Lecture 11:
C++ and structs
lots of random stuff
Announcements

• Projects
  – 3A assigned 4/24, due Weds 5/3
  – 2D assigned Monday 5/1, due Saturday 5/6
  – 3B assigned Monday 5/1, due Tuesday 5/9
  • BUT: probably need today’s lecture to know enough to do it ("constructors")
Random Topics
Random Topics #1

```c
fawcett:Dropbox childs$ cat t.c
#include <stdio.h>

struct arr
{
    int width;
    int height;
};

int main()
{
    struct arr y;
    FILE *f_in = fopen("hank", "r");
    fscanf(f_in, "%d\n", &y.width);
    fscanf(f_in, "%d\n", &y.height);
    printf("X is [%d, %d]\n", y.width, y.height);
}

fawcett:Dropbox childs$ cat hank
5
65
fawcett:Dropbox childs$ gcc t.c
fawcett:Dropbox childs$ ./a.out
X is [5, 65]
```
# Operator Precedence

<table>
<thead>
<tr>
<th>Precedence</th>
<th>Operator</th>
<th>Description</th>
<th>Associativity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>++ --</td>
<td>Suffix/postfix increment and decrement</td>
<td>Left-to-right</td>
</tr>
<tr>
<td></td>
<td>()</td>
<td>Function call</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[]</td>
<td>Array subscripting</td>
<td></td>
</tr>
<tr>
<td></td>
<td>.</td>
<td>Structure and union member access</td>
<td></td>
</tr>
<tr>
<td></td>
<td>-&gt;</td>
<td>Structure and union member access through pointer</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(type){list}</td>
<td>Compound literal(C99)</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>++ --</td>
<td>Prefix increment and decrement</td>
<td>Right-to-left</td>
</tr>
<tr>
<td></td>
<td>+ -</td>
<td>Unary plus and minus</td>
<td></td>
</tr>
<tr>
<td></td>
<td>! ~</td>
<td>Logical NOT and bitwise NOT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(type)</td>
<td>Type cast</td>
<td></td>
</tr>
<tr>
<td></td>
<td>*</td>
<td>Indirection (dereference)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&amp;</td>
<td>Address-of</td>
<td></td>
</tr>
<tr>
<td></td>
<td>sizeof</td>
<td>Size-of</td>
<td></td>
</tr>
<tr>
<td></td>
<td>__alignof</td>
<td>Alignment requirement(C11)</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>* / %</td>
<td>Multiplication, division, and remainder</td>
<td>Left-to-right</td>
</tr>
<tr>
<td>4</td>
<td>+ -</td>
<td>Addition and subtraction</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>&lt;&lt; &gt;&gt;</td>
<td>Bitwise left shift and right shift</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>&lt;= &gt;=</td>
<td>For relational operators &lt;= and &gt;= respectively</td>
<td></td>
</tr>
<tr>
<td></td>
<td>== !=</td>
<td>For relational = and ≠ respectively</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>&amp;</td>
<td>Bitwise AND</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>^</td>
<td>Bitwise XOR (exclusive or)</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td>Bitwise OR (inclusive or)</td>
</tr>
<tr>
<td>10</td>
<td>&amp;&amp;</td>
<td>Logical AND</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>? :</td>
<td>Ternary conditional</td>
<td>Right-to-Left</td>
</tr>
<tr>
<td>14</td>
<td>=</td>
<td>Simple assignment</td>
<td></td>
</tr>
<tr>
<td></td>
<td>+= -=</td>
<td>Assignment by sum and difference</td>
<td></td>
</tr>
<tr>
<td></td>
<td>*= /= %=</td>
<td>Assignment by product, quotient, and remainder</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;&lt;= &gt;&gt;=</td>
<td>Assignment by bitwise left shift and right shift</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&amp;= ^ =</td>
<td>=</td>
<td>Assignment by bitwise AND, XOR, and OR</td>
</tr>
<tr>
<td>15</td>
<td>,</td>
<td>Comma</td>
<td>Left-to-right</td>
</tr>
</tbody>
</table>

Random Topics #2
performance of different fread options?

