String Class / File IO

CIS 15 : Spring 2007

Functionalia

Office Hours (Last Change!) - Location Moved to 0317 N (Bridges Room)

Get Started HW 2 due on Sunday March 11, 11:59pm

Note: Midterm is on MONDAY, March 12th

Today:

- Command Line Args
- Wrap up String Class
- Begin File IO

Command Line Arguments

Remember in UNIX you can specify your behavior of running a program by supplying the UNIX command with **command line arguments.**

```
$ ls -l
```

`-l` is one **command line argument**

```
$ who am I
```

`am I` are two command line argument

```
$ g++ source.cpp -o bubba.exe
```

How many command line arguments are there?

Command Line Arguments

Remember in UNIX you can specify your behavior of running a program by supplying the UNIX command with **command line arguments.**

```
$ g++ source.cpp -o bubba.exe
```

0 1 2 3

Three?

Well, actually 4! The name of the program counts as a command line arguement:

How can we make our programs react to command line arguments?

It happens in `main()` ...

We are accustomed to specifying main in the follow fashion:

`int main();` or `int main(void);` (for those who are picky)

However, in C++ one can specify functions with a **different number** and **types** of **parameters** and **return** types!

(called Function/Operator Overloading)

Thus, another way of specifying main is:

```
int main(int argc, char * argv[]);
```

Argument Count
counts the number of arguments

Argument Vector
contains the arguments as strings

Try it out...

```
int main(int argc, char * argv[])
{
  for(int i = i; i < argc; i++)
    cout << argv[i] << " ";

  cout << endl;

  return 0;
}
```

```
... in UNIX ...

$ g++ mycode.cpp -o mirror
$ mirror hey
mirror hey
$ mirror hey quit making fun of me
mirror hey quit making fun of me
```

Error checking...

A common way to do quick error checking and behavior branching is to check the argument count.

If there are not enough arguments, respond with a help message.

```
int main(int argc, char * argv[])
{
  if(argc > 2)
  {
    cout << "Error! Usage:" << endl << argv[0] << " <filename>" << endl;
    return -1;
  }
  return 0;

}

...

$ awesome cool.txt neato.txt

Error! Usage:

awesome <filename>

$
```

Time Comparison

For HW 2, Part2 - Do a Time Comparison for the Selection Sort and the QuickSort Algorithms discussed in class. In order to do that you need to:

1. Use the UNIX utility **`time`**.

2. Make your Selection Sort Program and your QuickSort Program be able to take input on the command line.

`time` Utility

It's very simple.

```
$ time ls -l

...

real     0m0.006s
user     0m0.001s
sys      0m0.004s

$ time selectionsort ABC123231abcaba123

...

real     0m0.006s
user     0m0.001s
sys      0m0.004s
```

Make sure that you are NOT in the **`tcsh`** shell. (**`ksh`** is default)

`time` Utility

It's very simple.

```
$ time ls -l

...

real    0m0.006s
user    0m0.001s
sys     0m0.004s

$ time selectionsort ABC123231abcaba123

...

real    0m0.006s
user    0m0.001s
sys     0m0.004s
```

Make sure that you are NOT in the **`tcsh`** shell. (**`ksh`** is default)

`time` Utility

Run 10 different input strings of varying sizes (10 to 200 characters)

(Of course run the same input in both programs!)

```
$ time selectionsort ABC123231abcaba123
```

NOTE: Some characters in our input string may be incorrect! (Which characters are those?)

