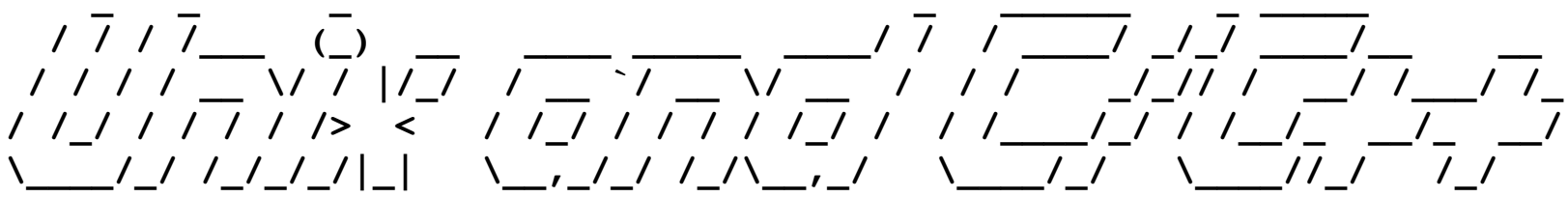
UNIVERSITY OF OREGON

CIS 330:



Lecture 18: templates and STL

May 27th, 2016

Hank Childs, University of Oregon

Announcements

- Weekend OH?
- Extra Credit
- 3H, 4B, 4C posted

Announcements: Rest of Term

- 3G: assigned Monday (two days ago), due Monday (5 days from now)
- 3H, 4B, 4C: assigned today, nominally due on Friday June 3rd
  - But: you can take until the Final to get them (3H, 4A, 4B) wrapped up (& they will not be late)

3H

- Will look at this together at the end of lecture

UNIVERSIT

Assignment:
Make your 3E project run memory error and leak free

4E

Tasks:

1) start with your 3E source code. (Don't use 3F, since it has exceptions and

Notes:

1) if I had memory errors, they would have occurred after "Command: proj3F" and before "HEAP SUMMARY". None there, so I'm OK.
   a. Don't forget to use your 3E code ... not 3F code.
2) Re memory leaks: "All heap blocks were freed" is the magic statement ... that means no memory leaks.

ed to use ix to do this ... esp.

clares your program leak free

Submit:

1) a screenshot of the valgrind output (see mine below)
2) your source code

```
hank@ix: ~/3F/3F 65$ valgrind proj3F ~/3A_input.pnm 3F_output.pnm
==16125== Memcheck, a memory error detector
==16125== Copyright (C) 2002-2011, and GNU GPL'd, by Julian Seward et al.
==16125== Using Valgrind-3.7.0 and LibVEX; rerun with -h for copyright info
==16125== Command: proj3F /home/users/hank/3A_input.pnm 3F_output.pnm
==16125==
==16125==
==16125== HEAP SUMMARY:
==16125==     in use at exit: 0 bytes in 0 blocks
==16125==   total heap usage: 33 allocs, 33 frees, 108,022,422 bytes allocated
==16125==
==16125== All heap blocks were freed -- no leaks are possible
==16125==
==16125== For counts of detected and suppressed errors, rerun with: -v
==16125== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 2 from 2)
```

4C

1) Extend your project 3 to having timing information
    a. The timings should be added to logger
    b. The timing for each Execute method should be logged individually
        i. Note: if you do the inheritance stuff cleverly, you should only have to add two gettimeofday calls total
2) Run a small performance study using your time
    a. Create the following pipeline: one source that creates a solid red image, one source that creates a solid blue image, and one Blender (who's inputs are the red image and blue image).
    b. Run the program multiple times. The first time make the solid colored images be 250x250. Then 500x500. Then 1000x1000. Then 2000x2000. Study the timings.
    c. Write a short report (i.e., several sentences) about your findings. Please do not use MS Word or RTF. This short report should be in a text file, so that I can easily view it with my "vi" program.

