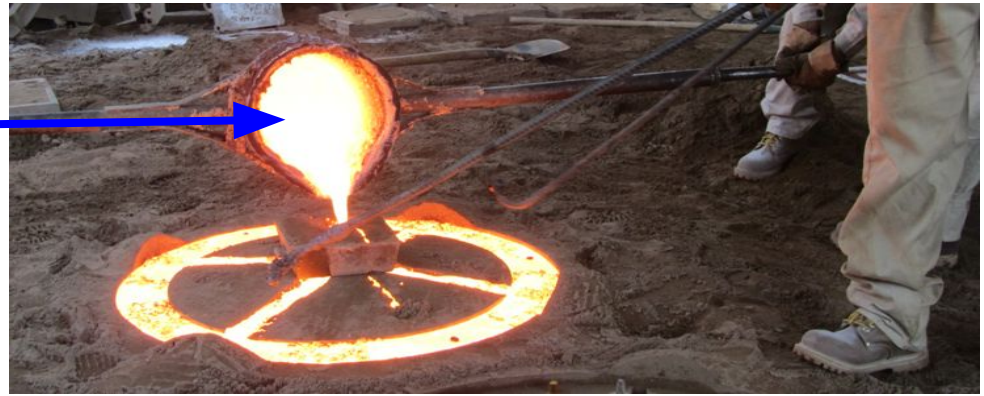

```
#include<iostream>
#include<string>
#include<vector>
#include<sstream>

int main ()
{
  std::cout<<"STARTING PROGRAM"<<std::endl;

  std::string str = "102:330:3133:76531:451:000:12:44412";
  std::cout<<"Str INIT : "<<str<<std::endl;
  for (int i=0; i<str.length(); i++)
  {
    if (str[i] == ':')
        str[i] = ' ';
  }

  std::cout<<"Str SLICED : "<<str<<std::endl;

  std::stringstream ss(str);
  std::string temp;
  int i=0;

  while (ss >> temp){
    std::cout<<"Parsing "<<i<<" : "<<temp<<std::endl;
    i++;
  }
}
```

```
meehan:Session8_tuesday > g++ strings5.cxx
meehan:Session8_tuesday > ./a.out
STARTING PROGRAM
Str INIT : 102:330:3133:76531:451:000:12:44412
Str SLICED : 102 330 3133 76531 451 000 12 44412
Parsing 0 : 102
Parsing 1 : 330
Parsing 2 : 3133
Parsing 3 : 76531
Parsing 4 : 451
Parsing 5 : 000
Parsing 6 : 12
Parsing 7 : 44412
meehan:Session8_tuesday >
```

Re-Read in the data from the

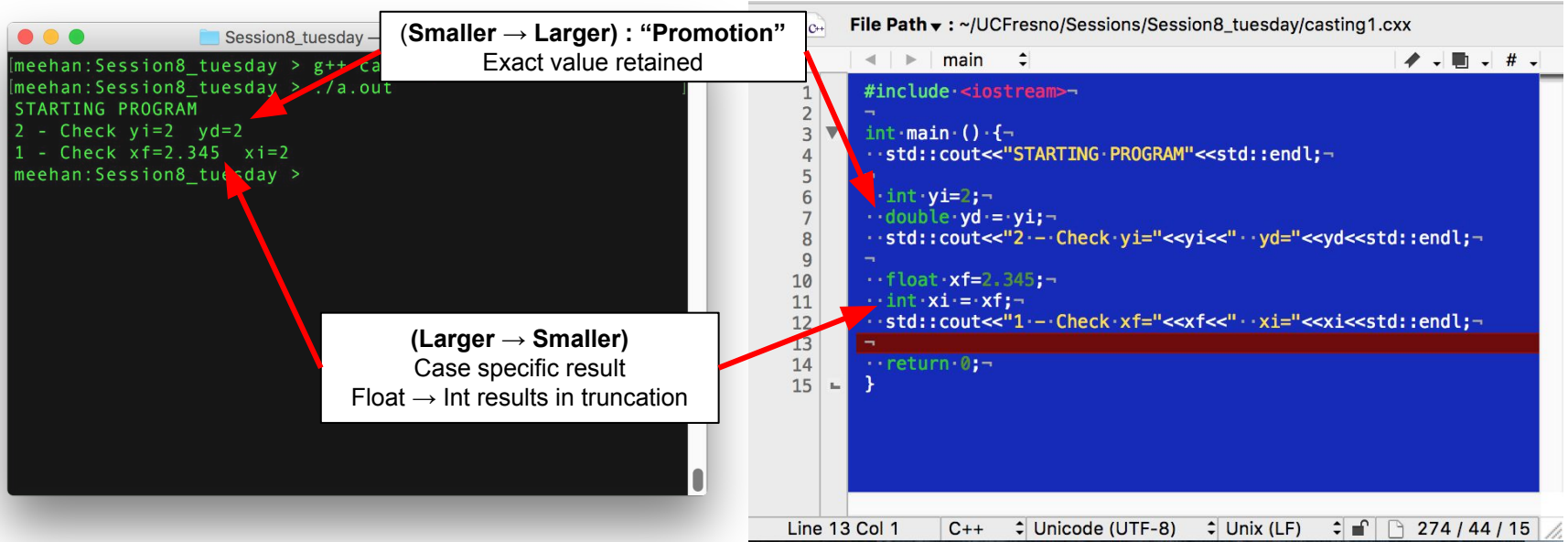
Casting - “data type manipulation”

- Cast (Merriam-Webster) : to form by this process
  - Casting changes the interpreted form of the memory that is being stored
- Manipulation of data types can be useful (but dangerous)
  - Necessary if a function takes a float as an argument but you have data as a string



Type Casting of Fundamental Types

- Conversion between fundamental numerical types is “intuitive”
  - Implicit (sloppy) : the compiler will take care of it
  - Explicit (careful) : I will specify exactly what this variable ends up as
    - [1]“c-like notation” = (new_type) var_to_cast , [2]“Functional notation” : new_type(var_to_cast)



Type Casting of Fundamental Types

- Conversion between fundamental numerical types is “intuitive”
  - Implicit (sloppy) : the compiler will take care of it
  - Explicit (careful) : I will specify **exactly** what this variable ends up as
    - [1]“c-like notation” = *(new_type)var_to_cast* , [2]“Functional notation” : *new_type(var_to_cast)*

Session8_tuesday — -bash — 58×20

```
[meehan:Session8_tuesday > g++ casting3.cxx
[meehan:Session8_tuesday > ./a.out
STARTING PROGRAM
Check
  xf=2.345
  xi_1=2
  xi_2=2
meehan:Session8_tuesday >
```

These two methods do
the same thing

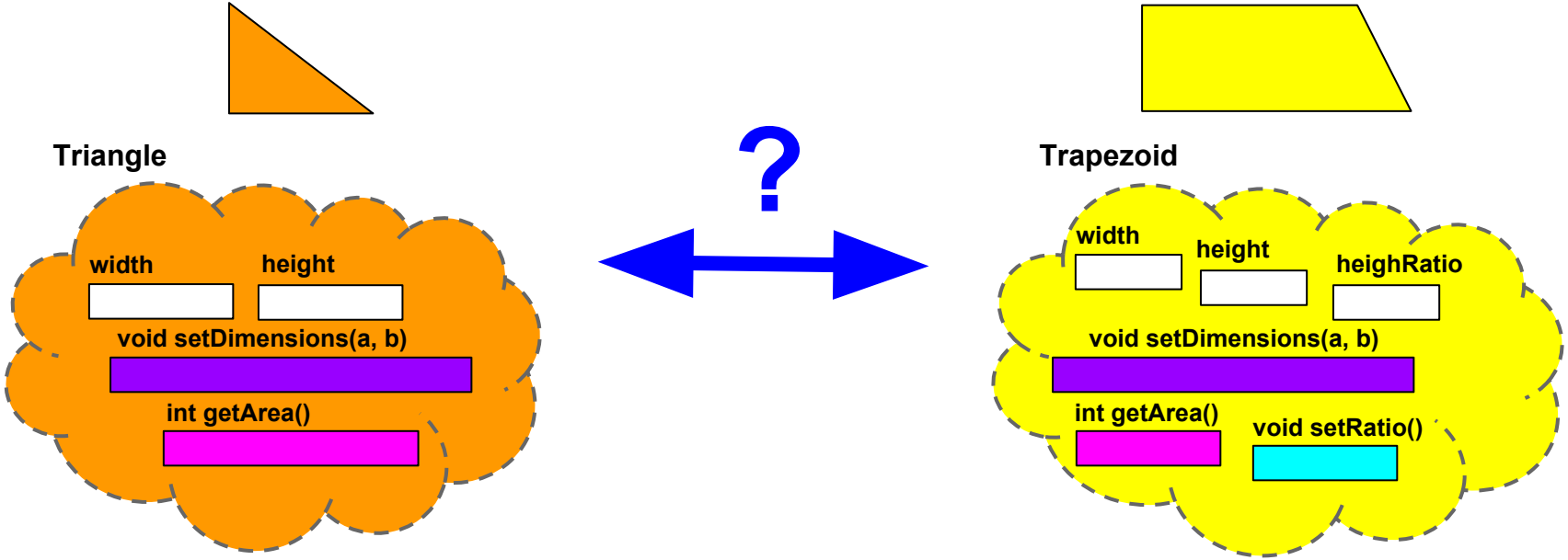
File Path : ~/UCFresno/Sessions/Session8_tuesday/casting3.cxx

