CS 1570

classes.mst.edu/compsci1570
~~cbx~~ mst.edu/~price

sites.google.com/mst/classes/price

absolutely no cell phones going off

3 misses – drop

tests – 50%
homework – 40%
final homework – 10%

~~office #125~~ 325G MWF 10:00–12:30
TTh [something]

Bb – grades

1st late assignment:

1. 1st 24hrs —10%
   2nd 24hrs —50%

2nd late

1st 24hrs -50%

price @ mst.edu

GNU

indent -2spaces

~~n t t~~

ask for help — Tues and Thurs
LEAD

Hardware

I. I/O devices

II. In the box

Software
Intangible

I. OS

II. Applications
1. image editors
2. browsers
—. etc

III. Compilers
translates </code> to 01001011O

8/26/16

sorting

Big O notation
$O(n^2)$ - slow
$O(n \ln(n))$ - fast

Bogo sort - random - $O(n!)$

Bubble sort - iterate through n times swapping for order

```
//    Programmer: [name F L]   date: [date]
//    File: [file nam]
//    Purpose: [description
//                          ]

#include <iostream>

using namespace std;

int main()
{
    [declare variables]

    [do stuff]

    return 0;
}
```

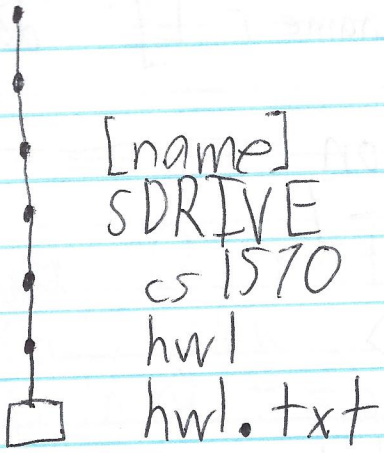
two spaces

don't go past 80th column

assignment #

$ cssubmit 1570 b 1

~~cs1570~~ ~~X~~ ~~X~~



C++

256 × 256 = 65,536

short: −32,768 → 32,767

float: [B][B][B][B] (exp, mantissa)

Variable Rules

- Variables aka identifiers — modifiable memory location
- In C++, all variables must be declared before used
- Declare all variables at the top of main()
  NEVER outside main(); declare constants first
- Make var names meaningful

syntax:

- Use only alpha-numeric and/or underscore
- Must not begin with number

Types

I. Numeric
  A. Integer
    1. short — 2bytes
    2. long — 8bytes
    3. int — SIZE NOT STANDARD ±1;
  B. Floating Point
    1. float — 4B = 6 sig figures
    2. double — 8 bytes 15 sig figs un a bazillion times.
    3. long double — NOT STANDARD
II. Non-Numeric
    1. char — 1byte 256 chars
    2. string — dynamic
    3. bool — 1 Byte (for true) (or false)

With strings C++ reads only to the first white space.
" " " → " , "\n"

cin>>"Billy" Bob";→"Billy"
cout<<"Billy"+" "+"Bob"; → Billy Bob

false ↔ 0
true ↔ 1
all other numbers → true

Constants

~~cost~~
const [type] [name] = [value];

Rules
1. Make const identifiers ALL CAPS
2. Must be initialized
3. Use constants NOT "magic numbers"
4. Don't base name on value

Casting

```
iBob = static_cast<int>(fLarry);
```

sBob/sLarry → short

Operators

- Assignment

```
lhs = rhs;
```

↑ Modifiable memory address

c = (5/9)*(f-32)

↑ always zero

use static_cast<float>(5)/9

- Arithmetic

+ − * / %

trouble maker

% a mod b is the remainder after integer division of a by b

- Fast Operators

~~++~~ −− ← actually run faster than -=-±1;

++

+= −= *= /= %=

Actually run faster!

But only use them if they run a bazillion times.

47%8
7
16%5
1
22%11
0
86%3
2

don't name "hw [#]"

~~next~~

g++ -W -Wall -s -pedia...

fg++ hw[#].cpp -o [output file]

Logical and Relational Operators

- Relational < <= > >= != ==
  return bool

trouble maker

```
if (var = 16)
```

Insidious Mistake
compiles, runs, assigns var 16

write constant on left

```
if (16 = var)
```

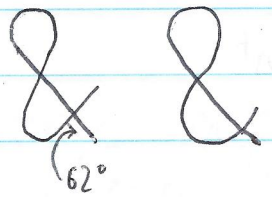
or avoid ==

```
if (!(16 != var))
```

and ∧ or ∨

- Logical Operators && || !