Live Code: Weds June 1st

- Disclaimer: don't be discouraged
- Will go thru ~3E, definitely not past 3F
- Who can attend?
  - Anyone who has completed the project
  - Anyone who has not completed the project, but is getting 50% credit
- Who can not attend?
  - Anyone who has extenuating circumstances and expects full credit for late work

Review

gettimeofday

GETTIMEOFDAY(2) BSD System Calls Manual GETTIMEOFDAY(2)

NAME

gettimeofday, settimeofday -- get/set date and time

SYNOPSIS

```
#include <sys/time.h>

int
gettimeofday(struct timeval *restrict tp, void *restrict tzp);

int
settimeofday(const struct timeval *tp, const struct timezone *tzp);
```

DESCRIPTION

The system's notion of the current Greenwich time and the current time zone is obtained with the **gettimeofday**() call, and set with the **settimeofday**() call. The time is expressed in seconds and microseconds since midnight (0 hour), January 1, 1970. The resolution of the system clock is hardware dependent, and the time may be updated continuously or in ``ticks.'' If tp is NULL and tzp is non-NULL, **gettimeofday**() will populate the timezone struct in tzp. If tp is non-NULL and tzp is NULL, then only the timeval struct in tp is populated. If both tp and tzp are NULL, nothing is returned.

The structures pointed to by tp and tzp are defined in <sys/time.h> as:

```
struct timeval {
        time_t       tv_sec;   /* seconds since Jan. 1, 1970 */
        suseconds_t  tv_usec;  /* and microseconds */
};

struct timezone {
        int     tz_minuteswest; /* of Greenwich */
        int     tz_dsttime;     /* type of dst correction to apply */
};
```

The timeval structure specifies a time value in seconds and microseconds. The values in timeval are opaque types whose length may vary on different machines; depending on

The timezone structure indicates om Greenwich), and a flag that, if nonzero, indicates that Daylight Saving time a

Only the super-user may set the t er than 1 (see init(8)), the time may only be advanced. This limitation is imp y time stamps on files. The system time can still be adjusted backwards using

(there are lots of Unix system calls, which do lots of different things)

RETURN

A 0 return value indicates that the call succeeded. A -1 return value indicates an error occurred, and in this case an error code is stored into the global variable errno.

gettimeofday example

```
fawcett:330 childs$ cat timings.C
#include <sys/time.h>
#include <stdio.h>

int main()
{
    int num_iterations = 100000000;
    int count = 0;
    struct timeval startTime;
    gettimeofday(&startTime, 0);
    for (int i = 0 ; i < num_iterations ; i++)
        count += i;
    struct timeval endTime;
    gettimeofday(&endTime, 0);
    double seconds = double(endTime.tv_sec - startTime.tv_sec) +
                     double(endTime.tv_usec - startTime.tv_usec) / 1000000.;
    printf("done executing, took %f\n", seconds);
}
```

gettimeofday example

```
fawcett:330 childs$ cat timings.C
#include <sys/time.h>
#include <stdio.h>

int main()
{
    int num_iterations = 100000000;
    int count = 0;
    struct timeval startTime;
    gettimeofday(&startTime, 0);
    for (int i = 0 ; i < num_iterations ; i++)
        count += i;
    struct timeval endTime;
    gettimeofday(&endTime, 0);
    double seconds = double(endTime.tv_sec - startTime.tv_sec) +
                     double(endTime.tv_usec - startTime.tv_usec) / 1000000.;
    printf("done executing, took %f\n", seconds);
}
fawcett:330 childs$ g++ -O2 timings.C
fawcett:330 childs$ ./a.out
done executing, took 0.000000
fawcett:330 childs$
```

gettimeofday example

```
fawcett:330 childs$ cat timings.C
#include <sys/time.h>
#include <stdio.h>

int main()
{
    int num_iterations = 100000000;
    int count = 0;
    struct timeval startTime;
    gettimeofday(&startTime, 0);
    for (int i = 0 ; i < num_iterations ; i++)
        count += i;
    printf("Count was %d\n", count);  /* NEW LINE OF CODE */
    struct timeval endTime;
    gettimeofday(&endTime, 0);
    double seconds = double(endTime.tv_sec - startTime.tv_sec) +
                     double(endTime.tv_usec - startTime.tv_usec) / 1000000.;
    printf("done executing, took %f\n", seconds);
}
```