In the comments in your HW, include your results as a little table:

```
/*
Test Case 1: ajdn0238hiubdf HH90b0972DBuYBD0872G08B
Test Case 2: 9809Yoiwug78Yd0n78INPHf97B087B088070897bIHUSNIH
Test Number            Selection Sort Time                QSort Time
1                      0.0020 sec                         0.0032 sec
2                      0.0020 sec                         0.0032 sec
...
Conclusion: Selection Sort worked best.
*/
```

And now strncat

```
void strncat(char dest[], char src[], int n)
{
  int dest_i  = 0;

  for(; dest[dest_i] != '\0'; dest_i++)
  { }

  for(int src_i = 0; src_i < n; src_i++, dest_i++)
    dest[dest_i] = src[src_i];

}

....

char wow[10] = "wow";
char wee[10] = "weee";

cout << wow << endl << wee << endl << endl;

mystrncat(wow, wee, strlen(wee) + 1);

cout << wow << endl << wee << endl;
...............
```

strncat

```
....

char wow[10] = "wow";
char wee[10] = "weee";

cout << wow << endl << wee << endl << endl;

mystrncat(wow, wee, strlen(wee) + 1);

cout << wow << endl << wee << endl;

...............

wow
weee

wowweee
weee
```

String ⟶ Numbers Conversions

```
char number1[10] = "45387";
char number2[10] = "3.14";
char number3[10];

number3 = number1 + number2;

cout << number3 << endl;
```

Problem?

String ⟶ Numbers Conversions

```
char number1[10] = "45387";
char number2[10] = "3.14";
char number3[10];

number3 = number1 + number2;

cout << number3 << endl;
```

Problem?

Can't "+" char [].

cstdlib

```
#include <cstdlib>

int num;

long bigNum;

double realNum;

float smallerRealNum;


num = atoi("42");

bigNum = atol("8002566205");

realNum = atof("12.667");

smallerRealNum = atof("1.1");
```

Which is Larger / More Precision?

cstdlib

```
#include <cstdlib>

int num;
long bigNum;
double realNum;
float smallerRealNum;

num = atoi("42");
bigNum = atol("8002566205");
realNum = atof("12.667");
smallerRealNum = atof("1.1");
```

long larger than int

double more precision than float

Number ⟶ String Conversion

```
char * itoa(int value, char * output, int base);
```

Converts the **value** to a string (**output**) with the appropriate **base**.

```
Base: decimal = 10, octal = 8, hexadecimal = 16.
```

```
char myString[10];
int value = 256;

itoa(value, myString, 16);
cout << myString << endl;
...
0x00000100
```

Output and availability depends on compiler

C++ String Class

```
#include <string>
```

Probably what you have been using in CIS 1.5

Allows one to work with strings as a data type. (Like **`int`**, and **`char`**)

Unlike **`int`** and **`char`**, **`string`** is an abstract data type.

Defined as a class (i.e. an object).

(That means, it's defined in a library somewhere - as opposed to being a native data type.)

Advantages:

- Easier of use
- Dynamic in Size

Dis-Advantages:

- Differing Implementation
- Overhead! (Depends on how light-weight of an implementation you want)

Reading and Writing to the String Class

```
string name;

name = "anonymous";

cin >> name;

cout << "Your name is: " << name << endl;

...
Hi My Name is Chipp.
Your name is: Hi
...
```

`cin` reads only up to the first white-space.

```
getline(cin, name);
```

Reads a line of text (including white-spaces)

```
...
Hi My Name is Chipp.
Your name is: Hi My Name is Chipp.
```

Initializing string

Because string is a class object, it is initialized using a **constructor**.

(A special function that is called when an object in C++ is instantiated)

```
string empty; // nothing special ""

string myName("Chipp Jansen");

string copyOf(myName);

string nickName(copyOf, 3);

string snore('z', 10);

string lastName(myName, 7, 6);
```

Initializing string

Because string is a class object, it is initialized using a **constructor**.

(A special function that is called when an object in C++ is instantiated)

```
string empty; // nothing special ""

string myName("Chipp Jansen"); // "Chipp Jansen"

string copyOf(myName);

string nickName(copyOf, 3);

string snore('z', 10);

string lastName(myName, 7, 6);
```

Initializing string

Because string is a class object, it is initialized using a **constructor**.