A	B	A∩B and	A∪B or
T	T	T	T
T	F	F	T
F	T	F	T
F	F	F	F



Order of Operations
1. relational
2. &&
3. ||

All Programming Languages
- I
- O
- Memory Mgmt
- Decision branching
- Looping

Imitation Game

Decision Branching
sytax if([condition])

Rules
1. Condition is in parenthesis
2. No semicolon [;]
3. Condition is Valid
4. Statement is simple or compound
5. else is optional
6.

if - else

```
if (     )          // comment
{
    [code]
}
else if (     )    // comment
{
    [code]
}
else                // comment
{
    [code]
}
```

don't do this

```
#include <cstdlib> // includes exit function

exit(123);  //shuts down program
```

```
  101
  100
 -----
  100
10000
-----
10100
```

```
     5050
  -------
2)10100
  10
  --
   01
    0
   --
   10
   10
   --
    00
     0
    --
```

5,050

```
int counter = 1;
short sum = 0;
while (counter <= 100)
{
    sum = sum + counter;
    counter = counter + 1;
}
cout << sum;
```

Loops

2 kinds

1. Sentinel
   - while
   - do-while
2. Counting
   - for

every loop must have a LCV (Loop Control Variable)

it must be

1. initialized
2. evaluated for update
3. updated

```
while (condition)
    statement
```

to abort infinite loops hit ctrl + C

```
do
    statement
while (condition);
```

use do-while for user input

with complex statement

```
do
{
    stuff
} while (condition);
```

same line →

for loops

counting loop

```
for( expr1 ; expr2 ; expr3)
    statement
```

expr1 — initialization of LCV

expr2 — checking of LCV

expr3 — update LCV

ex

```
int sum = 0;
int ctr = 1;
for( ctr = 1; ctr<100; ctr++)
    sum = sum + ctr;
cout << sum;
```

I'm not used to this

wrong

```
int sum = 0;
for ( int i = 0;  i <= 100; i++)
    sum += i;
cout<<sum;
cout << i;
```

works

doesn't compile

if(i % 2 == 0)
is slower than
if(!(i % 2))

code

```
for (int i = 5; i>0; i--)
{
    for(int j=i; j>0; j--)
        cout << "*";
    cout << endl;
}
```

output

```
*****
****
***
**
*
```

code

```
for (int i = 5; i>0; i--)
{
    for(int j=5-i; j>0; j--)
        cout << " ";  //2spaces
    for(int k=i; k>0; k--)
        cout << "*";
    cout << endl;
}
```

output

```
*****
 ****
  ***
   **
    *
```

Exam Friday - study there is an example test

hw4 - due next Tuesday

```
switch (key)
{
    case const1:
        statement1
        break;
    :
    :
    default:
        defaultstatement
}
```

- const1 - constn must be known at compile time
- int-types, char, bool, enum
- const1-constn must be the same type as the key

```
case x:
   statement x
case y:
   statement y
   break;
```

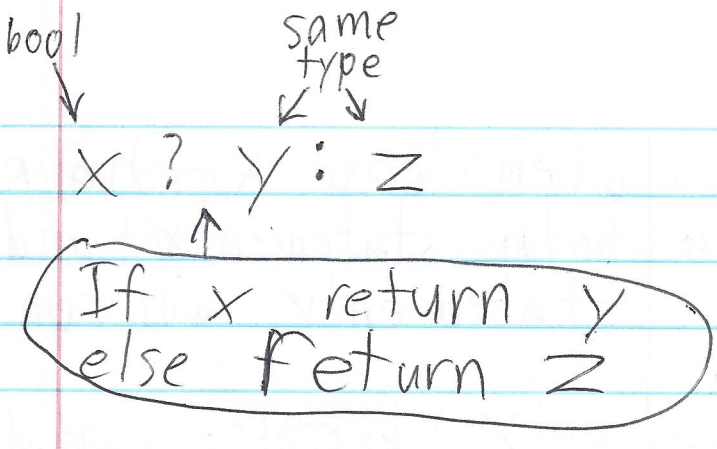
when case x happens, both statementx and statementy will run

only break; and } stop execution

Escape Sequences

\n — new line = \f + \r
\0 — null
\f — form feed
\r — carraige return
\a — alarm
\b — backspace
\'
\"
\\
\t — tab

endl flushes output buffer, it's called a manipulator



Random Numbers

```
#include <ctime>
#include <cstdlib>

int main()
{

    rand(); // returns [0, RAND_MAX]
            // returns same every time
            //
~~srand([integer]) //~~
    srand(time(NULL)); //so set seed
                       //based on time
```

On the test

- no comments in code
- round ~~static_cast<float>(static_cast<int>(n * [10^number of digits] + 0.5)) / [10^digits]~~

Functions

1. Cleanup
2. To avoid repeated code
3. Portability

```
[return type] [name]([parm list])
{
   [local const and var declarations]
   [code]
   return [value];
}
```

void — null return type

in jpico
ctrl + ins — copy selection
⇧ + ins — paste selection

prototype → `void [name]();`

```
int main()
{
    [name]();
}
```

call →

definition → `void [name]()`

```
{
    return;
}
```

only have 1 return

return
1. Copies value to its calling
2. Destroys local memory
3. Returns program control

```
void display(float num_to_print);
int main()
{
    display(bob);
}
void display(float num_to_print)
{
    return;
}
```

price likes this

price likes this

lower case

```
float cyl_vol(const float rad, const float ht);
.
.
.
  float vol;
  float the_radius = get_num();
  .
  .
  .
  vol = cyl_vol( the_radius, get_num() );
.
.
float cyl_vol...
```