It seems like there are maybe three different ways to use fread:

option 1: fread(location, size_of_element, number_of_elements, file)

option 2: fread(location, size_of_element * number_of_elements, 1, file)

option 3: loop over i < number_of_elements: fread(location + i, size_of_element, 1, file)

You might want to use different options depending on the context, but supposing it didn't matter, I was wondering which would be the best?

I figured option 3 would be the slowest because of all the function calls. I wrote a little program and got running times: option 2 < option 1 << option 3

Does anyone know why option 2 is the fastest? If you're interested, the test program I wrote is at: http://ix.cs.uoregon.edu/~hampton2/330/fread_test/

This isn't the most important thing in the world ... just goofing around :)
DRAM vs NV-RAM

- **DRAM: Dynamic Random Access Memory**
  - stores data
  - each bit in separate capacitor within integrated circuit
  - loses charge over time and must be refreshed
  - → volatile memory

- **NV-RAM: Non-Volatile Random Access Memory**
  - stores data
  - information unaffected by power cycle
  - examples: Read-Only Memory (ROM), flash, hard drive, floppy drive, …
Lecture 11 – File System
- File Systems could be implemented in DRAM.
- However, almost exclusively on NV-RAM
  - Most often hard drives
- Therefore, properties and benefits of file systems are often associated with properties and benefits of NV-RAM.
### DRAM vs NV-RAM properties

<table>
<thead>
<tr>
<th>Property</th>
<th>DRAM</th>
<th>NV-RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Capacity</td>
<td>~10GB</td>
<td>~10TB</td>
</tr>
<tr>
<td>Cost</td>
<td>$5/GB</td>
<td>$0.03/GB</td>
</tr>
<tr>
<td>Latency (in nanoseconds)</td>
<td>&lt;100</td>
<td>~10</td>
</tr>
</tbody>
</table>

*5 orders of magnitude!!*

What does 100000:1 mean?

- **Distance:** a 20” map of Oregon is 1:100,000 scale
  - Time: 1 second to 27 hours is 1:100,000 scale
  - Time: 1 minute to 69 days is 1:100,000 scale
  - Time: 1 hour to 11 years is 1:100,000 scale
  - Time: 1 day to 273 years is 1:100,000 scale
Random Topics #3
diff command embedded in Makefile does not work on ix machine

zhibin@ix-trusty: ~/cis330/3A 95$ make
gcc -o a 3A_c.c
./a 3A_input.pnm my_output.pnm
make: *** [my_output.pnm] Error 64
zhibin@ix-trusty: ~/cis330/3A 96$ make
diff my_output.pnm 3A_output.pnm

However, same Makefile works perfectly fine on my local machine.
Any thoughts?

Thanks in advance.
Random Topics #4
Why Can’t I Modify the Input in Yellow Diagonal

• Imagine I handed you the Mona Lisa with you and asked you to produce a version with a moustache...