(A special function that is called when an object in C++ is instantiated)

```
string empty; // nothing special “”

string myName(“Chipp Jansen”); // “Chipp Jansen”

string copyOf(myName); // “Chipp Jansen”

string nickName(copyOf, 3);

string snore(‘z’, 10);

string lastName(myName, 7, 6);
```

Initializing string

Because string is a class object, it is initialized using a **constructor**.

(A special function that is called when an object in C++ is instantiated)

```
string empty; // nothing special “”

string myName(“Chipp Jansen”); // “Chipp Jansen”

string copyOf(myName); // “Chipp Jansen”

string nickName(copyOf, 3); // “Chi”

string snore(‘z’, 10);

string lastName(myName, 7, 6);
```

Initializing string

Because string is a class object, it is initialized using a **constructor**.

(A special function that is called when an object in C++ is instantiated)

```
string empty; // nothing special ""

string myName("Chipp Jansen"); // "Chipp Jansen"

string copyOf(myName); // "Chipp Jansen"

string nickName(copyOf, 3); // "Chi"

string snore('z', 10); // "zzzzzzzzzz"

string lastName(myName, 7, 6);
```

Initializing string

Because string is a class object, it is initialized using a **constructor**.

(A special function that is called when an object in C++ is instantiated)

```
string empty; // nothing special ""

string myName("Chipp Jansen"); // "Chipp Jansen"

string copyOf(myName); // "Chipp Jansen"

string nickName(copyOf, 3); // "Chi"

string snore('z', 10); // "zzzzzzzzzz"

string lastName(myName, 7, 6); // "Jansen"
```

Comparing and Sorting strings

Instances of the string class can be compared (much like `int` and `char`) using : `<, >, <=, >=, ==, !=`

```
string abc("abc"), def("def"), abcd("abcd"), num("123"), upper
("ABC");


if(abc < def)

   cout << "abc comes before def" << endl;

if(abc < abcd)

   cout << "abc is shorter and thus comes before abcd" << endl;

if(abc > num)

   cout << "12345 is before a" << endl;

if(abc != ABC)

   cout << "Compare is case sensitive" << endl;
```

Concatenating and Referencing

String objects can be concatenated or appended with the `+` and the `+=` operator.

```
string abba("abba");

string cadabba("cadabba");

string magic;


magic = abba + cadabba;

cout << magic << endl;

magic += cadabba;

cout << magic << endl;

magic  += cadabba += abba;

cout << magic << endl << cadabba << endl << abba << endl;
```

Concatenating and Referencing

String objects can be concatenated or appended with the `+` and the `+=` operator.

```
string abba("abba");

string cadabba("cadabba");

string magic;


magic = abba + cadabba;

cout << magic << endl;

magic += cadabba;

cout << magic << endl;

magic  += cadabba += abba;

cout << magic << endl << cadabba << endl << abba << endl;
```

```
abbacadabba
abbacadabbacadabba
abbacadabbacadabbacadabba
abba
cadabbaabba
abba
```

Menu of Member Functions

The `string` object has a large set of functions associated to it.

Since string is a class, these functions are accessed as **member functions.**

One accesses member functions through **dot notation**.

```
string chipp("chipp");
string chippsPasswd("right");


cout << chipp.length << endl;
chippsPassword.append("on");
cout << chippsPassword << endl;
chipp.swap(chippsPasswd);
cout << "login: " << chipp << endl;
cout << "passwd: " << chippsPasswd << endl;
```

Menu of Member Functions

The `string` object has a large set of functions associated to it (**Ch 10.7**)

Since string is a class, these functions are accessed as **member functions.**

One accesses member functions through **dot notation**.

```
string chipp("chipp");
string chippsPasswd("right");

cout << chipp.length << endl;
chippsPassword.append("on");
cout << chippsPassword << endl;
chipp.swap(chippsPasswd);
cout << "login: " << chipp << endl;
cout << "passwd: " << chippsPasswd << endl;
```

```
5
righton
login: righton
passwd: chipp
```

Files.

UNIX File Permissions

As discussed in our UNIX tutorial, everything in UNIX is treated as a FILE.