every function should have a single purpose

.h

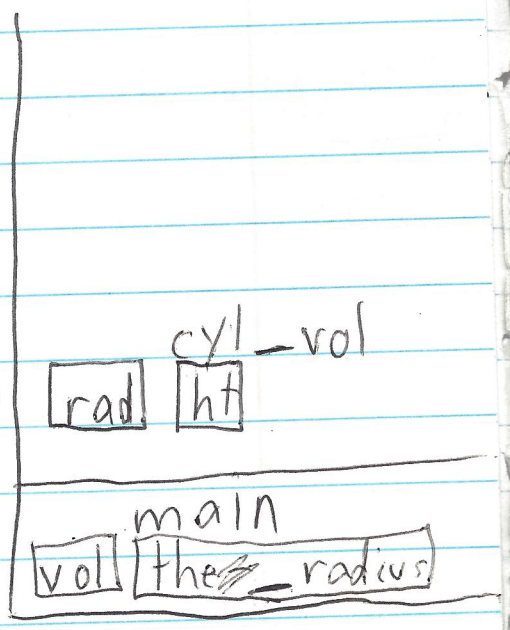
.cpp

.

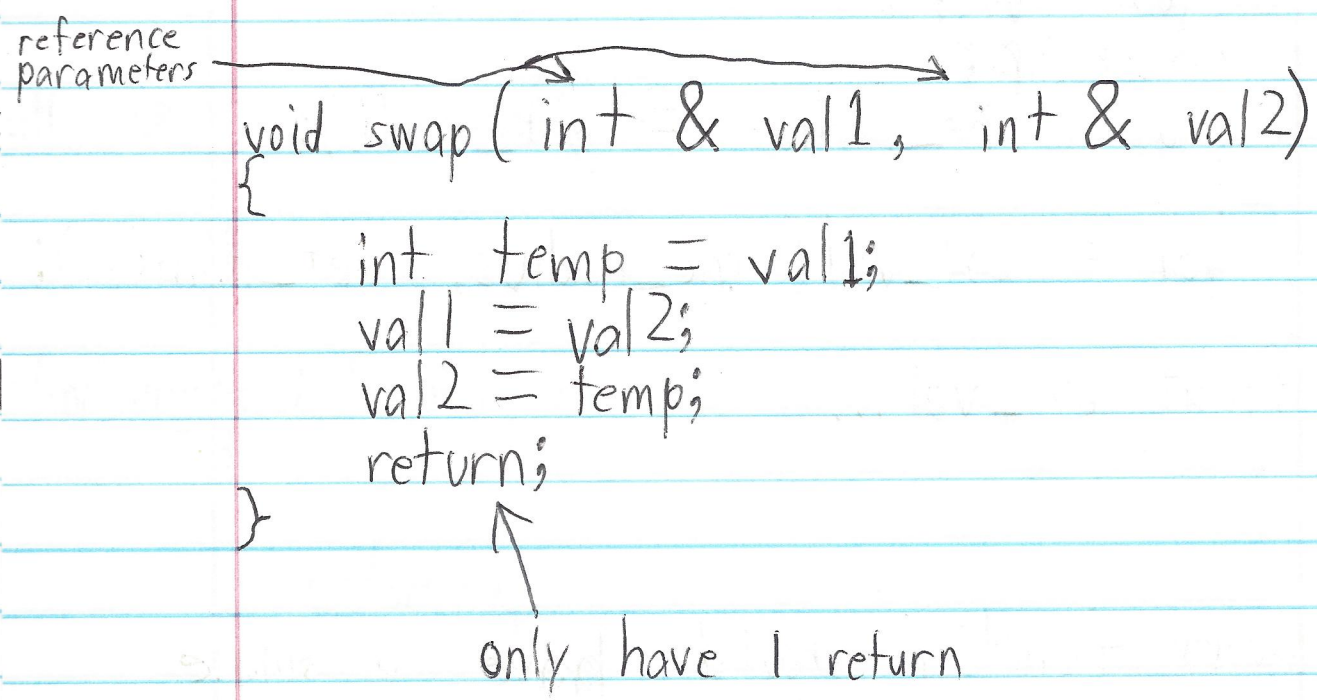
Stack — FILO or LIFO

Queue — FIFO

Run-time Stack



Pass-by-Refence



Code Documentation

Requires Good English

Your code will have (for functions)
a set of 3 comments
1. function description — what (NOT how) a function does
2. precondition — what must be true of the inputs to gaurantee the postcondition
3. postcondition — a statement of what is true after successful execution

comment prototypes

```
//Description: The greeting function outputs a message to the screen
//Precondition: none.
//Postcondition: A message is displayed to screen.
void greeting();
```

```
// Description: The swap() function will
// swap the two values passed in the
// calling function.
// Precondition: Both parameters must be
// modifiable memory locations.
//
```

```
// Description: The vol_cyl() function
// calculates and returns
```

Separate Files

header .h
main .cpp
implementation .cpp

in header
1. Global Constants
2. Prototypes
3. Structs
4. Classes
5. Templates

names
[name].h
[name].cpp
[name]_functs.cpp

in .h
#ifndef ~~MY~~ [NAME]_H
#define [NAME]_H
[code]
#endif

all caps [name]

in main and implementation
#include "[header file name]"

#include <iostream>
using namespace std;

to compile, in directory
fg++ *.cpp -o [output file]

Function Overloading

Creating more than one function with the same name. The parameter list must be different.

compiles, runs

```
float f();

f();
```

```
float ave(const float f1, const float f2);
float ave(const float f1, const float f2, const float f3);
```