• Would you?:
  – make a reproduction and add the moustache to the reproduction
  – vandalize the original
With Respect to 3B...
How to Make a Reproduction

```c
struct name *
ToLowerGood(struct name *input)
{
    struct name *rv = malloc(sizeof(struct name));
    int nchars = strlen(input->buffer);
    rv->buffer = malloc(sizeof(char)*nchars + 1);
    for (int i = 0 ; i < nchars ; i++)
    {
        char c = input->buffer[i];
        if (c >= 'A' && c <= 'Z')
            c = 'a' + (c-'A');
        rv->buffer[i] = c;
    }
    rv->buffer[nchars] = '\0';
    return rv;
}

/* goal: hack childs */

printf("N2's buffer is %s\n", n2->buffer);
printf("N1's buffer is %s\n", n1.buffer);
```

```c
struct name *
ToLowerBad(struct name *input)
{
    int nchars = strlen(input->buffer);
    for (int i = 0 ; i < nchars ; i++)
    {
        char c = input->buffer[i];
        if (c >= 'A' && c <= 'Z')
            c = 'a' + (c-'A');
        input->buffer[i] = c;
    }
    return input;
}
```
Review
Definition of Rectangle in rectangle.c
Why is this a problem?

```c
#include <prototypes.h>

int main()
{
    struct Rectangle r;
    InitializeRectangle(r, 0, 1, 0, 1.5);
}
```

```c
struct Rectangle;
void InitializeRectangle(struct Rectangle *r, double v1, double v2, double v3, double v4);
```

```c
struct Rectangle
{
    double maxX, maxY, minX, minY;
}
void InitializeRectangle(
    struct Rectangle *r,
    double v1,
    double v2,
    double v3,
    double v4)
{
    r->minX = v1;
    r->maxX = v2;
    r->minY = v3;
    r->maxY = v4;
}
```

“gcc –c driver.c” needs to make an object file. It needs info about Rectangle then, not later.
The fix is to make sure driver.c has access to the Rectangle struct definition.

gcc –E shows what the compiler sees after satisfying “preprocessing”, which includes steps like “#include”.

This is it. If the compiler can’t figure out how to make object file with this, then it has to give up.
What is the problem with this configuration?
Compilation error

C02LN00GFD58:project hank$ make
  gcc -I. -c rectangle.c
In file included from rectangle.c:2:
In file included from ./prototypes.h:2:
./struct.h:2:8: **error:** redefinition of 'Rectangle'
  struct Rectangle
  ^

./struct.h:2:8: **note:** previous definition is here
  struct Rectangle
  ^

1 error generated.
make: *** [rectangle.o] Error 1
gcc -E rectangle.c

C02LN00GFD58:project hank$ gcc -E -I. rectangle.c

# 1 "rectangle.c"
# 1 "<built-in>" 1
# 1 "<built-in>" 3
# 162 "<built-in>" 3
# 1 "<command line>" 1
# 1 "<built-in>" 2
# 1 "rectangle.c" 2
# 1 "./struct.h" 1

struct Rectangle
{
    double minX, maxX, minY, maxY;
};
# 2 "rectangle.c" 2
# 1 "./prototypes.h" 1
# 1 "./struct.h" 1

struct Rectangle
{
    double minX, maxX, minY, maxY;
};
# 3 "./prototypes.h" 2

void InitializeRectangle(struct Rectangle *r, double v1, double v2, double v3, double v4);
# 3 "rectangle.c" 2

void InitializeRectangle(struct Rectangle *r, double v1, double v2, double v3, double v4)
{
    r->minX = v1;
    r->maxX = v2;
    r->minY = v3;
    r->maxY = v4;
}
#ifndef
  
#define
  
  to
  
  the
  
  rescue

#ifndef RECTANGLE_330
#define RECTANGLE_330

struct Rectangle
{
  double minX, maxX, minY, maxY;
};

#endif

Why does this work?

This problem comes up a lot with big projects, and especially with C++. 

struct.h
A new compiler: g++

• g++ is the GNU C++ compiler
  – Flags are the same
  – Compiles C programs as well
    • (except those that aren’t valid C++ programs)
.c vs .C

• Unix is case sensitive
  – (So are C and C++)

• Conventions:
  – .c: C file
  – .C: C++ file
  – .cxx: C++ file
  – .cpp: C++ file (this is pretty rare)

Gnu compiler will sometimes assume the language based on the extension ... CLANG won’t.
Problem with C...

C02LN00GFD58:330 hank$ cat doubler.c
float doubler(float f) { return 2*f; }
C02LN00GFD58:330 hank$ gcc -c doubler.c
C02LN00GFD58:330 hank$ cat doubler_example.c
#include <stdio.h>

int doubler(int);

int main()
{
    printf("Doubler of 10 is %d\n", doubler(10));
}
C02LN00GFD58:330 hank$ gcc -c doubler_example.c
C02LN00GFD58:330 hank$ gcc -o doubler_example doubler.o doubler_example.o
C02LN00GFD58:330 hank$ ./doubler_example
Doubler of 10 is 2

Problem with C...