main

```
1  #include <iostream>
2
3  int main () {
4    std::cout<<"STARTING PROGRAM"<<std::endl;
5
6    float xf=2.345;
7    int xi_1 = (int)xf;
8    int xi_2 = int(xf);
9
10   std::cout<<"Check "<<std::endl
11            <<"  xf="<<xf<<std::endl
12            <<"  xi_1="<<xi_1<<std::endl
13            <<"  xi_2="<<xi_2<<std::endl;
14
15   return 0;
16 }
```

Type Casting of Classes

- The memory footprint of classes is more complex?
  - Multiple member variables, functions, … etc.
- What happens if we try to “remold” an instance of one class into another?
  - If we are lucky ➔ it works , If we are unlucky ➔ it compiles (and doesn’t work)



Type Casting of Classes

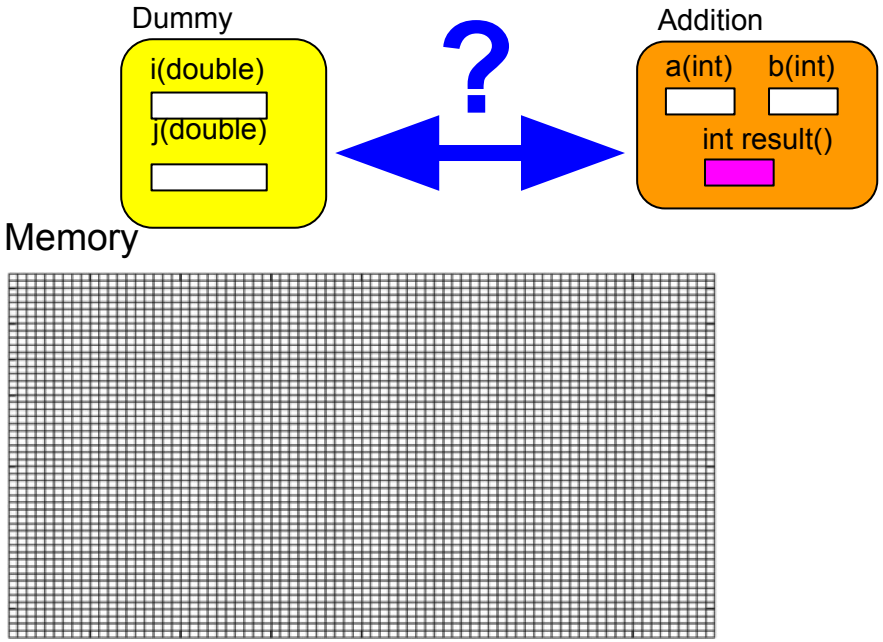
- The memory footprint of classes is more complex?
  - Multiple member variables, functions, … etc.
- What happens if we try to “remold” an instance of one class into another?
  - If we are lucky ➔ it works , If we are unlucky ➔ it compiles (and doesn’t work)

File Path : ~/UCFresno/Sessions/Session8_tuesday/casting4.cxx

(no symbol selected)

```
//·class·type-casting¬
#include·<iostream>¬

class·Dummy·{¬
····double·i,j;¬
};¬
¬
class·Addition·{¬
····int·x,y;¬
··public:¬
····Addition·(int·a,·int·b)·{·x=a;·y=b;·}¬
····int·result()·{·return·x+y;}¬
};¬

int·main·()·{¬
··std::cout<<"STARTING·PROGRAM"<<std::endl;¬
¬
··Dummy·d;¬
··Addition·*·padd;¬
··padd·=·(Addition*)·&d;¬
··std::cout<<"Result·:·"<<padd->result()<<std::endl;¬
··return·0;¬
}
```

Line 1 Col 1 | C++ | Unicode (UTF-8) | Unix (LF) | 3…



Type Casting of Classes

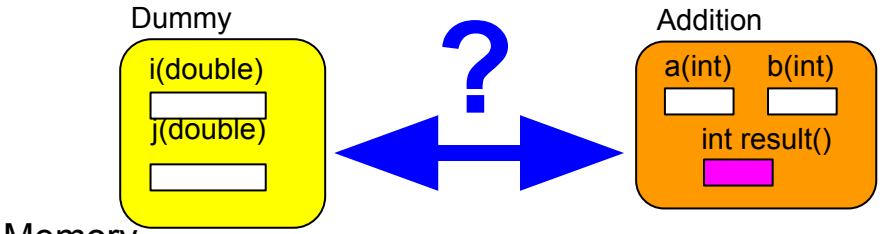
- The memory footprint of classes is more complex?
  - Multiple member variables, functions, … etc.
- What happens if we try to “remold” an instance of one class into another?
  - If we are lucky ➔ it works , If we are unlucky ➔ it compiles (and doesn’t work)

File Path : ~/UCFresno/Sessions/Session8_tuesday/casting4.cxx

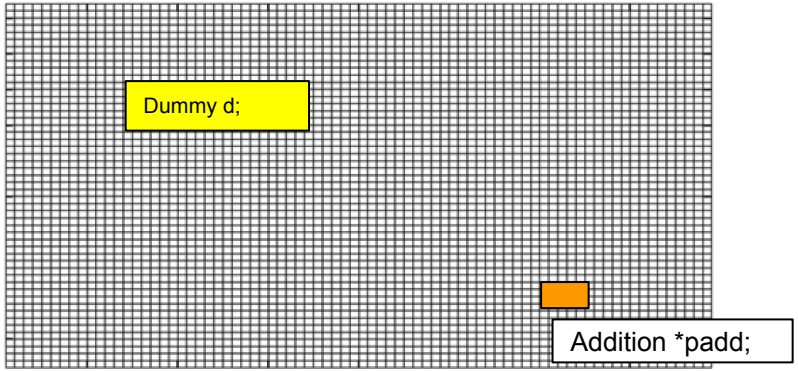
(no symbol selected)

```
1  //·class·type-casting¬
2  #include·<iostream>¬
3  ¬
4  class·Dummy·{¬
5  ····double·i,j;¬
6  };¬
7  ¬
8  class·Addition·{¬
9  ····int·x,y;¬
10 ··public:¬
11 ····Addition·(int·a,·int·b)·{·x=a;·y=b;·}¬
12 ····int·result()·{·return·x+y;}¬
13 };¬
14 ¬
15 int·main·()·{¬
16 ··std::cout<<"STARTING·PROGRAM"<<std::endl;¬
17 ¬
18 ··Dummy·d;¬
19 ··Addition·*·padd;¬
20 ··padd·=·(Addition*)·&d;¬
21 ··std::cout<<"Result·:·"<<padd->result()<<std::endl;¬
22 ··return·0;¬
23 }
```

Line 1 Col 1 | C++ | Unicode (UTF-8) | Unix (LF) | 3…



Memory



Type Casting of Classes

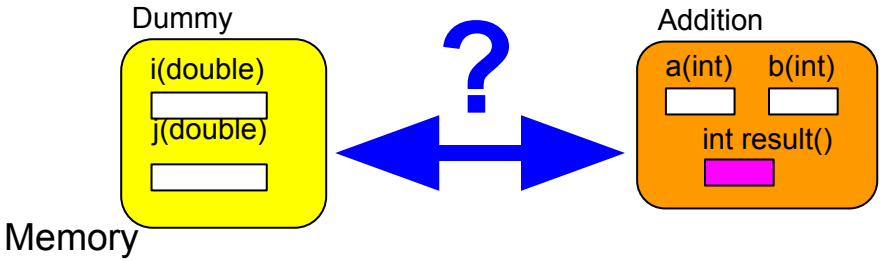
- The memory footprint of classes is more complex?
  - Multiple member variables, functions, … etc.
- What happens if we try to “remold” an instance of one class into another?
  - If we are lucky ➔ it works , If we are unlucky ➔ it compiles (and doesn’t work)