Static Variables

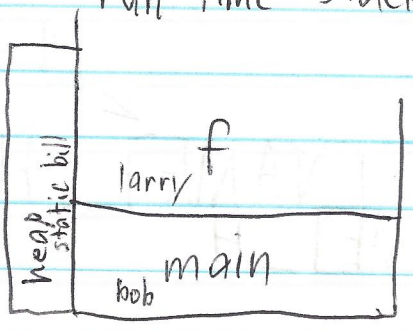
if not initialized compiler initializes to 0

syntax

[static [type] [identifier] [optional initialization];

The declaration is executed only once

If initialized in a function, its value is not destroyed when the function is exited.



templates

Rules

1. Template code must be in header file
2. typename must be in parameter list

```
void swap(int & i1, int & i2)
{
  int temp = i1;
  i1 ^= i2 ^= i1 ^= i2;
  return;
}
void swap(double & i1, double & i2);

template <typename T>
void swap(T & t1, T & t2)
{
  [swap code]
  return;
}
}}}}

swap([string], [string]); // uses template
```

```
template<class T, class U>
void nstuff(const T t, const U u, const int n)
{
  [stuff]
  return;
}
```

With templates, ~~menti~~ mention ~~undefin~~ potentially undefined operators in precondition

```
//Precondition: The << operator must be defined
// for T and U
```

Default parameters

```
void f(int x = 1, char y = 'c');

f(); //compiles
```

structs — OOP

```
struct pile
{
  short sector;
  short size;
};
pile p1;
```
← junk

structs are types

~~struct [name]~~

```
struct pile
{
  short sector = 0;
  short size = 0;
};
pile p1;
```
← zeros

"definition" or "declaration" in header file

```
struct [name]
{
   [type] [name];
   :
};
```
← don't forget

```
struct point
{
  float m_Xcoord;
  float m_Ycoord;
};
```
← m stands for "member" recommended, not required

```
point bob;

bob.m_Xcoord = 5;
cout << bob.m_Xcoord << endl;
```
← proper

```
void print_pt( const point & a_pt)
{
  cout << "(" << a_pt.m_Xcoord << "," << a_pt.m_Ycoord << ")";
  return;
}
```

```
point frank = {1, 2};  // works
```

~~forward declaration~~ doesn't work

```
struct point;

struct line
{
   point left;
   point right;
};
```

[declaration of point]

put comments in implementation file (just once)

Operator Overloading

arity - unary, binary, or ternary
number of operands

```
bool operator<(const tomato t1, const tomato t2)
{
   return t1.wt < t2.wt;
}
```

prototype with struct

NOT in struct

define in implementation file

```
[return type] operator[symbol]([parameter(s)])
{
   [code ending with return statement]
}
```

Arrays

syntax

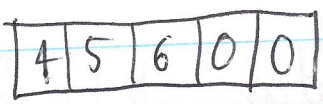
[type] [name] [[integer known at compile time (const or literal)]]; ← junk

use const

[type] [name] [[const or literal]] = { [initial value] }; ← 0 0 .. 0 initialized

[short arr[5] = {4, 5, 6};

creates



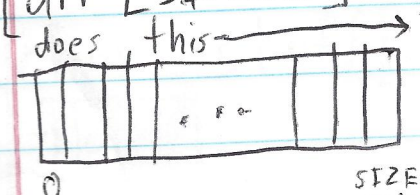
[short arr[] = {4, 5, 6};

creates

"walking off the array"

[arr[SIZE] = 4;

does this →



may or may not throw "seg fault" error

may crash OS

Array parameters

arrays can be passed but not returned

void f([type] [name][], [const] [integer type] dataSize);

↑ array reference

do not put reference (&) symbol on arrays

in hw7 use bubble sort

test on Friday Oct 28

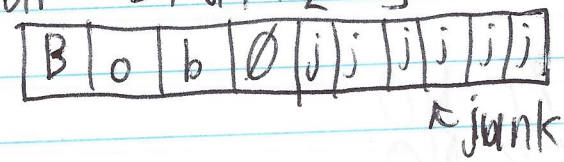
Strings

null-terminated character arrays (NTCAs)
aka cstrings and c-style strings

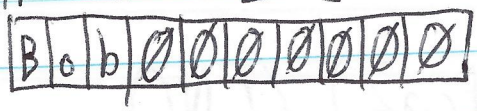
An array of characters that has its data terminated by a null character ('\0')

```
char stuff[10]; //NOT NTCA
```

```
char stuff[10] = "Bob"; // NTCA
```

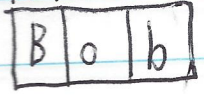



```
char stuff[10] = {'B', 'o', 'b'}; //NTCA
```




```
char stuff[] = "Bob"; //NTCA
```


```
char stuff[] = {'B', 'o', 'b'}; //NOT NTCA
```



too small

initialization `char stuff[10]; //NOT NTCA`

```
cin >> stuff; // makes stuff an NTCA
              // only reads to first whitespace
```