C02LN00GFD58:330  hank$ nm doubler.o
00000000000000048  s  EH_frame0
00000000000000000  T  _doubler
00000000000000060  S  _doubler.eh
C02LN00GFD58:330  hank$ nm doubler
doubler.c  doubler_example  doubler_example.o
doubler.o  doubler_example.c  doubler_user.o
C02LN00GFD58:330  hank$ nm doubler_example.o
00000000000000068  s  EH_frame0
00000000000000032  s  L_.str
U  _doubler
0000000000000000  T  _main
00000000000000080  S  _main.eh
U  _printf

No checking of type...
Problem is fixed with C++...

C02LN00GFD58:330 hank$ cat doubler.c
float doubler(float f) { return 2*f; }
C02LN00GFD58:330 hank$ g++ -c doubler.c
clang: warning: treating 'c' input as 'c++' when in C++ mode, this behavior is deprecated
C02LN00GFD58:330 hank$ cat doubler_example.c
#include <stdio.h>

int doubler(int);

int main()
{
    printf("Doubler of 10 is %d\n", doubler(10));
}
C02LN00GFD58:330 hank$ g++ -c doubler_example.c
clang: warning: treating 'c' input as 'c++' when in C++ mode, this behavior is deprecated
C02LN00GFD58:330 hank$ g++ -o doubler_example doubler_example.o doubler.o
Undefined symbols for architecture x86_64:
  "doubler(int)", referenced from:
    _main in doubler_example.o
ld: symbol(s) not found for architecture x86_64
clang: error: linker command failed with exit code 1 (use -v to see invocation)
C02LN00GFD58:330 hank$
Problem is fixed with C++...

This will affect you with C++. Before you got unresolved symbols when you forgot to define the function. Now you will get it when the arguments don’t match up. Is this good?
Mangling

- Mangling refers to combing information about arguments and “mangling” it with function name.
  - Way of ensuring that you don’t mix up functions.
  - Why not return type too?
- Causes problems with compiler mismatches
  - C++ compilers haven’t standardized.
  - Can’t take library from icpc and combine it with g++. 
C++ will let you overload functions with different types

```c
float doubler(float f) { return 2*f; }
int doubler(int f) { return 2*f; }
```

```
C02LN00GFD58:330 hank$ gcc -c t.c
```

```
t.c:2:5: error: conflicting types for 'doubler'
    int doubler(int f) { return 2*f; }
    ^
```

```
t.c:1:7: note: previous definition is here
float doubler(float f) { return 2*f; }
    ^
```

1 error generated.

```
C02LN00GFD58:330 hank$ g++ -c t.C
C02LN00GFD58:330 hank$ 
```
C++ also gives you access to mangling via “namespaces”

```c
#include <stdio.h>

namespace CIS330
{
    int GetNumberOfStudents(void) { return 56; }
}

namespace CIS610
{
    int GetNumberOfStudents(void) { return 9; }
}

int main()
{
    printf("Number of students in 330 is %d, but in 610 was %d\n",
            CIS330::GetNumberOfStudents(),
            CIS610::GetNumberOfStudents());
}
```

Functions or variables within a namespace are accessed with “::”
C++ also gives you access to mangling via “namespaces”

The “using” keyword makes all functions and variables from a namespace available without needing “::”. And you can still access other namespaces.

```c
namespace CIS610
{
    int GetNumberOfStudents(void) { return 9; }
}

using namespace CIS330;

int main()
{
    printf("Number of students in 330 is %d, but in 610 was %d\n",
            GetNumberOfStudents(),
            CIS610::GetNumberOfStudents());
}
```

```bash
C02LN00GFD58:330 hank$ cat cis330.C
C02LN00GFD58:330 hank$ g++ cis330.C
C02LN00GFD58:330 hank$ ./a.out
Number of students in 330 is 56, but in 610 was 9
C02LN00GFD58:330 hank$ 
```
References