gettimeofday example

```
fawcett:330 childs$ cat timings2.C
#include <sys/time.h>
#include <stdio.h>

int LoopFunction(int iteration, int &count)
{
    count += iteration;
}

int main()
{
    int num_iterations = 100000000;
    int count = 0;
    struct timeval startTime;
    gettimeofday(&startTime, 0);
    for (int i = 0 ; i < num_iterations ; i++)
        LoopFunction(i, count);
    /* No longer need this: printf("Count was %d\n", count);  */
    struct timeval endTime;
    gettimeofday(&endTime, 0);
    double seconds = double(endTime.tv_sec - startTime.tv_sec) +
                     double(endTime.tv_usec - startTime.tv_usec) / 1000000.;
    printf("done executing, took %f\n", seconds);
}
fawcett:330 childs$ g++ -O2 timings2.C
fawcett:330 childs$ ./a.out
done executing, took 0.213101
```

Debugging

Pitfall #2: millions of print statements

```
void make_black(unsigned char *b, int width, int height,
                int buffer_size)
{
    for (int i = 0 ; i < width ; i++)
    {
        for (int j = 0 ; j < height ; j++)
        {
            int pixel_index  = j*width+i;
            int buffer_index = 3*pixel_index;
            cerr << "About to write to index"
                 << buffer_index << endl;
            b[buffer_index+0] = 0;
            b[buffer_index+1] = 0;
            b[buffer_index+2] = 0;
        }
    }
```

This will result in millions of print statements … hard to debug.

Reducing print statements

```
void make_black(unsigned char *b, int width, int height,
                int buffer_size)
{
    for (int i = 0 ; i < width ; i++)
    {
        for (int j = 0 ; j < height ; j++)
        {
            int pixel_index  = j*width+i;
            int buffer_index = 3*pixel_index;
            if (buffer_index < 0 || buffer_index >= buffer_size)
            {
                cerr << "About to write to index"
                     << buffer_index << endl;
                exit(EXIT_FAILURE);
            }
            b[buffer_index+0] = 0;
            b[buffer_index+1] = 0;
            b[buffer_index+2] = 0;
        }
    }
}
```

Debugging: Debuggers

Debuggers

- Allow you to set breakpoints
- Allow you to inspect variables
- Show you where a program crashed

Debuggers

- gdb:
  - GNU debugger
  - goes with gcc/g++
- lldb:
  - CLANG debugger

Debugging Symbols

- Object files:
  - by default, are compact and contain minimal information that connects to your original program
  - optionally, can be instructed to contain increased linkage
    - what line of code did these instructions come from?
    - what variable name is in this register?

You enable debugging symbols by adding "-g" to compile line "gcc –c file.C" → "gcc –g –c file.C"

Running with gdb

```
hank@ix: ~ 7$ cat myprogram.C
#include <stdlib.h>
int main()
{
    int *p = NULL;
    int X = *p;
}
hank@ix: ~ 8$ g++ -g myprogram.C
hank@ix: ~ 9$ gdb a.out
```

Running with gdb

```
(gdb) run
Starting program: /home/users/hank/a.out

Program received signal SIGSEGV, Segmentation fault.
0x0000000004004c4 in main () at myprogram.C:5
5               int X = *p;
(gdb) where
#0  0x0000000004004c4 in main () at myprogram.C:5
(gdb)
```

Arguments

- You are running "./proj3A 3A_input.pnm 3A_output.pnm"
- In gdb, you would do:

  % gdb proj3A

  (gdb) run 3A_input.pnm 3A_output.pnm

"core" files

- When a program crashes, it can create a "core" file
  - This file contains the state of memory when it crashes
  - It is very large, and can fill up filesystems
    - So system administrators often disable its generation
      - "ulimit –c 0" → "ulimit –c 10000000"
  - You can run a debugger on a program using a core file (tells you where it crashed)
    - gdb a.out core