Source Files, Executable Programs, Devices, Processes...

Since UNIX is a multi-user system, Files need permissions associated to them:

```
$ ls -l AllMySecrets.txt
-rwxr-xr--   1 chipp    sys   10 Feb 27 22:53 AllMySecrets.txt
```

File Permissions: **r**ead, **w**rite, e**x**ectute, (**-** no permission)

UNIX File Permissions

As discussed in our UNIX tutorial, everything in UNIX is treated as a FILE.

Source Files, Executable Programs, Devices, Processes...

Since UNIX is a multi-user system, Files need permissions associated to them:

```
$ ls -l AllMySecrets.txt

-rwxr-xr--   1 chipp    sys   10 Feb 27 22:53 AllMySecrets.txt
```

u g o

Defined in this order: **u**ser, **g**roup, **o**thers.

UNIX File Permissions

As discussed in our UNIX tutorial, everything in UNIX is treated as a FILE.

Source Files, Executable Programs, Devices, Processes...

Since UNIX is a multi-user system, Files need permissions associated to them:

```
$ ls -l AllMySecrets.txt
-rwxr-xr--   1 chipp    sys   10 Feb 27 22:53 AllMySecrets.txt
```

You must have Read and/or Write Permission to access or modify a file.

How do you change these permissions? Read up on **`chmod`**

UNIX File System

File Systems (Disk):