• A reference is a simplified version of a pointer.
• Key differences:
  – You cannot do pointer manipulations
  – A reference is always valid
    • a pointer is not always valid
• Accomplished with & (ampersand)
  – &: address of variable (C-style, still valid)
  – &: reference to a variable (C++-style, also now valid)

You have to figure out how ‘&’ is being used based on context.
Examples of References

```c
#include <stdio.h>

void ref_doubler(int &x) { x = 2*x; }

int main()
{
    int x1 = 2;
    ref_doubler(x1);
    printf("Val is %d\n", x1);
}
```

```bash
hank$ cat ref.C
hank$ g++ ref.C
hank$ ./a.out
Val is 4
```
References vs Pointers vs Call-By-Value

```c
#include <stdio.h>

void ref_doubler(int &x) { x = 2*x; }
void ptr_doubler(int *x) { *x = 2**x; }
void val_doubler(int x) { x = 2*x; }

int main()
{
    int x1 = 2, x2 = 2, x3 = 2;
    ref_doubler(x1);
    ptr_doubler(&x2);
    val_doubler(x3);
    printf("Vals are %d, %d, %d\n", x1, x2, x3);
}
```

ref_doubler and ptr_doubler are both examples of call-by-reference. val_doubler is an example of call-by-value.
References

• Simplified version of a pointer.

• Key differences:
  – You cannot manipulate it
    • Meaning: you are given a reference to exactly one instance ... you can’t do pointer arithmetic to skip forward in an array to find another object
  – A reference is always valid
    • No equivalent of a NULL pointer ... must be a valid instance
New Stuff
Different Misc C++ Topic: initialization during declaration using parentheses

This isn’t that useful for simple types, but it will be useful when we start dealing with objects.
Learning classes via structs

• structs and classes are closely related in C++
• I lectured Friday on changes on how “structs in C++” are different than “structs in C”
• Soon I will describe how classes and structs in C++ differ.
3 Big changes to structs in C++

1) You can associate “methods” (functions) with structs
Methods vs Functions

• Methods and Functions are both regions of code that are called by name (“routines”)

• With functions:
  – the data it operates on (i.e., arguments) are explicitly passed
  – the data it generates (i.e., return value) is explicitly passed
  – stand-alone / no association with an object

• With methods:
  – associated with an object & can work on object’s data
  – still opportunity for explicit arguments and return value
Function vs Method

(left) function is separate from struct
(right) function (method) is part of struct

C02LN00GFD58:330 hank$ cat function.c

typedef struct
{
  int i;
} Integer;

int doubler(int x) { return 2*x; };

int main()
{
  Integer i;
  i.i = 3;
  i.i = doubler(i.i);
}

typedef struct
{
  int i;

  void doubler(void) { i = 2*i; };
} Integer;

int main()
{
  Integer i;
  i.i = 3;
  i.doubler();

(left) arguments and return value are explicit
(right) arguments and return value are not necessary, since they are associated with the object
Tally Counter

3 Methods: Increment Count, Get Count, Reset
Methods & Tally Counter

• Methods and Functions are both regions of code that are called by name (“routines”)
• With functions:
  – the data it operates on (i.e., arguments) are explicitly passed
  – the data it generates (i.e., return value) is explicitly passed
  – stand-alone / no association with an object
• With methods:
  – associated with an object & can work on object’s data
  – still opportunity for explicit arguments and return value
C-style implementation of TallyCounter

```c
C02LN00GFD58:TC hank$ cat tallycounter_c.c
#include <stdio.h>

typedef struct
{
    int count;
} TallyCounter;

void ResetTallyCounter(TallyCounter *tc) { tc->count = 0; }
int GetCountFromTallyCounter(TallyCounter *tc) { return tc->count; }
void TallyCounterIncrementCount(TallyCounter *tc) { tc->count++; }

int main()
{
    TallyCounter tc;
    tc.count = 0;
    TallyCounterIncrementCount(&tc);
    TallyCounterIncrementCount(&tc);
    TallyCounterIncrementCount(&tc);
    TallyCounterIncrementCount(&tc);
    printf("Count is %d\n", GetCountFromTallyCounter(&tc));
}
```