File Path : ~/UCFresno/Sessions/Session8_tuesday/casting4.cxx

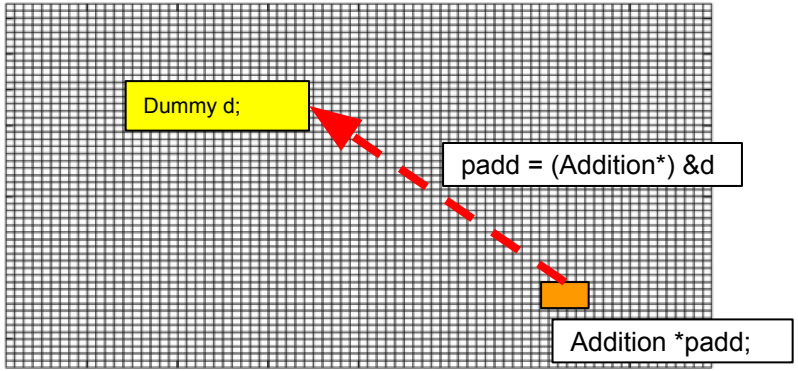
(no symbol selected)

```
//·class·type-casting¬
#include·<iostream>¬
¬
class·Dummy·{¬
····double·i,j;¬
};¬
¬
class·Addition·{¬
····int·x,y;¬
··public:¬
····Addition·(int·a,·int·b)·{·x=a;·y=b;·}¬
····int·result()·{·return·x+y;}¬
};¬
¬
int·main·()·{¬
··std::cout<<"STARTING·PROGRAM"<<std::endl;¬
¬
··Dummy·d;¬
··Addition·*·padd;¬
··padd·=·(Addition*)·&d;¬
··std::cout<<"Result·:·"<<padd->result()<<std::endl;¬
··return·0;¬
}
```

Line 1 Col 1 | C++ | Unicode (UTF-8) | Unix (LF) | 3...



Memory



Type Casting of Classes

- The memory footprint of classes is more complex?
  - Multiple member variables, functions, … etc.
- What happens if we try to “remold” an instance of one class into another?
  - If we are lucky ➔ it works , If we are unlucky ➔ it compiles (and doesn’t work)

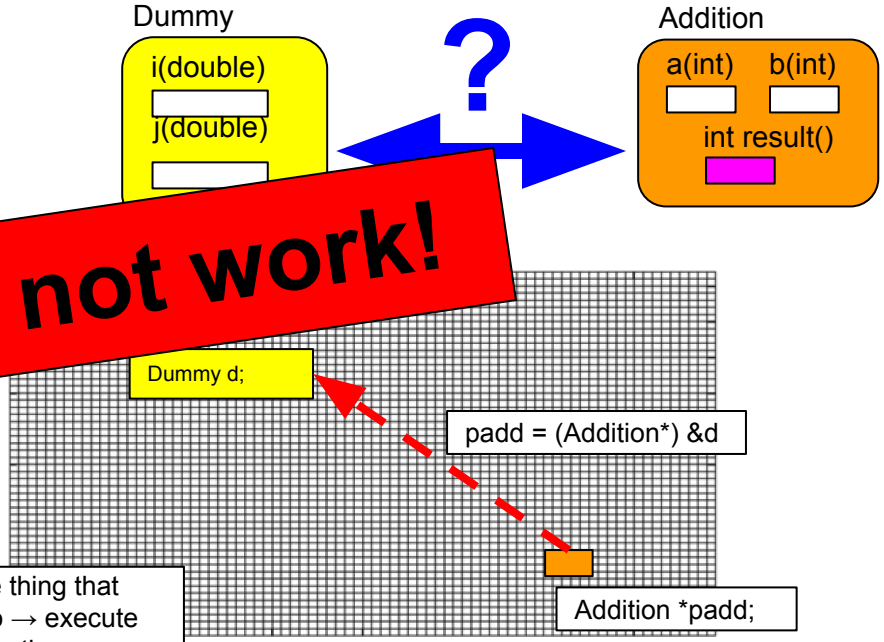
File Path : ~/UCFresno/Sessions/Session8_tuesday/casting4.cxx

(no symbol selected)

```
1  //·class·type-casting¬
2  #include·<iostream>¬
3  ¬
4  class·Dummy·{¬
5  ····double·i,j;¬
6  };¬
7  ¬
8  class·Addition·{¬
9  ····int·x,y;¬
10 ··public:¬
11 ····Add
12 ····int
13 };¬
14 ¬
15 int·main
16 ··std::co
17 ¬
18 ··Dummy·d
19 ··Addition·*·padd;¬
20 ··padd·=·(Addition*)·&d
21 ··std::cout<<"Result·:·"<<padd->result()<<std::endl;¬
22 ··return·0;¬
23 }
```

Line 1 Col 1 | C++ | Unicode (UTF-8) | Unix (LF)

This should not work!



“Search in the thing that padd points to → execute the result() function

Type Casting of Classes

- The memory footprint of classes is more complex?
  - Multiple member variables, functions, … etc.
- What happens if we try to “remold” an instance of one class into another?
  - If we are lucky → it works , If we are unlucky → it compiles (and doesn’t work)

File Path : ~/UCFresno/Sessions/Session8_tuesday/casting4.cxx

(no symbol selected)

```cpp
//·class·type-casting¬
#include·<iostream>¬
¬
class·Dummy·{¬
····double·i,j;¬
};¬
¬
class·Addition·{¬
····int·x,y;¬
··public:¬
····Addition·(int·a,·int·b)·{·x=a;·y=b;·}¬
····int·result()·{·return·x+y;}¬
};¬
¬
int·main·()·{¬
··std::cout<<"STARTING·PROGRAM"<<std::endl;¬
¬
··Dummy·d;¬
··Addition·*·padd;¬
··padd·=·(Addition*)·&d;¬
··std::cout<<"Result·:·"<<padd->result()<<std::endl;¬
··return·0;¬
}
```

Line 1 Col 1 | C++ | Unicode (UTF-8) | Unix (LF) | 3…

Session8_tuesday — -bash — 58×20

```
[meehan:Session8_tuesday > g++ casting4.cxx
[meehan:Session8_tuesday > ./a.out
STARTING PROGRAM
Result : 0
meehan:Session8_tuesday >
```

Why 0??? … this output is RUBBISH!

More Complex Casting : dynamic

- Previous page : Pointer/class casting can be dangerous
- Methods exist that allow for access/protection ➔ *dynamic_cast*
  - checks that the object you are starting from is complete
  - upcasting (derived ➔ base) : resultant pointer may lose some attributes
  - downcasting (base ➔ derived) : only valid if the base class *has all necessary attributes*

Session8_tuesday — -bash — 57×20

```
meehan:Session8_tuesday > g++ casting5.cxx
meehan:Session8_tuesday > ./a.out
STARTING PROGRAM
Null pointer on second type-cast.
ENDING PROGRAM
meehan:Session8_tuesday >
```

File Path : ~/UCFresno/Sessions/Session8_tuesday/casting5.cxx

class Base

```
1  //·dynamic_cast¬
2  #include·<iostream>¬
3  using·namespace·std;¬
4  ¬
5  class·Base·{¬
6  ··virtual·void·dummy()·{}¬
7  };¬
8  class·Derived:·public·Base·{¬
9  ··int·a;¬
10 };¬
11 ¬
12 int·main·()·{¬
13 ··std::cout<<"STARTING·PROGRAM"<<std::endl;¬
14 ¬
15 ··Base·*·pba·=·new·Derived;¬
16 ··Base·*·pbb·=·new·Base;¬
17 ··Derived·*·pd;¬
18 ¬
19 ··pd·=·dynamic_cast<Derived*>(pba);¬
20 ··if·(pd==0)·cout·<<·"Null·pointer·on·first·type-cast.\n";¬
21 ··pd·=·dynamic_cast<Derived*>(pbb);¬
22 ··if·(pd==0)·cout·<<·"Null·pointer·on·second·type-cast.\n";¬
23 ¬
24 ··std::cout<<"ENDING·PROGRAM"<<std::endl;¬
25 ··return·0;¬
26 }
```