BillyBobbbb ↵
walks off array

```
cout << stuff; // works
```

```
int arr[10];
cin >> arr; // NO
cout << arr; // NO
```

```
if(ntca1 == ntca2) // NO
```

assignment `ntca1 = "ASDF"; // ABSOLUTELY NOT!!!`

You cannot assign an array

Built-in functions with NTCAs

#include <cstring>

length: int strlen([ntca]);

copy: void strcpy([target NTCA], [const source NTCA]);

concatenate: void strcat([target NTCA], [const source NTCA]);

compare: int strcmp([NTCA1], [NTCA2]);

{ if NTCA1 > NTCA2 strcmp() returns positive lexicographic difference
{ if NTCA1 == NTCA2 strcmp() returns 0
{ if NTCA1 < NTCA2 strcmp() returns negative lexicographic difference

strncpy([target NTCA], [source NTCA], [# of chars to cpy]);

re: strncat([target NTCA], [source NTCA], [# of chars from source to cat onto target]);

strncmp([NTCA1], [NTCA2], [# of chars to cmp to]);

strcpy(ntca, "bob"); // works

#include <string>

cin >> name;

name.length()

>> extraction

<< insertion

cin and cout are ~~variables~~ objects.
"stream" objects

for strings getline([input stream (cin)], [string variable], [delimiting char ('\n')])

[input stream (cin)].getline([char array], [max number of chars to read less than or equal to array length − 1])

>>
skips leading white space
leaves (↵) delimiter
getline will NOT skip leading white space
and will discard the delimiter

discards chars on the istream [(cin) stream].ignore([max number of chars to pitch], [delimiting char])

gets one char [(cin) stream].get([char reference var])
[(cin) stream].putback([char reference var])

#include <cctype>

bool isupper([char])
islower
ispunct
isspace
isalpha
isdigit

void toupper([char &])
tolower

1. char by char using get()
2. word by word using extraction
3. line by line using getline()

```
cout << "Prompt: ";
cin >> str;
cout << "Another prompt: ";
~~getline~~
cin.getline(ntca, 29); // doesn't work
```

```
cout << "Prompt: ";
cin >> str;
cout << "Another prompt: ";
~~cin >>~~
cin.ignore(29, '\n');
cin.getline(ntca, 29);
```

File I/O

```
#include <fstream>

ifstream fin;
ofstream fout;
```

```
ifstream fin("[name].dat");
```

or

```
fin.open("[name].dat");
```

```
string fname = "[name].dat";
fin.open(fname.c_str());
```

```
do
{
   fin.clear();
   cout << "enter name of file: ";
   cin >> fname;
   fin.open(fname.c_str());
}while(!fin);
fin >> var;
fin.close();
```

```
string wh;
ifstream in("in.dat");
ofstream out("out.dat");
while(getline(in, wh))
{
   out << wh << endl;
}
in.close();
out.close();
```

73345 — CS1001

don't do this

```
while(fin >> var)
  [process]
```

2-part data

```
while(fin >> var)
{
  fin >> var2;
  [process]
}
```

```
fin >> var;
while(!fin.eof())
{
  [process]
  fin >> var;
}
```

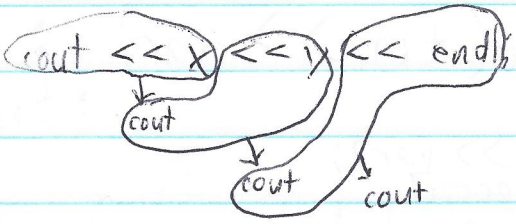
Passing streams to functions

Rule: when passing or returning a stream it must be a rerence

```
template <typename T_filetype>
void fileopen(T_filetype & file, const string
  ↳ prompt)
{
  string filename;
  short counter = 0;
  do
  {
    cout << "enter the name of the " << prompt <<
      "file to connect: ";
    cin >> filename;
    counter++;
    file.open(filename.c_str());
    if (!file)
      cout << "Try again, ya loser! ";
    if (counter == MAX_TRIES)
    {
      cout << "Jeez...";
      exit(162);
    }
  } while (counter < MAX_TRIES && !file);
  return;
```

overloading the insertion operator

```
ostream& operator<<( ostream& out, const [your type]& [name])
{
  out << [format];
  return out;
}
```



you don't have to use return values

```
istream& operator >> ( istream& in, [type]& [name] )
{
  in >> ... or in.getline(...
  return in;
}
```

OOP

a class encapsulates variables (data) ~~and data~~ and functions (behavior)

member variables reside in the private section
normally, member functions reside in the public section. Public member functs dictate the only way private member variables can be manipulated

every class gets 2 files, a header for the class declaration and a cpp for the member function definitions

```
class [name]
{
  public:
    [member functions]
  private:
    [member variables
    member functions]
};
```

fraction.h

tells compilers not to allow this function to change the calling object

```
class fraction
{
public:
  void readin();
  fraction reciprocal();
  int getNum() const;

private:
  int m_Num;
  int m_Den;
};
```

use m_

fraction.cpp

scope resolution operator :: (2 colons)

```
void fraction::readin()
{
  [readin code]
  return;
}

fraction fraction::reciprocal()
{
  fraction temp;
  temp.m_Num = m_Den;
  temp.m_Den = m_Num;
  return temp;
}
```

access to private in member function

```
fraction multfracs(const fraction& lhs, const fraction& rhs)
{
  fraction temp;
  temp.setNum(lhs.getNum() * rhs.getNum());
  :
```

```
public:
    void setNum(const int n);

    int getDen()const { return m_Den; }
```