C02LN00GFD58:TC hank$ gcc tallycounter_c.c
C02LN00GFD58:TC hank$ ./a.out
Count is 4
C++-style implementation of TallyCounter

```c
#include <stdio.h>

typedef struct
{
    int    count;

    void   Reset() { count = 0; }
    int    GetCount() { return count; }
    void   IncrementCount() { count++; }
} TallyCounter;

int main()
{
    TallyCounter tc;
    tc.count = 0;
    tc.IncrementCount();
    tc.IncrementCount();
    tc.IncrementCount();
    tc.IncrementCount();
    printf("Count is %d\n", tc.GetCount());
}
```

```
$ cat tallycounter.C
$ g++ tallycounter.C
$ ./a.out
Count is 4
```
typedef struct
{
    int count;

    void Initialize() { count = 0; };
    void Reset() { count = 0; };
    int getCount() { return count; };
    void IncrementCount() { count++; };
} TallyCounter;

int main()
{
    TallyCounter tc;
    tc.Initialize();
    tc.IncrementCount();
    tc.IncrementCount();
    tc.IncrementCount();
    tc.IncrementCount();
    printf("Count is %d\n", tc.GetCount());
}
Constructors

• Constructor: method for constructing object.
  – Called automatically

• There are several flavors of constructors:
  – Parameterized constructors
  – Default constructors
  – Copy constructors
  – Conversion constructors

I will discuss these flavors in upcoming slides
Note the typedef went away ... not needed with C++.

(This is the flavor called “default constructor”)
Argument can be passed to constructor. (This is the flavor called “parameterized constructor”)
More traditional file organization

- struct definition is in .h file
  - #ifndef / #define
- method definitions in .C file
- driver file includes headers for all structs it needs
Methods can be defined outside the struct definition. They use C++’s namespace concept, which is automatically in place. (e.g., TallyCounter::IncrementCount)
“this”: pointer to current object

- From within any struct’s method, you can refer to the current object using “this”
Copy Constructor

• Copy constructor: a constructor that takes an instance as an argument
  – It is a way of making a new instance of an object that is identical to an existing one.

```c
struct TallyCounter
{
  int count;

  TallyCounter(void);
  TallyCounter(int c);
  TallyCounter(TallyCounter &);

  void Reset();
  int GetCount();
  void IncrementCount();
};

TallyCounter::TallyCounter(TallyCounter &c)
{
  count = c.count;
}
```
Constructor Types

```c
struct TallyCounter
{
    int    count;

    TallyCounter(void);
    TallyCounter(int c);
    TallyCounter(TallyCounter &);

    void Reset();
    int  GetCount();
    void IncrementCount();
};
```
Example of 3 Constructors

```c
C02LN00GFD58:TC hank$ cat main.C
#include <stdio.h>
#include <TallyCounter.h>

int main()
{
    TallyCounter tc;       /* Default constructor */
    tc.IncrementCount();

    TallyCounter tc2(10);  /* Parameterized constructor */
    tc2.IncrementCount();
    tc2.IncrementCount();

    TallyCounter tc3(tc);  /* copy constructor */
    tc3.IncrementCount();
    tc3.IncrementCount();
    tc3.IncrementCount();

    printf("Counts are %d, %d, %d\n", tc.GetCount(),
            tc2.GetCount(), tc3.GetCount());
}
C02LN00GFD58:TC hank$ ./main

?????????????????????
Conversion Constructor

```c
struct ImperialDistance
{
    double miles;
};

struct MetricDistance
{
    double kilometers;