More Complex Casting : dynamic

- Previous page : Pointer/class casting can be dangerous
- Methods exist that allow for access/protection ➔ *dynamic_cast*
  - checks that the object you are starting from is complete
  - upcasting (derived ➔ base) : resultant pointer may lose some attributes
  - downcasting (base ➔ derived) : only valid if the base class *has all necessary attributes*

Session8_tuesday — -bash — 57×20

```
[meehan:Session8_tuesday > g++ casting5.cxx
[meehan:Session8_tuesday > ./a.out
STARTING PROGRAM
Null pointer on second type-cast.
ENDING PROGRAM
meehan:Session8_tuesday >
```

File Path : ~/UCFresno/Sessions/Session8_tuesday/casting5.cxx

class Base

```
1  //·dynamic_cast¬
2  #include·<iostream>¬
3  using·namespace·std;¬
4  ¬
5  class·Base·{¬
6  ··virtual·void·dummy()·{}¬
7  };¬
8  class·Derived:·public·Base·{¬
9  ··int·a;¬
10 };¬
11 ¬
12 int·main·()·{¬
13 ··std::cout<<"STARTING·PROGRAM"<<std::endl;¬
14 ¬
15 ··Base·*·pba·=·new·Derived;¬
16 ··Base·*·pbb·=·new·Base;¬
17 ··Derived·*·pd;¬
18 ¬
19 ··pd·=·dynamic_cast<Derived*>(pba);¬
20 ··if·(pd==0)·cout·<<·"Null·pointer·on·first·type-cast.\n";¬
21 ··pd·=·dynamic_cast<Derived*>(pbb);¬
22 ··if·(pd==0)·cout·<<·"Null·pointer·on·second·type-cast.\n";¬
23 ¬
24 ··std::cout<<"ENDING·PROGRAM"<<std::endl;¬
25 ··return·0;¬
26 }
```

Upcasting = “implicit casting” : (e.g. like previously)

More Complex Casting : dynamic

- Previous page : Pointer/class casting can be dangerous
- Methods exist that allow for access/protection ➔ *dynamic_cast*
  - checks that the object you are starting from is complete
  - upcasting (derived ➔ base) : resultant pointer may lose some attributes
  - downcasting (base ➔ derived) : only valid if the base class *has all necessary attributes*

Session8_tuesday — -bash — 57×20

```
[meehan:Session8_tuesday > g++ casting5.cxx
[meehan:Session8_tuesday > ./a.out
STARTING PROGRAM
Null pointer on second type-cast.
ENDING PROGRAM
meehan:Session8_tuesday >
```

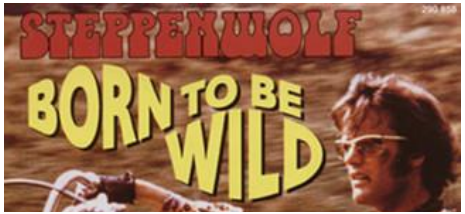
Downcasting :
pbb is a “Base” class
Going to <Derived> → ***“What is int a?”***

File Path ▾ : ~/UCFresno/Sessions/Session8_tuesday/casting5.cxx

class Base

```
1  //·dynamic_cast¬
2  #include·<iostream>¬
3  using·namespace·std;¬
4  ¬
5  class·Base·{¬
6  ··virtual·void·dummy()·{}¬
7  };¬
8  class·Derived:·public·Base·{¬
9  ··int·a;¬
10 };¬
11 ¬
12 int·main·()·{¬
13 ··std::cout<<"STARTING·PROGRAM"<<std::endl;¬
14 ¬
15 ··Base·*·pba·=·new·Derived;¬
16 ··Base·*·pbb·=·new·Base;¬
17 ··Derived·*·pd;¬
18 ¬
19 ··pd·=·dynamic_cast<Derived*>(pba);¬
20 ··if·(pd==0)·cout·<<·"Null·pointer·on·first·type-cast.\n";¬
21 ··pd·=·dynamic_cast<Derived*>(pbb);¬
22 ··if·(pd==0)·cout·<<·"Null·pointer·on·second·type-cast.\n";¬
23 ¬
24 ··std::cout<<"ENDING·PROGRAM"<<std::endl;¬
25 ··return·0;¬
26 }
```

More Complex Casting : static



- Previous page : Pointer/class casting can be dangerous
- These additional checks incur processing overhead (efficiency loss) ➔ *static_cast*
  - No checks performed ➔ YOU must ensure consistency while programming
  - Effectively the same as implicit conversion from before (may compile AND may give garbage)

dynamic_cast

File Path : ~/UCFresno/Sessions/Session8_tuesday/casting5.cxx

class Base

```
//·dynamic_cast¬
#include·<iostream>¬
using·namespace·std;¬
¬
class·Base·{¬
··virtual·void·dummy()·{}¬
};¬
class·Derived:·public·Base·{¬
··int·a;¬
};¬
¬
int·main·()·{¬
··std::cout<<"STARTING·PROGRAM"<<std::endl;¬
¬
··Base·*·pba·=·new·Derived;¬
··Base·*·pbb·=·new·Base;¬
··Derived·*·pd;¬
¬
··pd·=·dynamic_cast<Derived*>(pba);¬
··if·(pd==0)·cout·<<·"Null·pointer·on·first·type-cast.\n";¬
··pd·=·dynamic_cast<Derived*>(pbb);¬
··if·(pd==0)·cout·<<·"Null·pointer·on·second·type-cast.\n";¬
¬
··std::cout<<"ENDING·PROGRAM"<<std::endl;¬
··return·0;¬
}
```

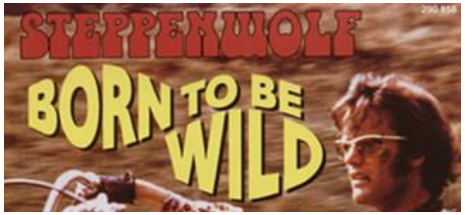
static_cast

File Path : ~/UCFresno/Sessions/Session8_tuesday/casting6.cxx

(no symbol selected)

```
1  //·static_cast¬
2  #include·<iostream>¬
3  using·namespace·std;¬
4  ¬
5  class·Base·{¬
6  ··int·a;¬
7  ··virtual·void·dummy()·{}¬
8  };¬
9  class·Derived:·public·Base·{¬
10 ··int·a;¬
11 };¬
12 ¬
13 int·main·()·{¬
14 ··std::cout<<"STARTING·PROGRAM"<<std::endl;¬
15 ¬
16 ··Base·*·pba·=·new·Derived;¬
17 ··Base·*·pbb·=·new·Base;¬
18 ··Derived·*·pd;¬
19 ¬
20 ··pd·=·static_cast<Derived*>(pba);¬
21 ··if·(pd==0)·cout·<<·"Null·pointer·on·first·type-cast.\n";¬
22 ··pd·=·static_cast<Derived*>(pbb);¬
23 ··if·(pd==0)·cout·<<·"Null·pointer·on·second·type-cast.\n";¬
24 ¬
25 ··std::cout<<"ENDING·PROGRAM"<<std::endl;¬
26 ··return·0;¬
27 }
```

More Complex Casting : static



- Previous page : Pointer/class casting can be dangerous
- These additional checks incur processing overhead (efficiency loss) ➔ *static_cast*
  - No checks performed ➔ YOU must ensure consistency while programming
  - Effectively the same as implicit conversion from before (may compile AND may give garbage)

static_cast

Session8_tuesday — -bash — 58×20

```
[meehan:Session8_tuesday > g++ casting6.cxx
[meehan:Session8_tuesday > ./a.out
STARTING PROGRAM
ENDING PROGRAM
meehan:Session8_tuesday >
```

Downcasting :
No error is thrown! We have ***NO*** guarantees of what we are working with now

File Path : ~/UCFresno/Sessions/Session8_tuesday/casting6.cxx

(no symbol selected)

```
1  // static_cast
2  #include <iostream>
3  using namespace std;
4
5  class Base {
6    int a;
7    virtual void dummy() {}
8  };
9  class Derived: public Base {
10   int a;
11 };
12
13 int main () {
14   std::cout<<"STARTING PROGRAM"<<std::endl;
15
16   Base * pba = new Derived;
17   Base * pbb = new Base;
18   Derived * pd;
19
20   pd = static_cast<Derived*>(pba);
21   if (pd==0) cout << "Null pointer on first type-cast.\n";
22   pd = static_cast<Derived*>(pbb);
23   if (pd==0) cout << "Null pointer on second type-cast.\n";
24
25   std::cout<<"ENDING PROGRAM"<<std::endl;
26   return 0;
27 }
```

Casting : strings to/from numbers

- Reading/parsing data from a file gives strings (by default)
- What if we want to convert this to numerical data for processing within code?