Valgrind: a memory checker

```
hank@ix: ~ 14$ valgrind a.out
==13623== Memcheck, a memory error detector
==13623== Copyright (C) 2002-2011, and GNU GPL'd, by Julian Seward et al.
==13623== Using Valgrind-3.7.0 and LibVEX; rerun with -h for copyright info
==13623== Command: a.out
==13623==
==13623== Invalid read of size 4
==13623==    at 0x4004C4: main (myprogram.C:5)
==13623==  Address 0x0 is not stack'd, malloc'd or (recently) free'd
==13623==
==13623==
==13623== Process terminating with default action of signal 11 (SIGSEGV)
==13623==  Access not within mapped region at address 0x0
==13623==    at 0x4004C4: main (myprogram.C:5)
==13623==  If you believe this happened as a result of a stack
==13623==  overflow in your program's main thread (unlikely but
==13623==  possible), you can try to increase the size of the
==13623==  main thread stack using the --main-stacksize= flag.
==13623==  The main thread stack size used in this run was 8388608.
==13623==
==13623== HEAP SUMMARY:
==13623==     in use at exit: 0 bytes in 0 blocks
==13623==   total heap usage: 0 allocs, 0 frees, 0 bytes allocated
==13623==
==13623== All heap blocks were freed -- no leaks are possible
==13623==
==13623== For counts of detected and suppressed errors, rerun with: -v
==13623== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 2 from 2)
Segmentation fault (core dumped)
```

Valgrind and GDB

- Valgrind and gdb are available on ix
  - Older versions of Mac are possible, newer are not
  - Linux is easy
- You will have an assignment to have a memory error-free and memory leak-free program with valgrind.

Templates

Motivation

```
int   Doubler(int X) { return 2*X; };
float Doubler(float X) { return 2*X; };


int main()
{
    int   X = 2;
    float Y = 2.6;
    cout << "2*X = " << Doubler(X) << ", 2*Y = " << Doubler(Y) << endl;
}
fawcett:330 childs$ g++ logger_defines.C
fawcett:330 childs$ ./a.out
2*X = 4, 2*Y = 5.2
```

```
fawcett:330 childs$ nm a.out
0000000100000d7a s  stub helpers
00000001000010b0 D _NXArgc
00000001000010b8 D _NXArgv
0000000100000ac7 t __GLOBAL__I__Z7Doubleri
0000000100000a84 t __Z41__static_initialization_and_destruction_0ii
0000000100000b26 T __Z7Doublerf
0000000100000b18 T __Z7Doubleri
```

Motivation

```
fawcett:330 childs$ nm a.out
0000000100000d7a s  stub helpers
00000001000010b0 D _NXArgc
00000001000010b8 D _NXArgv
0000000100000ac7 t __GLOBAL__I__Z7Doubleri
0000000100000a84 t __Z41__static_initialization_and_destruction_0ii
0000000100000b26 T __Z7Doublerf
0000000100000b18 T __Z7Doubleri
```

```
DOUBLER_MACRO(int)
DOUBLER_MACRO(float)


int main()
{
    int   X = 2;
    float Y = 2.6;
    cout << "2*X = " << Doubler(
}
fawcett:330 childs$ g++ logger_d
fawcett:330 childs$ ./a.out
2*X = 4, 2*Y = 5.2
```

```
fawcett:330 childs$ g++ -E logger_defines.C
# 1 "logger_defines.C"
# 1 "<built-in>"
# 1 "<command-line>"
# 1 "logger_defines.C"
# 15 "logger_defines.C"
int Doubler(int X) { return 2*X; };
float Doubler(float X) { return 2*X; };


int main()
{
    int X = 2;
    float Y = 2.6;

}
```

First Template

```
fawcett:330 childs$ cat doubler_template.C

#include <iostream>

using std::cout;
using std::endl;

template <class T> T Doubler(T X) { return 2*X; };

int main()
{
    int   X = 2;
    float Y = 2.6;
    cout << "2*X = " << Doubler(X) << ", 2*Y = " << Doubler(Y) << endl;
}

fawcett:330 childs$ g++ doubler_template.C
fawcett:330 childs$ ./a.out
2*X = 4, 2*Y = 5.2
```

```
fawcett:330 childs$ nm a.out
0000000100000d7a s  stub helpers
00000001000010c0 D _NXArgc
00000001000010c8 D _NXArgv
0000000100000abb t __GLOBAL__I_main
0000000100000a78 t __Z41__static_initialization_and_destruction_0ii
0000000100000ced T __Z7DoublerIfET_S0_
0000000100000cdf T __Z7DoublerIiET_S0_
```