← in-line definition

```
void fraction::setNum(const int n)
{
  m_Num = n;
  return;
}
```

```
class fraction
{
  public:
    void readin();
    void print()const;
    {
    friend fraction multfracs(const fraction & lhs, const fraction & rhs);
}
```

don't use friend functions unless you have to

```
fraction multfracs(
```

friend functs cannot be const

Constructors

1. automatically called
2. named the name of the class
3. have no return type and no return statement
4. often overloaded
5. C++ will automatically write the defaut constructor
6. The default is supressed if you write any constructor

in .h

```
class fraction
{
  public:
    fraction(const int n, const int d);
    fraction(): m_Num(0), m_Den(1){}  // initialization list overloads default constructor
```

in .cpp

```
fraction::fraction(const int n, const int d)
{
  m_Num = n;
  m_Den = d;
}
```

```
fraction f(1, 2);
```

```
class fraction
{
  public:
    fraction(const int n = 0, const int d = 1):
      m_Num(n) { ~~m_Den(d);~~ setDen(d);}
```

another initialization list in .cpp

```
fraction::fraction(): m_Num(0), m_Den(1) {}
```

Copy Constructors

Automatically Defined

```
class fraction
{
 public:
   fraction(const fraction & source);
```

very important

```
fraction::fraction(const fraction & source)
{
  m_Num = source.m_Num;
  m_Den = source.m_Den;
}
```

~~Operator Overloading~~

2D Array

[type] [name] [[size 1]][[size 2]]

size 1: number of rows

size 2: number of columns

```
class town
{
 private:
   char ~~grid~~ m_Grid[MAX][MAX];
   int ~~size;~~ m_size;
```

```
void f(char q[][size], ...)
```

all dimensions after the 1st must have sizes

Operator Overloading

syntax

1. You can't create new operators
2. You can't change the precedence
3. You can't change the arity
4. You can't overload :: . .* sizeof ?:

1. Make definitions sensible
2. Define / pairs one in terms of the other
3. Decide carefully to make an operator member or non-member
   - If an operator does not change an object, make it a non-member; if an operator changes an object, make it a member

Make non-members freinds

"this" is a pointer to the calling object
*this is the calling object

```
friend ostream& operator<<( ostream& o,
                            const fraction & f);

fraction f, g, h;

h= ~f;
```

```
class fraction
{
  public:
    friend fraction operator~( const fraction & f);
    fraction operator-() const;
};

fraction operator~( const fraction & f)
{
  fraction r;
  r.m_Num = f.m_Den;
  r.m_Den = f.m_Num; r.setDen( f.m_Num);
  return r;
}

fraction fraction::operator-() const
{
  fraction r;
  r.m_Num = -m_Num;
  r.m_Den = m_Den;
  return r;
}
```

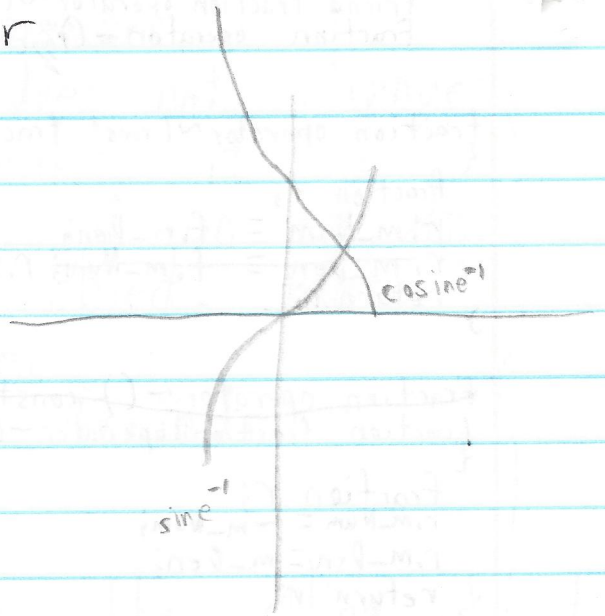
```
class fraction
{
public:
friend fraction operator*(const fraction & lhs, const fraction & rhs);
fraction & operator*=(const fraction & rhs);
friend istream& operator>>(istream& in, fraction & f);

friend fraction operator*(const fraction &lhs, const fraction & rhs)
{
  fraction result(lhs);
  return result *= rhs;
}

fraction& fraction::operator*=(const fraction & rhs)
{
  m_Num *= rhs.m_Num;
  m_Den *= rhs.m_Den;
  return *this;
}

istream & operator>>( istream & in, fraction & f)
{
  in >> f.m_Num >> f.m_Den >>; // needs form [num] [den]
  return in;
}
```

c++ will automatically write
1. default constructor
2. copy constructor
3. operator=



More Operators

the special operators —
must be overloaded as members

= [] ()

"assignment" "projection" "function evaluation"

1. = (copy assignment)

C++ automatically provides this operator

the auto assignment works as long as the class contains no pointers

with pointers you must overload = operator, copy constructor

```
A a, b;
a = b; // copy assignment
B g;
a = g; // works if you define it (NOT copy assignment)
```

```
class fraction
{
  public:
    fraction& operator=(const fraction & rhs);
};

fraction& fraction::operator=(const fraction& rhs)
{
  m_Num = rhs.m_Num;
  m_Den = rhs.m_Den;
  return *this;
}
```