Can we convert a float to a std::string?

File Path : ~/UCFresno/Sessions/Session8_tuesday/casting7.cxx

main

```
1  #include <iostream>
2  #include <string>
3
4  int main () {
5    std::cout<<"STARTING PROGRAM"<<std::endl;
6
7    float xi = 4.5;
8
9    std::string xs = (std::string)xi;
10
11   std::cout<<"Check : "<<xi<<"  "<<xs<<std::endl;
12
13   return 0;
14 }
```

Line 9 Col 33 | C++ | Unicode (UTF-8) | Unix (LF)

Can we convert a std::string to a float?

File Path : ~/UCFresno/Sessions/Session8_tuesday/casting8.cxx

(no symbol selected)

```
1  #include <iostream>
2  #include <string>
3
4  int main () {
5    std::cout<<"STARTING PROGRAM"<<std::endl;
6
7    std::string xs = "4.5";
8
9    float xi = (float)xs;
10
11   std::cout<<"Check : "<<xi<<"  "<<xs<<std::endl;
12
13   return 0;
14 }
```

Line 1 Col 1 | C++ | Unicode (UTF-8) | Unix (LF)

Casting : strings to/from numbers

- Reading/parsing data from a file gives strings (by default)
- What if we want to convert this to numerical data for processing within code?

float → std::string : NO

Session8_tuesday — -bash — 74×23

```
meehan:Session8_tuesday > g++ casting7.cxx
casting7.cxx:9:20: error: no matching conversion for C-style cast from
      'float' to 'std::string' (aka 'basic_string<char, char_traits<char>,
      allocator<char> >')
  std::string xs = (std::string)xi;
                   ^~~~~~~~~~~~~~~
/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolc
hain/usr/bin/../include/c++/v1/string:1342:40: note:
      candidate constructor not viable: no known conversion from 'float'
      to 'const allocator_type' (aka 'const std::__1::allocator<char>')
      for 1st argument
    _LIBCPP_INLINE_VISIBILITY explicit basic_string(const allocato...
                              ^
/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolc
hain/usr/bin/../include/c++/v1/string:1349:5: note:
      candidate constructor not viable: no known conversion from 'float'
      to 'const std::__1::basic_string<char>' for 1st argument
    basic_string(const basic_string& __str);
    ^
/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolc
hain/usr/bin/../include/c++/v1/string:1354:5: note:
      candidate constructor not viable: no known conversion from 'float'
      to 'std::__1::basic_string<char>' for 1st argument
```

std::string → float : NO

Session8_tuesday — -bash — 74×23

```
meehan:Session8_tuesday > g++ casting8.cxx
casting8.cxx:9:14: error: cannot convert 'std::string' (aka
      'basic_string<char, char_traits<char>, allocator<char> >') to
      'float' without a conversion operator
  float xi = (float)xs;
             ^~~~~~~~~
1 error generated.
meehan:Session8_tuesday >
```

Conversion Methods

- Existing classes in the std::string library facilitate this conversion
  - Example (string → float) : [std::string::stof()]

File Path : ~/UCFresno/Sessions/Session8_tuesday/casting9.cxx

main

```
1  #include <iostream>
2  #include <string>
3
4  int main ()
5  {
6    std::string weight_str = "686.97";
7
8    float weight_float = std::stof(weight_str);
9    std::cout<<"Checking "<<std::endl
10            << " weight(str)   : "<<weight_str<<std::endl
11            << " weight(float) : "<<weight_float<<std::endl;
12   return 0;
13 }
```

Line 10 Col 57 | C++ | Unicode (UTF-8) | Unix (LF) | 303 / 3...

Session8_tuesday — -bash — 58×18

```
meehan:Session8_tuesday > g++ casting9.cxx
meehan:Session8_tuesday > ./a.out
Checking
 weight(str)   : 686.97
 weight(float) : 686.97
meehan:Session8_tuesday >
```

Arguments to main()

- We can run (complex) programs … but they only execute in one way
- Q : How can we configure our program externally such that it is more flexible
  - A : Input arguments from the command line (similar bash command) - Link Here to Tutorial

int argc

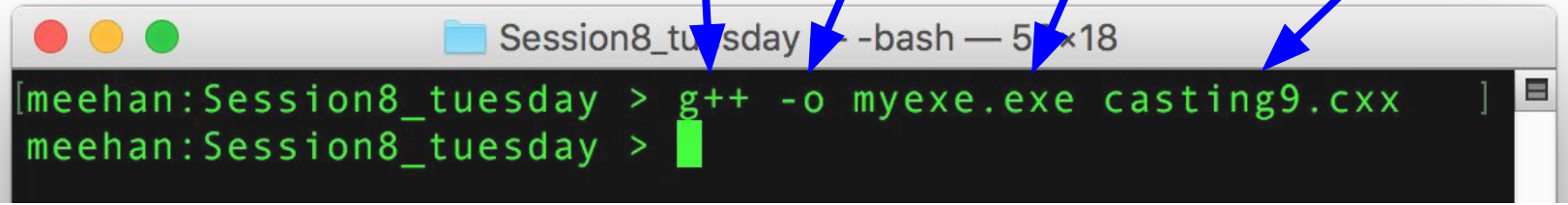
Number of arguments

"4"

char* argv[]

Vector of strings of arguments

argv[0]="g++" argv[1]="-o" argv[2]="myexe.exe" argv[3]="casting9.cxx"



How many arguments?

- Access arguments in main() by addition to its definition
  - char *argv[] : The vector<std::string> of arguments
  - int argc : How long is the argv[] vector?
- These come together and the format is expected by g++ compiler

main() function now has arguments : Format is basic c++ standard

Access these arguments in the function … just like a normal function/arguments

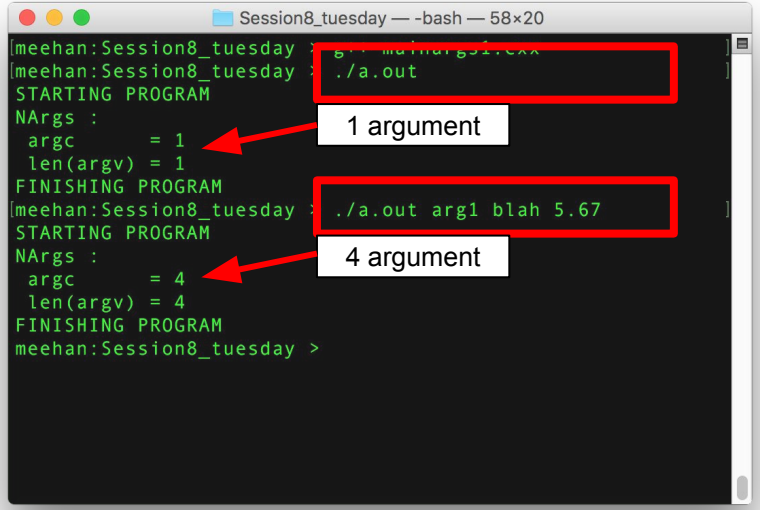
File Path : ~/UCFresno/Sessions/Session8_tuesday/mainargs1.cxx

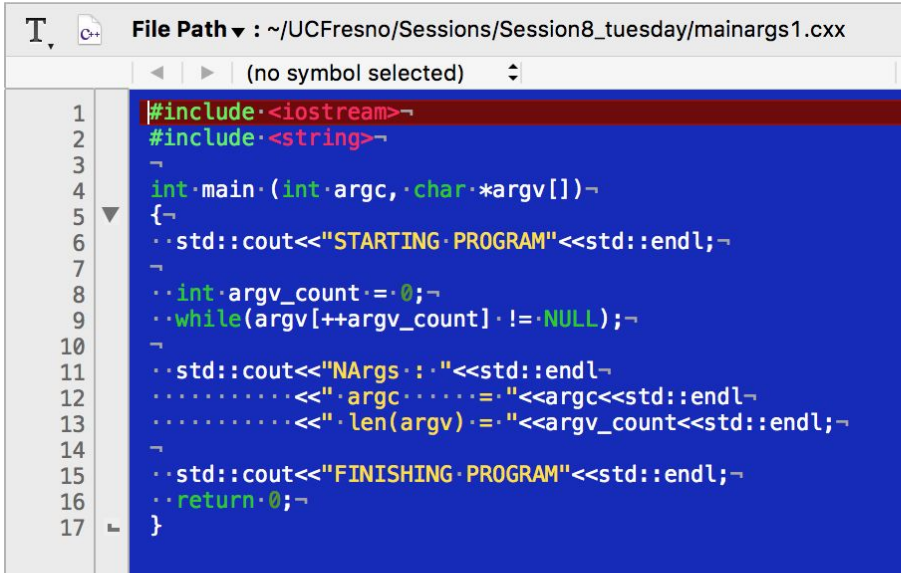
(no symbol selected)

```
1  #include <iostream>
2  #include <string>
3
4  int main (int argc, char *argv[])
5  {
6    std::cout<<"STARTING PROGRAM"<<std::endl;
7
8    int argv_count = 0;
9    while(argv[++argv_count] != NULL);
10
11   std::cout<<"NArgs : "<<std::endl
12           <<" argc      = "<<argc<<std::endl
13           <<" len(argv) = "<<argv_count<<std::endl;
14
15   std::cout<<"FINISHING PROGRAM"<<std::endl;
16   return 0;
17 }
```

How many arguments?