Will now do an example to compare templates and virtual functions

- Will take some buildup…

Money Class

```
class Money
{
  public:
                Money(int d, int c) { dollars = d; cents = c; };
    bool        operator<(const Money &m);

  private:
    int dollars;
    int cents;
};

bool Money::operator<(const Money &m)
{
    if (dollars < m.dollars)
        return true;
    if (dollars == m.dollars)
        return (cents < m.cents);

    return false;
}
```

```
int main()
{
    Money m(6, 85);
    Money m2(6, 25);
    bool lt = m < m2;
    cerr << "LT = " << lt << endl;
    lt = m2 < m;
    cerr << "LT = " << lt << endl;
}
C02LN00GFD58:330 hank$ g++ money.C
C02LN00GFD58:330 hank$ ./a.out
LT = 0
LT = 1
```

License Plate Class

```
class LicensePlate
{
  public:
                LicensePlate(char c1, char c2, char c3,
                             int i1, int i2, int i3)
                   {
                      letters[0] = c1;
                      letters[1] = c2;
                      letters[2] = c3;
                      numbers[0] = i1;
                      numbers[1] = i2;
                      numbers[2] = i3;
                   }

    bool        operator<(const LicensePlate &);

  private:
    char        letters[3];
    int         numbers[3];
};
```

```
bool LicensePlate::operator<(const LicensePlate &rhs)
{
    for (int i = 0 ; i < 3 ; i++)
    {
        if (letters[i] < rhs.letters[i])
            return true;
        if (letters[i] > rhs.letters[i])
            return false;
    }
    for (int i = 0 ; i < 3 ; i++)
    {
        if (numbers[i] < rhs.numbers[i])
            return true;
        if (numbers[i] > rhs.numbers[i])
            return false;
    }

    // equal
    return false;
}
```

```
int main()
{
    LicensePlate lp1('a', 'b', 'c', 4, 5, 6);
    LicensePlate lp2('c', 'b', 'a', 6, 5, 4);
    bool lt = lp1 < lp2;
    cerr << "LT = " << lt << endl;
    lt = lp2 < lp1;
    cerr << "LT = " << lt << endl;
}
```

Sorting With Templates

```
template <class T>
void Sort(T **X, int nX)
{
    for (int i = 0 ; i < nX ; i++)
        for (int j = i+1 ; j < nX ; j++)
        {
            if (*X[j] < *X[i])
            {
                T *tmp = X[j];
                X[j] = X[i];
                X[i] = tmp;
            }
        }
}
```

```
C02LN00GFD58:330 hank$ g++ template_sort.C
C02LN00GFD58:330 hank$ ./a.out
0: $4.25
1: $5.25
2: $6.25
3: $6.85
0: abc456
1: bba654
2: cba654
3: cda654
```

```
int main()
{
    Money m1(6, 85);
    Money m2(6, 25);
    Money m3(4, 25);
    Money m4(5, 25);

    LicensePlate lp1('a', 'b', 'c', 4, 5, 6);
    LicensePlate lp2('c', 'b', 'a', 6, 5, 4);
    LicensePlate lp3('c', 'd', 'a', 6, 5, 4);
    LicensePlate lp4('b', 'b', 'a', 6, 5, 4);

    Money        *money_list[4] = { &m1, &m2, &m3, &m4 };
    LicensePlate *lp_list[4]    = { &lp1, &lp2, &lp3, &lp4 };

    Sort(money_list, 4);
    Sort(lp_list, 4);

    for (int i = 0 ; i < 4 ; i++)
        cout << i << ": $" << money_list[i]->dollars << "."
             << money_list[i]->cents << endl;

    for (int i = 0 ; i < 4 ; i++)
    {
        cout << i << ": ";
        PrintLicensePlate(lp_list[i]);
        cout << endl;
    }
}
```