- FAT, NTFS — Windows
- HFS, HFS+ — Apple
- ext2, ext3 — Linux/UNIX

journaling file systems

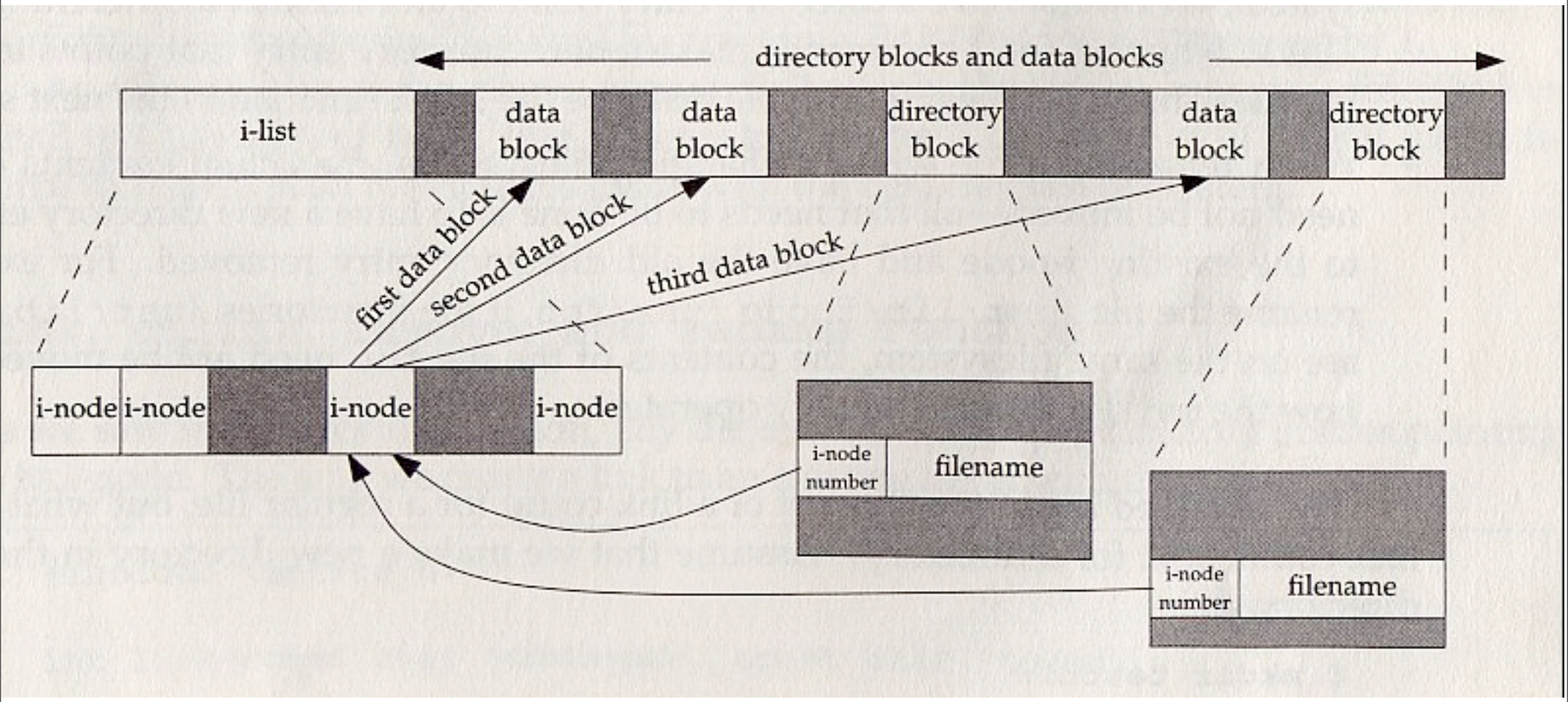
UNIX File System

UNIX: Files are stored as **inodes**.

```
$ ls -i <filename>
```

Data and Directories are stored in blocks.

Where is the filename stored?



Review: Accessing a File in C++

Reading from a File

```
#include <fstream>
...
ifstream lottery;
int numbers[7];

lottery.open("lotterynumbers.txt");

for(int i = 0; i < 7; i++)
   lottery >> numbers[i];

lottery.close();
```

Review: Accessing a File in C++

Writing to a File.

```
#include <fstream>
...
ofstream lottery;
int numbers[7] = {7,6,7,7,5,0,6};

lottery.open("lotterynumbers.txt");

for(int i = 0; i < 7; i++)
   lottery << numbers[i];

lottery.close();
```

Review: Accessing a File in C++

Error Checking on a file opening.

```
#include <fstream>
...
ifstream lottery;
int numbers[7];


lottery.open("lotterynumbers.txt");
if(lottery.fail())
   cout << "Error opening file" << endl;
else
{
   for(int i = 0; i < 7; i++)
      lottery >> numbers[i];
   lottery.close();
}
```

Review: Accessing a File in C++

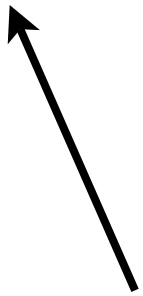
Reading until the end of the file (EOF)

```
#include <fstream>
...
ifstream lottery;
int numbers[7];
int i;
lottery.open("lotterynumbers.txt");


i = 0;
while(lottery >> numbers[i])
   i++;


lottery.close();
```

What's the danger here?

Flexible Read/Write to Files

```
#include <fstream>
...
fstream myFile;
myFile.open("journal.txt", ios::out);
```

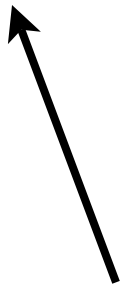


File Access Flags

```
myFile << "Writing out!";
```

Flexible Read/Write to Files

```
#include <fstream>
...
fstream myFile;
myFile.open("journal.txt", ios::in);

int code;
myFile >> code;
```



File Access Flag

Flexible Read/Write to Files

```
#include <fstream>
...
fstream myFile;
myFile.open("journal.txt", ios::in | ios::out);

int code;
myFile >> code;
code++;
myFile << code;
```

File Access Flags

File Access Flag(s)	Behavior
`ios::out`	Data written out. Existing file is deleted (by default). No file results in creation.
`ios::in`	Data read. No file results in error.
`ios::in | ios::out`	Read/Write. File preserved. No file results in creation.
`ios::out | ios::app`	Append. Go to the end of file, start writing. No file results in creation.

More?

Read Chapter 12.