- Access arguments in main() by addition to its definition
  - char *argv[] : The vector<std::string> of arguments
  - int argc : How long is the argv[] vector?
- These come together and the format is expected by g++ compiler





Organization of workflow direction

- Examining the argc main() argument → redirect workflow depending on input
  - Only read arguments if argc says they will be there
  - Only use if argument read-in at program onset

Session8_tuesday — -bash — 58×20

```
[meehan:Session8_tuesday > g++ mainargs2.cxx
[meehan:Session8_tuesday > ./a.out
STARTING PROGRAM
NArgs :
 argc       = 1
Run Case A :
Arg 0 : ./a.out
FINISHING PROGRAM
[meehan:Session8_tuesday > ./a.out 4.562
STARTING PROGRAM
NArgs :
 argc       = 2
Run Case B :
Arg 0 : ./a.out
Arg 1 : 4.562
FINISHING PROGRAM
meehan:Session8_tuesday >
```

File Path : ~/UCFresno/Sessions/Session8_tuesday/mainargs

(no symbol selected)

```
#include <iostream>
#include <string>

int main (int argc, char *argv[])
{
  std::cout<<"STARTING PROGRAM"<<std::endl;

  std::cout<<"NArgs : "<<std::endl
           << " argc      = "<<argc<<std::endl;

  std::string arg0;
  std::string arg1;

  if(argc==1){
    std::cout<<"Run Case A :"<<std::endl;
    arg0 = argv[0];
    std::cout<<"Arg 0 : "<<arg0<<std::endl;
  }
  else if(argc==2){
    std::cout<<"Run Case B :"<<std::endl;
    arg0 = argv[0];
    arg1 = argv[1];
    std::cout<<"Arg 0 : "<<arg0<<std::endl;
    std::cout<<"Arg 1 : "<<arg1<<std::endl;
  }

  std::cout<<"FINISHING PROGRAM"<<std::endl;
  return 0;
}
```

Line 1 Col 1 | C++ | Unicode (UTF-8) | Unix (LF)

Making inputs "useful"

- char *argv[] main() argument will give us std::string objects
  - But we don't always want std::string's
- Casting allows us to fix this
  - std::string → float

Session8_tuesday — -bash — 58×20

```
[meehan:Session8_tuesday > g++ mainargs3.cxx
[meehan:Session8_tuesday > ./a.out
STARTING PROGRAM
Not correct arguments
... EXITTING
[meehan:Session8_tuesday > ./a.out 4.56 1.3
STARTING PROGRAM
Converting and adding :
Result : 4.56 + 1.3 = 5.86
FINISHING PROGRAM
meehan:Session8_tuesday >
```

Yay! **We made an addition machine.**

File Path ▾ : ~/UCFresno/Sessions/Session8_tuesday/mainargs3.cxx

t Options (no symbol selected)

```
1  #include <iostream>
2  #include <string>
3
4  int main (int argc, char *argv[])
5  {
6    std::cout<<"STARTING PROGRAM"<<std::endl;
7
8    float arg1;
9    float arg2;
10
11   if(argc!=3){
12     std::cout<<"Not correct arguments"<<std::endl;
13     std::cout<<"... EXITTING"<<std::endl;
14     return 0;
15   }
16
17   std::cout<<"Converting and adding :"<<std::endl;
18   arg1 = std::stof(argv[1]);
19   arg2 = std::stof(argv[2]);
20
21   float result = arg1+arg2;
22
23   std::cout<<"Result : "<<arg1<<" + "<<arg2<<" = "<<result<<std::endl;
24
25   std::cout<<"FINISHING PROGRAM"<<std::endl;
26   return 0;
27 }
```

Line 1 Col 1 | C++ | Unicode (UTF-8) | Unix (LF) | 545 / 74 / 27 | ...

Overloading Functions

- Overloading functions : “what it does is determined by how you use it”
  - “How you use it” → defined by arguments
  - [1] Different number of arguments *func(int, int)* vs. *func(int, int, int)*
  - [2] Different types of arguments *func(int, int)* vs. *func(int, int, double)*

Three “operate()” functions … :-/

File Path : ~/UCFresno/Sessions/Session8_tuesday/overloading1.cxx

operate

```
1  //·overloading·functions¬
2  #include·<iostream>¬
3  ¬
4  int·operate·(int·a,·int·b){¬
5  ··std::cout<<"Operate·with·two·integers"<<std::endl;¬
6  ··return·(a*b);¬
7  }¬
8  ¬
9  double·operate·(double·a,·double·b){¬
10 ··std::cout<<"Operate·with·two·doubles·integers"<<std::endl;¬
11 ··return·(a/b);¬
12 }¬
13 ¬
14 float·operate·(int·a,·int·b,·int·c){¬
15 ··std::cout<<"Operate·with·two·integers"<<std::endl;¬
16 ··return·(a*b*c);¬
17 }¬
```

When called in main() they behave differently

```
18 ¬
19 int·main·()¬
20 {¬
21 ··std::cout<<"STARTING·PROGRAM"<<std::endl;¬
22 ¬
23 ··int·x=5,y=2,z=8;¬
24 ··double·n=5.0,m=2.0;¬
25 ¬
26 ··int····operint·····=·operate·(x,y);¬
27 ··double·operdbl·····=·operate·(n,m);¬
28 ··float··operintmore·=·operate·(x,y,z);¬
29 ¬
30 ··std::cout·<<·"OperateInt·····:·"·<<·operint·<<·std::endl;¬
31 ··std::cout·<<·"OperateDbl·····:·"·<<·operdbl·<<·std::endl;¬
32 ··std::cout·<<·"OperateIntMore·:·"·<<·operintmore·<<·std::endl;¬
33 ¬
34 ··return·0;¬
35 }
```

Line 16 Col 16 | C++ | Unicode (UTF-8) | Unix (LF) | 790 / 115 / 35

Overloading Functions

- Overloading functions : “what it does is determined by how you use it”
  - “How you use it” ➔ defined by arguments
  - [1] Different number of arguments *func(int, int)* vs. *func(int, int, int)*
  - [2] Different types of arguments *func(int, int)* vs. *func(int, int, double)*

Session8_tuesday — -bash — 58×20

```
[meehan:Session8_tuesday > g++ overloading1.cxx
[meehan:Session8_tuesday > ./a.out
STARTING PROGRAM
Operate with two integers
Operate with two doubles integers
Operate with two integers
OperateInt      : 10
OperateDbl      : 2.5
OperateIntMore : 80
meehan:Session8_tuesday >
```

When called in main() they behave differently

```
18
19  int main ()
20  {
21    std::cout<<"STARTING PROGRAM"<<std::endl;
22
23    int x=5,y=2,z=8;
24    double n=5.0,m=2.0;
25
26    int    operint     = operate (x,y);
27    double operdbl     = operate (n,m);
28    float  operintmore = operate (x,y,z);
29
30    std::cout << "OperateInt     : " << operint << std::endl;
31    std::cout << "OperateDbl     : " << operdbl << std::endl;
32    std::cout << "OperateIntMore : " << operintmore << std::endl;
33
34    return 0;
35  }
```

Line 16 Col 16 | C++ | Unicode (UTF-8) | Unix (LF) | 790 / 115 / 35

Preprocessors : #stuff

- Preprocessors are short little bits of code defined *before* the main()
  - We've been dealing with *#include "header.h"* → compiler replaces this with code in header file
  - NEW : *#define* → Use to make "shortcut" : *#define macro-name replacement-text*
- Many others exist : [Link here for more info]
  - If you see "#" then google "preprocessor"