Doing the same with inheritance

```
class Sortable
{
  public:
    virtual bool       operator<(const Sortable *) = 0;
};

class LicensePlate : public Sortable
{
  public:
               LicensePlate(char c1, char c2, char c3,
                            int i1, int i2, int i3)
                  {
                     letters[0] = c1;
                     letters[1] = c2;
                     letters[2] = c3;
                     numbers[0] = i1;
                     numbers[1] = i2;
                     numbers[2] = i3;
                  }

    bool       operator<(const Sortable *);

  public:
    char       letters[3];
    int        numbers[3];
};
```

```
void Sort(Sortable **X, int nX)
{
    for (int i = 0 ; i < nX ; i++)
        for (int j = i+1 ; j < nX ; j++)
        {
            if (*X[j] < X[i])
            {
                Sortable *tmp = X[j];
                X[j] = X[i];
                X[i] = tmp;
            }
        }
}
```

```
int main()
{
    LicensePlate lp1('a', 'b', 'c', 4, 5, 6);
    LicensePlate lp2('c', 'b', 'a', 6, 5, 4);
    LicensePlate lp3('c', 'd', 'a', 6, 5, 4);
    LicensePlate lp4('b', 'b', 'a', 6, 5, 4);

    Sortable *lp_list[4]    = { &lp1, &lp2, &lp3, &lp4 };

    Sort(lp_list, 4);

    for (int i = 0 ; i < 4 ; i++)
    {
        cout << i << ": ";
        PrintLicensePlate((LicensePlate *)lp_list[i]);
        cout << endl;
    }
}
```

Templates vs Virtual Functions

- Virtual functions:
  - Had to affect inheritance hierarchy
  - Overhead in function call (virtual function table)
- Templates:
  - Did not need to affect inheritance hierarchy, although function names had to coincide
  - No additional overhead (resolved at compile time)

Standard Template Library

Standard Template Library

- Standard Template Library: STL
- Many, many templated types
- Can ease your programming burden
- Can also hide significant performance issues
  - And you use C/C++ for performance
- My recommendation: use with caution for code that needs to be performant

Vector

```
#include <vector>

using std::vector;

int main()
{
    vector<int> intArray(2);
    intArray[0] = 0;
    intArray[1] = 1;
    intArray.push_back(1);
    intArray.push_back(2);
    intArray.push_back(3);
    intArray.push_back(5);
    cout << "Size is " << intArray.size() << endl;
    cout << "Last val of Fib is " << intArray[5] << endl;
}
C02LN00GFD58:330 hank$ g++ vector.C
C02LN00GFD58:330 hank$ ./a.out
Size is 6
Last val of Fib is 5
```

std::vector

- Always has the amount of memory you need
- Many STL algorithms work on it
- Memory allocation:
  - If you don't have enough, double it
    - (can be a big overestimation)
- Overhead in access
  - Maybe not a big deal if data-intensive?

STL: map

```
#include <map>
#include <string>

using std::map;
using std::string;

int main()
{
    map<string, int> ageLookup;
    ageLookup["Hank"] = 37;
    ageLookup["Charlotte"] = 11;
    ageLookup["William"] = 9;

    cout << "Hank's age is " << ageLookup["Hank"] << endl;
    cout << "Carissa's age is " << ageLookup["Carissa"] << endl;
}
C02LN00GFD58:330 hank$ g++ map.C
C02LN00GFD58:330 hank$ ./a.out
Hank's age is 37
Carissa's age is 0
```

Outline

- Announcements
- Special Topics
- Review
- Potpourri from Lec 17
- Templates
- Standard Template Library
- C++ Strings
- Bonus Slides

(not a template thing): String

- C++ string class is very useful
- Great implementation of a class that encapsulates a string

```
#include <string>

using std::string;

int main()
{
    string str = "Hello";
    str += " world";
    cout << str << endl;
}
C02LN00GFD58:330 hank$ g++ string.C
C02LN00GFD58:330 hank$ ./a.out
Hello world
```

String methods

Iterators:

begin	Return iterator to beginning (public member function)
end	Return iterator to end (public member function)
rbegin	Return reverse iterator to reverse beginning (public member function)
rend	Return reverse iterator to reverse end (public member function)
cbegin C++11	Return const_iterator to beginning (public member function)
cend C++11	Return const_iterator to end (public member function)
crbegin C++11	Return const_reverse_iterator to reverse beginning (public member function)
crend C++11	Return const_reverse_iterator to reverse end (public member function)