2. []

for the [] operator, you can only have ONE parameter

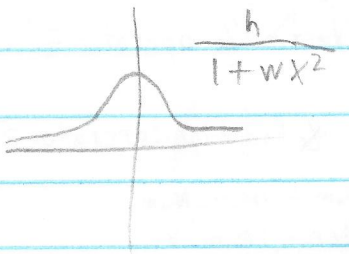
It is usually overloaded for array-type objects, used for indexing

```
int operator [](const int i) const;
int Fraction::operator[](const int i) const
{
    return (i==0? m_Num : m_Den);
}
```

~~bob = f; // no~~
~~cout <<f;~~

```
int& operator [](const int i);
int& fraction::operator[](const int i)
{
```

3. ()




```
class bellcurve
{
    float m_Height;
    float m_Width;
    public:
        bellcurve(float h, float w): m_Height(h), m_Width(w) {}
        float operator()(const float x) const;

float bellcurve::operator()(const float x) const
{
    return m_Height / (1 + m_Width * x * x);
}

bellcurve Ari(1, 2);
cout << Ari(3);  → 0.051
```

static members

fraction.h (completely contrived)

```
class fraction
{
  public:
    static float zero_tolerance;
    ...
};
```

static members live on the heap, not the run-time stack

fraction.cpp

```
float fraction::zero_tolerance = 0.005;
...
if(static_cast<double>(f.getNum()) / f.getDen()
    < fraction::zero_tolerance)
  f.setNum(0);
```

```
public:
  static void setTol(const float t){ fraction::zero_tolerance = t; }
```

```
class stooge
{
  public:
    static void moe();
    static void larry();
    static void curly();
};

stooge::curly();
```

```
class A
{
  public:
    static const int x=4;
    static const float y=2.8; //you can't
                              // have a
                              // static
                              // const float
```

```
class A
{
private:
  int & y;
  const int z; // these MUST be constructed
               // in an initialization list
```

Template Classes

stack.h

```
template<class T>
class stack
{
  private:
    T data[MAX]; m_data[MAX];
    int m_size;
  public:
    stack() : m_size(0) { }
    bool push( const int T to_push);
    bool isFull() const { return m_size == MAX; }
    bool pop( int T& stuff );
    bool isEmpty() const { return !m_size; }
    stack( const stack<T>& source);
};
```

before template:

```
stack a(); //WRONG
stack a;
a.push(14);
a.push(2);
a.push(13);
```

stack.hpp

```
#include "stack.hpp"
template<class T>
bool stack<T>::push( const int T to_push)
{
  bool pusht = false;
  if( !isFull() )
  {
    m_data[m_size] = to_push;
    m_size++;
    pusht = true;
  }
  return pusht;
}
template<class T>
bool stack<T>::pop( int T& stuff)
{
  bool popt = false;
  if( !isEmpty() )
  {
    stuff = m_data[m_size - 1];
    m_size--;
    popt = true;
  }
  return popt;
}
```

with template:

```
stack<float> b;
```

```
template<class T>
stack<T>::stack( const stack<T>&s)
{
```

Test Review

1. ofstream fout
2. fout.open("stuff.dat");
3. No, because it only processes every other datum it extracts from fin
4. foo() cannot alter the calling object
5.
6. class book
```
{
  private:
    string m_title;
    int m_pages;
    int m_pics;
  public:
    book();
    book(const book & b);
    string get_title() const;
    void set_title(const string title);
};
```
7.
```
string book::get_title() const
{
  return m_title;
}
void book::set_title(const string title)
{
  m_title = title;
  return;
}
```
8.
```
book::book(const book & b)
{
  m_title = b.m_title;
  m_pages = b.m_pages;
  m_pics = b.m_pics;
}
```

9. bool operator<(const book & rhs) const;

```
bool book::operator<(const book & rhs) const
{
return static_cast<float>(
```

10. friend ostream& operator<<(ostream& o,
const book& b);

```
ostream& ...
{
  o << b.m_title << " Pages:"
    << b.m_pages << ...
  return o;
}
```

11.
```
template < class T_item_type >
class store
{
  private:
    T_item_type m_items[MAX];
    int m_num;
  public:
    store(): m_num(0) {}
};
```

12. store<book> book_store;

13. T/F

```
// This file demos enums
#include <iostream>
#include <ctime>

using namespace std;

int main()
{
   srand(time(0));                                    // seed generator
   enum coin_toss {head,tail};                        // define enumerated type
   string tosses[2] = {"head", "tail"};               // an array of the enum nar
                                                      //   so that we can output

   coin_toss flip;
                                                      //   those names.
   short count_heads = 0;                             // counters for num tails
   short count_tails = 0;                             //   and heads
   for (int i=1;i<=200;i++)
   {
     flip = coin_toss(rand()%2);                      // randomly give flip value
     cout<<"We tossed a "<<tosses[flip]<<endl;
     flip ? count_tails++ : count_heads++ ;           // counting heads and tails
   }
   cout<<endl<<"We tossed:      "<<count_heads<<" heads"<<endl
             <<"                "<<count_tails<<" tails"
             <<endl<<"....exiting"<<endl;


   return 0;
}

/*
    .
    .
    .
We tossed a head
We tossed a tail
We tossed a tail
We tossed a tail
We tossed a tail
We tossed a tail
We tossed a head
We tossed a tail
We tossed a head
We tossed a head
We tossed a tail
We tossed a tail
We tossed a tail
We tossed a head
We tossed a head
We tossed a tail
We tossed a head

We tossed:      110 heads
                90 tails
                ....exiting

*/
```

```
#include <unistd>
sleep([num seconds]);
usleep([num microseconds]);
```