Session8_tuesday — -bash — 49×15

```
[meehan:Session8_tuesday > g++ preprocessor1.cxx ]
[meehan:Session8_tuesday > ./a.out ]
STARTING PROGRAM
Error in preprocessor1.cxx at line 14
Radius : 3
Area   : 28.2743
Error in preprocessor1.cxx at line 21
Min(3,28.2743) = 3
meehan:Session8_tuesday >
```

main

```
1  #include <iostream>
2  #include <math.h>
3
4  #define PI 3.14159
5
6  #define MIN(a,b)  (((a)<(b)) ? a : b)
7
8  #define CERR std::cout<<"Error in "<<__FILE__<<" at line "<<__LINE__<<std::endl;
9
10 int main ()
11 {
12   std::cout<<"STARTING PROGRAM"<<std::endl;
13
14   CERR
15
16   float radius=3.0;
17   float area=PI*pow(radius,2);
18   std::cout<<"Radius : "<<radius<<std::endl;
19   std::cout<<"Area   : "<<area<<std::endl;
20
21   CERR
22
23   std::cout<<"Min("<<radius<<","<<area<<") = "<<MIN(radius,area)<<std::endl;
24
25   return 0;
26 }
```

Preprocessors : #stuff

- Preprocessors are short little bits of code defined *before* the main()
  - We've been dealing with *#include "header.h"* → compiler replaces this with code in header file
  - NEW : *#define* → Use to make "shortcut" : *#define macro-name replacement-text*
- Many others exist : [Link here for more info]
  - If you see "#" then google "preprocessor"

[1] SYMBOL = constant number

```
meehan:Session8_tuesday > g++ preprocessor1.cxx
meehan:Session8_tuesday > ./a.out
STARTING PROGRAM
Error in preprocessor1.cxx at line 14
Radius : 3
Area   : 28.2743
Error in preprocessor1.cxx at line 21
Min(3,28.2743) = 3
meehan:Session8_tuesday >
```

```
#include <iostream>
#include <math.h>

#define PI 3.14159

#define MIN(a,b)  (((a)<(b)) ? a : b)

#define CERR std::cout<<"Error in "<<__FILE__<<" at line "<<__LINE__<<std::endl;

int main ()
{
  std::cout<<"STARTING PROGRAM"<<std::endl;

  CERR

  float radius=3.0;
  float area=PI*pow(radius,2);
  std::cout<<"Radius : "<<radius<<std::endl;
  std::cout<<"Area   : "<<area<<std::endl;

  CERR

  std::cout<<"Min("<<radius<<","<<area<<") = "<<MIN(radius,area)<<std::endl;

  return 0;
}
```

Preprocessors : #stuff

- Preprocessors are short little bits of code defined *before* the main()
  - We've been dealing with *#include "header.h"* → compiler replaces this with code in header file
  - NEW : *#define* → Use to make "shortcut" : *#define macro-name replacement-text*
- Many others exist : [Link here for more info]
  - If you see "#" then google "preprocessor"

Session8_tuesday — -bash — 49×15

```
[meehan:Session8_tuesday > g++ preprocessor1.cxx
[meehan:Se
STARTING
Error in preprocessor1.cxx at line 14
Radius : 3
Area   : 28.2743
Error in preprocessor1.cxx at line 21
Min(3,28.2743) = 3
meehan:Session8_tuesday >
```

[2] SYMBOL = simple little function

main

```
1  #include <iostream>
2  #include <math.h>
3
4  #define PI 3.14159
5
6  #define MIN(a,b)  (((a)<(b)) ? a : b)
7
8  #define CERR std::cout<<"Error in "<<__FILE__<<" at line "<<__LINE__<<std::endl;
9
10 int main ()
11 {
12   std::cout<<"STARTING PROGRAM"<<std::endl;
13
14   CERR
15
16   float radius=3.0;
17   float area=PI*pow(radius,2);
18   std::cout<<"Radius : "<<radius<<std::endl;
19   std::cout<<"Area   : "<<area<<std::endl;
20
21   CERR
22
23   std::cout<<"Min("<<radius<<","<<area<<") = "<<MIN(radius,area)<<std::endl;
24
25   return 0;
26 }
```

Preprocessors : #stuff

- Preprocessors are short little bits of code defined *before* the main()
  - We've been dealing with *#include "header.h"* → compiler replaces this with code in header file
  - NEW : *#define* → Use to make "shortcut" : *#define macro-name replacement-text*
- Many others exist : [Link here for more info]
  - If you see "#" then google "preprocessor"

Session8_tuesday — -bash — 49×15

```
[meehan:Session8_tuesday > g++ preprocessor1.cxx
[meehan:Session8_tuesday
STARTING PROGRAM
Error in preprocessor1.c
Radius : 3
Area   : 28.2743
Error in preprocessor1.cxx at line 21
Min(3,28.2743) = 3
meehan:Session8_tuesday >
```

[3] SYMBOL = Output info statement

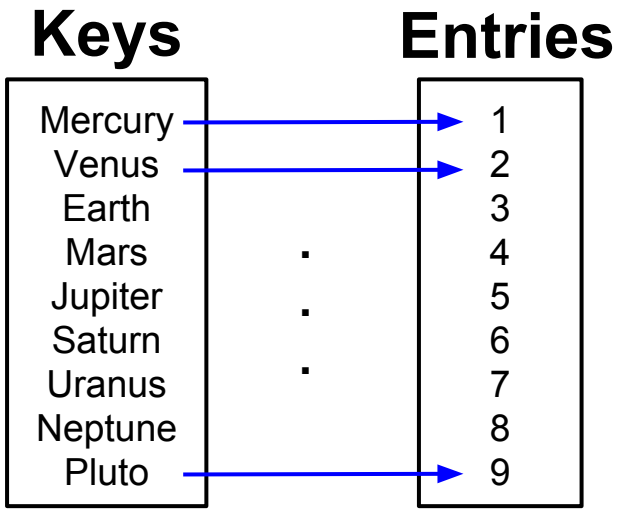
main

```
#include <iostream>
#include <math.h>

#define PI 3.14159

#define MIN(a,b)  (((a)<(b)) ? a : b)

#define CERR std::cout<<"Error in "<<__FILE__<<" at line "<<__LINE__<<std::endl;

int main ()
{
  std::cout<<"STARTING PROGRAM"<<std::endl;

  CERR

  float radius=3.0;
  float area=PI*pow(radius,2);
  std::cout<<"Radius : "<<radius<<std::endl;
  std::cout<<"Area   : "<<area<<std::endl;

  CERR

  std::cout<<"Min("<<radius<<","<<area<<") = "<<MIN(radius,area)<<std::endl;

  return 0;
}
```

__FILE__ and *__LINE__* are both protected "keywords"

Associative Memory : std::maps()

- How can we best associate a digestible label?
- Use two “related” vectors?
  - vector<string> = keys
  - vector<int> = entries



File Path : ~/UCFresno/Sessions/Session8_tuesday/maps0.cxx

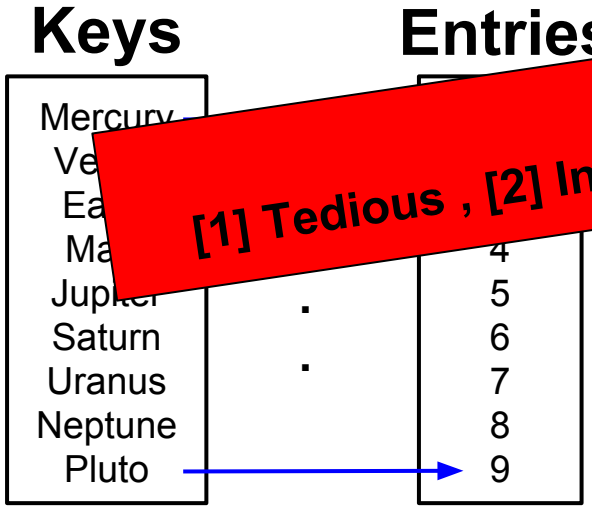
(no symbol selected)

```
#include <iostream>
#include <string>
#include <vector>

int main()
{
  std::cout<<"STARTING PROGRAM"<<std::endl;

  std::vector<std::string> name;
  std::vector<int> order;

  name.push_back("mercury");
  order.push_back(1);

  name.push_back("venus");
  order.push_back(2);

  name.push_back("earth");
  order.push_back(3);

  name.push_back("mars");
  order.push_back(4);

  name.push_back("jupiter");
  order.push_back(5);

  for(int i=0; i<(int)name.size(); i++){
    std::cout<<name.at(i)<<"  "<<order.at(i)<<std::endl;
  }

  return 0;
}
```

Line 3 Col 18 | C++ | Unicode (UTF-8) | Unix (LF) | 543 /...