Capacity:

size	Return length of string (public member function)
length	Return length of string (public member function)
max_size	Return maximum size of string (public member function)
resize	Resize string (public member function)
capacity	Return size of allocated storage (public member function)
reserve	Request a change in capacity (public member function)
clear	Clear string (public member function)
empty	Test if string is empty (public member function)
shrink_to_fit C++11	Shrink to fit (public member function)

String methods

Element access:

operator[]	Get character of string (public member function)
at	Get character in string (public member function)
back C++11	Access last character (public member function)
front C++11	Access first character (public member function)

Modifiers:

operator+=	Append to string (public member function)
append	Append to string (public member function)
push_back	Append character to string (public member function)
assign	Assign content to string (public member function)
insert	Insert into string (public member function)
erase	Erase characters from string (public member function)
replace	Replace portion of string (public member function)
swap	Swap string values (public member function)
pop_back C++11	Delete last character (public member function)

String operations:

c_str	Get C string equivalent (public member function)
data	Get string data (public member function)
get_allocator	Get allocator (public member function)
copy	Copy sequence of characters from string (public member function)
find	Find content in string (public member function)
rfind	Find last occurrence of content in string (public member function)
find_first_of	Find character in string (public member function)
find_last_of	Find character in string from the end (public member function)
find_first_not_of	Find absence of character in string (public member function)
find_last_not_of	Find non-matching character in string from the end (public member function)
substr	Generate substring (public member function)
compare	Compare strings (public member function)

Bonus Topics

Upcasting and Downcasting

- Upcast: treat an object as the base type
  - We do this all the time!
  - Treat a Rectangle as a Shape
- Downcast: treat a base type as its derived type
  - We don't do this one often
  - Treat a Shape as a Rectangle
    - You better know that Shape really is a Rectangle!!

Upcasting and Downcasting

```
class A
{
};

class B : public A
{
  public:
              B() { myInt = 5; };
    void      Printer(void) { cout << myInt << endl; };

  private:
    int       myInt;
};

void Downcaster(A *a)
{
    B *b = (B *) a;
    b->Printer();
}

int main()
{
    A a;
    B b;

    Downcaster(&b);  // no problem
    Downcaster(&a);  // no good
}
```

```
fawcett:330 childs$ g++ downcaster.C
fawcett:330 childs$ ./a.out
5
-1074118656
```

what do we get?

Upcasting and Downcasting

- C++ has a built in facility to assist with downcasting: dynamic_cast
- I personally haven't used it a lot, but it is used in practice
- Ties in to std::exception

Default Arguments

```
void Foo(int X, int Y = 2)
{
   cout << "X = " << X << ", Y = " << Y << endl;
}

int main()
{
    Foo(5);
    Foo(5, 4);
}

fawcett:330 childs$ g++ default.C
fawcett:330 childs$ ./a.out
X = 5, Y = 2
X = 5, Y = 4
```

default arguments: compiler pushes values on the stack for you if you choose not to enter them

Booleans

- New simple data type: bool (Boolean)
- New keywords: true and false

```
int main()
{
    bool b = true;
    cout << "Size of boolean is " << sizeof(bool) << endl;
}
fawcett:330 childs$ g++ Boolean.C
fawcett:330 childs$ ./a.out
```

Backgrounding

- "&": tell shell to run a job in the background
  - Background means that the shell acts as normal, but the command you invoke is running at the same time.
- "sleep 60" vs "sleep 60 &"

When would backgrounding be useful?

Suspending Jobs

- You can suspend a job that is running

  Press “Ctrl-Z”
- The OS will then stop job from running and not schedule it to run.
- You can then:
  - make the job run in the background.
    - Type “bg”
  - make the job run in the foreground.
    - Type “fg”
      - like you never suspended it at all!!

printf

- Print to stdout
  - printf("hello world\n");
  - printf("Integers are like this %d\n", 6);
  - printf("Two floats: %f, %f", 3.5, 7.0);

Have you ever wondered how printf takes a variable number of arguments?