enumeration

Armita (section < teacher)
aan87@mst.edu

in header file
~~top of main~~

capitalize → constants

```
enum color
{
    Red, //0
    Black, //1
    Yellow, //2
    Orange //3
};
```

```
color c;
c = Red;
cout<<c; // 0
c=3; // NO "cannot convert from integer to color"
```

```
enum color
{
  Red = -3,
  Black, //-2
  Yellow =5,
  Orange =5,
  Blue //6
};
```

```
enum ParseResult
{
  Success,
  ErrorOpening,
  ErrorReading,
  ErrorParsing
};
~~int readFile()~~ ParseResult readFile()
{
  if(!openFile())
    return ErrorOpening;
  else if(!readFile())
    return ErrorReading;
  else if(!parseFile())
    return ErrorParsing;
  ~~re~~
  else return Success;
}
```

```
int main()
{
  if(readFile() == success)
```

Namespaces

in foo.h
```
int dosomething( int x, int y)
{
  return x+y;
}
```

in goo.h
```
int dosomething( int x, int y)
{
  return x-y;
}
```

```
#include "foo.h"
#include "goo.h"
int main()
{
   dosomething( 5, 7); // doesn't compile
```

solution

in foo.h
```
namespace FOO
{
   int dosomething( int x, int y)
   {
     return x+y;
   }
}
```

in goo.h
```
namespace GOO
{
   :
```

then you can say
~~un~~ using namespace FOO;

or

```
FOO::dosomething(5, 7);
GOO:: dosomething( 11, 12);
```

```
namespace H1
{
  void myFunc()
  {
    :
  }
}

namespace H2
{
  void myFunc()
  {
    :
  }
}

namespace H1
{
  void Func2()
  {
    :
  }
}
```

in main file

```
#include
int main()
{
  H1::myFunc();
  H2::myFunc();
```

once you add (using namespace ...;) a namespace, you cannot undo it.
but namespaces are limited by scope

can you put method definitions in class definition?

does eof() only return false when the end of the file is reached

Command Line Arguments

← standard →

```
int main( int argC, char * argV[] )
```

~~by default, argC = 1~~

~~argC is the length of argV~~

./test

argV: [test]

array length 1

test is argV[0]

TW in.dat

argV[0] argV[1]

argC = 2

TW

argV[0]

argC = 1

~~if~~ if(argC = 3)

cout << argV[0]

```
#include <cstdlib>
```

str → int atoi([ntca])

str → float atof([ntca])

str → long atol([ntca])

Vectors

```
#include <vector>

vector<int> vec1;
  cout << vec1.size(); //0

vector<float> vec2(10); // initial size

int size;
cin >> size;
vector<char> vec3(size);

vec1.resize(5); // new size

vec1[0] = 15; // .at() does the same thing as []
vec1.at(1) = 10;

vec2.push_back(12); // increases length and puts in parameter size by 1

vec2.pop_back(); // decreases size by 1 and returns the lost element
vec2.front(); // first element lowest index element
vec2.back(); // highest index element

while( !myVector.empty() )
{
  sum += myVector.back();
  myVector.pop_back();
}
```

Output Formatting

Default 6 sigfig

```
double x = 1234.56789;
cout << x;   1234.57
```

```
cout.precision(3);
cout << x;   1230
```

```
cout.width(10);  // lasts only for next output
```

ios flags

```
cout.setf( ios::<flag> );
```

```
cout.setf( ios::showpt );
```

scientific
fixed
left
right
showpos
showpt
internal

Tuesday 3:00 — 5:00 BCH 125

~~mandatory~~

optional

if you take it, ~~se~~ email price

"quicksort2" by TW 1/15/2017 SDRIVE/cs1570/hw1/quicksortTest

```
void quicksort( T arr[], const long min, const long max )
{
  if( max > min )
  {
    long pivotIndex = (max - min) /2 + min;
    T temp = arr[pivotIndex];
    long high = max;
    long low = min;
    while( high > low)
    {
      if( pivotIndex < high)
      {
        if( arr[high] < temp )
        {
          arr[pivotIndex] = arr[high];
          pivotIndex++;
          arr[high] = arr[pivotIndex];
        }
        else
          high--;
      }
      if( pivotIndex > low )
      {
        if(arr[low] > temp)
        {
          arr[pivotIndex] = arr[low];
          pivotIndex--;
          arr[low] = arr[pivotIndex];
        }
        else
          low++;
      }
    }
    arr[pivotIndex] = temp;
    quicksort( arr, min, pivotIndex - 1 );
    quicksort( arr, pivotIndex + 1, max );
  }
  return;
}
```