Associative Memory : std::maps()

- How can we best associate a digestible label?
- Use two “related” vectors?
  - vector<string> = keys
  - vector<int> = entries



Ewwwwww!

[1] Tedious , [2] Inefficient , [3] Easily screwup-able

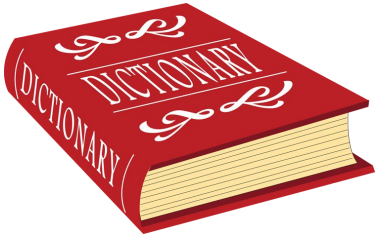
File Path : ~/UCFresno/Sessions/Session8_tuesday/maps0.cxx

(no symbol selected)

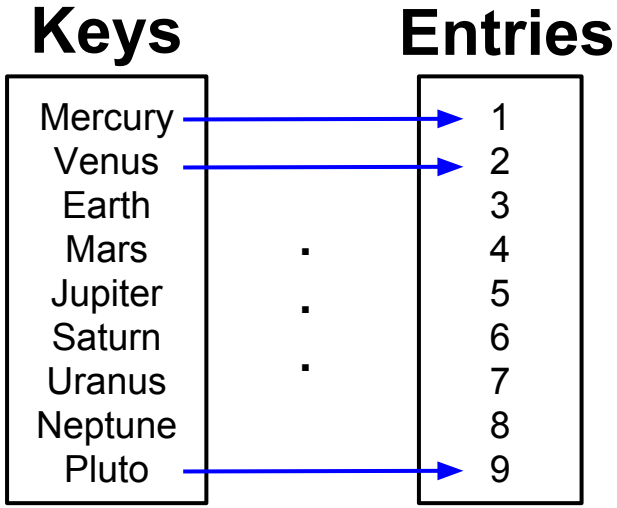
```
#include <iostream>
#include <string>
#include <vector>

int main()
{
    name.push_back("venus");
    order.push_back(2);

    name.push_back("earth");
    order.push_back(3);

    name.push_back("mars");
    order.push_back(4);

    name.push_back("jupiter");
    order.push_back(5);

    for(int i=0; i<(int)name.size(); i++){
        std::cout<<name.at(i)<<"  "<<order.at(i)<<std::endl;
    }

    return 0;
}
```

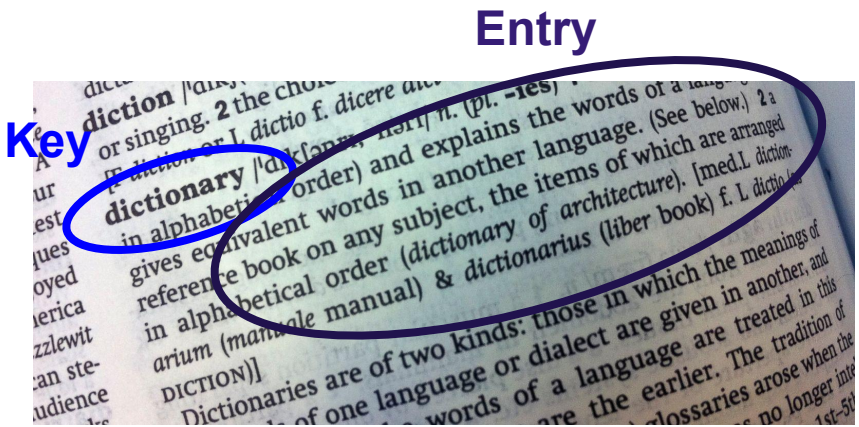
Line 3 Col 18 | C++ | Unicode (UTF-8) | Unix (LF) | 543 /...

Associative Memory : std::maps()



- How can we best associate a digestible label?
- Create a lookup table? (e.g. “Dictionary”) ➔ c++ = std::map , Link to Tutorial
  - **Keys** = labels concerning where data is located
  - **Entries** = data stored there





std::maps()

- Declare them via the template *std::map< type_key , type_entry> name;*
- Add new entries “as if it were an array”
  - Safer to use the *map.insert(make_pair(key, entry));* syntax
- Access performed through iterator

Session8_tuesday — -bash — 50×24

```
[meehan:Session8_tuesday > g++ maps1.cxx
[meehan:Session8_tuesday > ./a.out
STARTING PROGRAM
Access 0 : earth  3
Access 1 : jupiter  5
meehan:Session8_tuesday >
```

What is the order of the access based on this example?

(no symbol selected)

```
#include <iostream>
#include <map>
#include <string>
#include <iterator>

int main()
{
  std::cout<<"STARTING PROGRAM"<<std::endl;

  std::map<std::string, int> mapOfPlanets;

  //add new entries - can overwrite current entry
  mapOfPlanets["mercury"]=1;
  mapOfPlanets["venus"]=2;
  mapOfPlanets["earth"]=3;
  mapOfPlanets["mars"]=4;

  //add new entry - will NOT overwrite existing entry
  mapOfPlanets.insert( std::make_pair("jupiter", 5) );
  mapOfPlanets.insert( std::make_pair("saturn", 6) );
  mapOfPlanets.insert( std::make_pair("uranus", 7) );
  mapOfPlanets.insert( std::make_pair("neptune", 8) );
  mapOfPlanets.insert( std::make_pair("pluto", 9) );


  std::map<std::string, int>::iterator it = mapOfPlanets.begin();
  std::cout<<"Access 0 : "<<it->first<<"  "<<it->second<<std::endl;

  it++;

  std::cout<<"Access 1 : "<<it->first<<"  "<<it->second<<std::endl;

}
```

std::maps()

- Can remove elements as well
  - *map::erase()*
- Looping over entire map
  - done via iterator member access

Session8_tuesday — -bash — 50×25

```
[meehan:Session8_tuesday > g++ maps2.cxx
[meehan:Session8_tuesday > ./a.out
STARTING PROGRAM

Initial Content : 9
Looping : earth :: 3
Looping : jupiter :: 5
Looping : mars :: 4
Looping : mercury :: 1
Looping : neptune :: 8
Looping : pluto :: 9
Looping : saturn :: 6
Looping : uranus :: 7
Looping : venus :: 2

Final Content : 8
Looping : earth :: 3
Looping : jupiter :: 5
Looping : mars :: 4
Looping : mercury :: 1
Looping : neptune :: 8
Looping : saturn :: 6
Looping : uranus :: 7
Looping : venus :: 2
meehan:Session8_tuesday >
```

File Path : ~/UCFresno/Sessions/Session8_tuesday/maps2.cxx

(no symbol selected)

```cpp
#include <iostream>
#include <map>
#include <string>
#include <iterator>

int main()
{
  std::cout<<"STARTING PROGRAM"<<std::endl;

  std::map<std::string, int> mapOfPlanets;

  //add new entries - can overwrite current entry
  mapOfPlanets["mercury"]=1;
  mapOfPlanets["venus"]=2;
  mapOfPlanets["earth"]=3;
  mapOfPlanets["mars"]=4;

  //add new entry - will NOT overwrite existing entry
  mapOfPlanets.insert( std::make_pair("jupiter", 5) );
  mapOfPlanets.insert( std::make_pair("saturn", 6) );
  mapOfPlanets.insert( std::make_pair("uranus", 7) );
  mapOfPlanets.insert( std::make_pair("neptune", 8) );
  mapOfPlanets.insert( std::make_pair("pluto", 9) );

  std::cout<<std::endl<<"Initial Content : "<<mapOfPlanets.size()<<std::endl;
  std::map<std::string, int>::iterator it = mapOfPlanets.begin();
  while(it != mapOfPlanets.end()){
      std::cout<<"Looping : "<<it->first<<" :: "<<it->second<<std::endl;
      it++;
  }

  mapOfPlanets.erase("pluto");  //removing element

  std::cout<<std::endl<<"Final Content : "<<mapOfPlanets.size()<<std::endl;
  it = mapOfPlanets.begin();
  while(it != mapOfPlanets.end()){
      std::cout<<"Looping : "<<it->first<<" :: "<<it->second<<std::endl;
      it++;
  }

  return 0;
}
```

Line 1 Col 1 | C++ | Unicode (UTF-8) | Unix (LF) | Last saved: 3/21/17, 3:34:17 PM | 1,...

Conclusion

- That is all of the new c++ content that we will learn
  - Next time is purely on debugging
- I can't stress enough : ***GO EDUCATE YOURSELF***
  - This is a slow process and most effective when in the context of something (i.e. a project)