Variable-Length Argument Lists

```
#include <stdarg.h>
#include <stdlib.h>
#include <stdio.h>

int SumIntList(int X, ...)
{
    va_list ap;   /* points to each unnamed arg in turn */
    int sum = 0;
    int ival;
    int i;

    va_start(ap, X); /* make ap point to 1st unnamed arg */
    for (i = 0 ; i < X ; i++)
    {
        ival = va_arg(ap, int);
        sum += ival;
    }
    va_end(ap);

    return sum;
}

int main()
{
    printf("List sum = %d\n", SumIntList(3, 13, 17, 22));
    printf("List sum = %d\n", SumIntList(5, 1, 2, 3, 4, 5));
}
C02LN00GFD58:Promotion hank$ ./a.out
List sum = 52
List sum = 15
```

Adapted from Kernigan & Ritchie C book

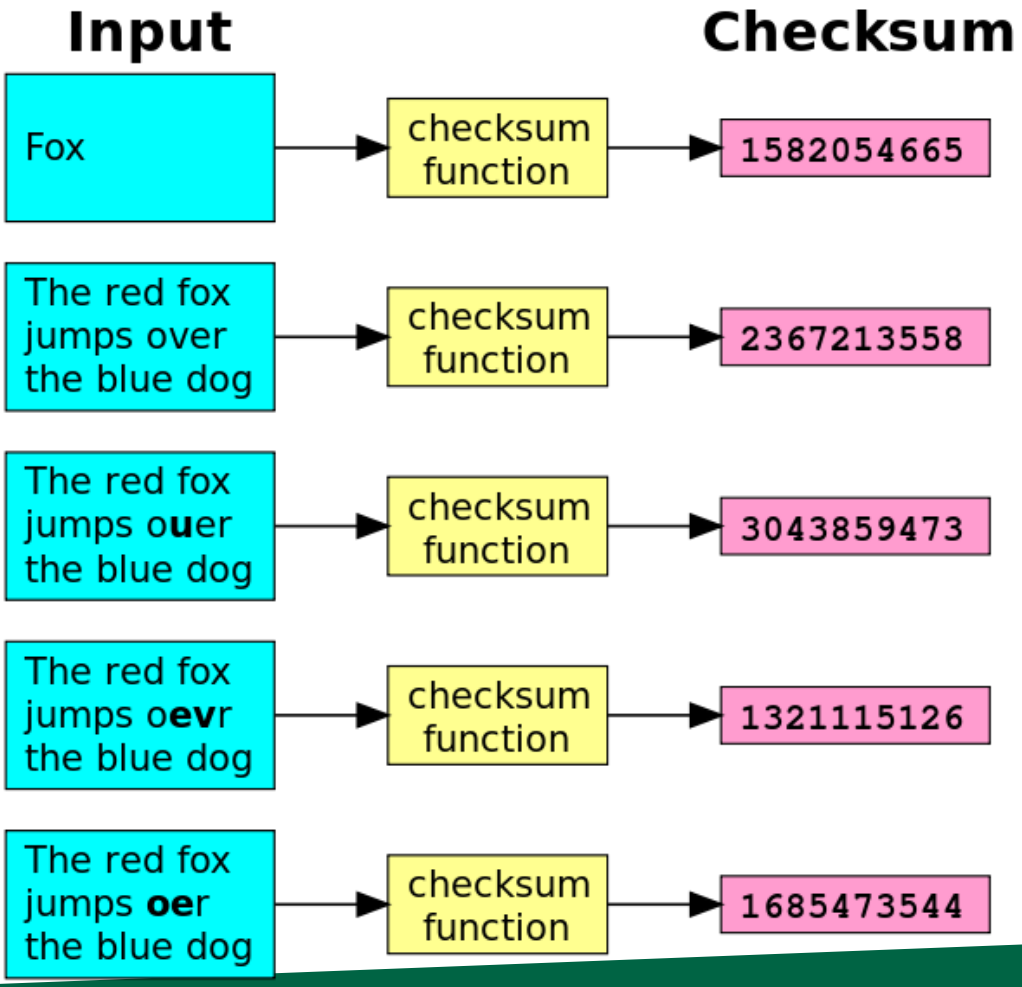
3H

- Will look at this together now

Stress Test Project (3H)

- We will have ~60 stress tests
- We can't check in 60 baseline images and difference them all
  - Will slow ix to a grind
- Solution:
  - We commit "essence of the solution"
  - We also complement that all images posted if needed.

Checksums



Most useful when input is very large and checksum is very small

From Wikipedia