    MetricDistance() { kilometers = 0; }
    MetricDistance(ImperialDistance &id)
    {
        kilometers = id.miles*1.609;
    }
};
```
3 big changes to structs in C++

1) You can associate “methods” (functions) with structs
2) You can control access to data members and methods
Access Control

• New keywords: public and private
  – public: accessible outside the struct
  – private: accessible only inside the struct

  • Also “protected” ... we will talk about that later

```cpp
struct TallyCounter {
    private:
        int count;

    public:
        TallyCounter(void);
        TallyCounter(int c);
        TallyCounter(TallyCounter &);
        void Reset();
        int GetCount();
        void IncrementCount();
};
```

Everything following is private. Only will change when new access control keyword is encountered.

Everything following is now public. Only will change when new access control keyword is encountered.
public / private

```cpp
struct TallyCounter
{
  public:
    TallyCounter(void);
    TallyCounter(int c);
    TallyCounter(TallyCounter &);
  
  private:
    int count;
  
  public:
    void Reset();
    int GetCount();
    void IncrementCount();
};
```

You can issue public and private as many times as you wish...
The compiler prevents violations of access controls.

```
128-223-223-72-wireless:TC hank$ cat main.C
#include <stdio.h>
#include <TallyCounter.h>

int main()
{
    TallyCounter tc;
    tc.count = 10;
}

128-223-223-72-wireless:TC hank$ make
g++ -I. -c main.C
main.C:7:8: error: 'count' is a private member of 'TallyCounter'
    tc.count = 10;
    ^
./TallyCounter.h:12:12: note: declared private here
    int count;
    ^

1 error generated.
make: *** [main.o] Error 1
```
The friend keyword can override access controls.

```cpp
struct TallyCounter {
    friend int main();

public:
    TallyCounter(void);
    TallyCounter(int c);
    TallyCounter(TallyCounter &);

private:
    int count;
}
```

- Note that the struct declares who its friends are, not vice-versa
  - You can’t declare yourself a friend and start accessing data members.
- friend is used most often to allow objects to access other objects.

This will compile, since main now has access to the private data member “count”.
class vs struct

• class is new keyword in C++
• classes are very similar to structs
  – the only differences are in access control
    • primary difference: struct has public access by default, class
      has private access by default
• Almost all C++ developers use classes and not structs
  – C++ developers tend to use structs when they want to
    collect data types together (i.e., C-style usage)
  – C++ developers use classes for objects ... which is
    most of the time

You should use classes!
Even though there isn’t much difference ...
3 big changes to structs in C++

1) You can associate “methods” (functions) with structs
2) You can control access to data members and methods
3) Inheritance
Simple inheritance example

```c
struct A {
    int x;
};

struct B : A {
    int y;
};

int main() {
    B b;
    b.x = 3;
    b.y = 4;
}
```

- Terminology
  - B inherits from A
  - A is a base type for B
  - B is a derived type of A

- Noteworthy
  - “:” (during struct definition) → inherits from
  - Everything from A is accessible in B
    - (b.x is valid!!)
Object sizes

#include <stdio.h>

struct A
{
  int x;
};

struct B : A
{
  int y;
};

int main()
{
  B b;
  b.x = 3;
  b.y = 4;
  printf("Size of A = %lu, size of B = %lu\n", sizeof(A), sizeof(B));
}

128-223-223-72-wireless:330 hank$ g++ simple_inheritance.C
128-223-223-72-wireless:330 hank$ ./a.out
Size of A = 4, size of B = 8
Inheritance + TallyCounter

```cpp
struct TallyCounter
{
    friend   int main();

    public:
        TallyCounter(void);
        TallyCounter(int c);
        TallyCounter(TallyCounter &);

    private:
        int   count;

    public:
        void   Reset();
        int    GetCount();
        void   IncrementCount();
};

struct FancyTallyCounter : TallyCounter
{
    void    DecrementCount() { count--; }
}
```

FancyTallyCounter inherits all of TallyCounter, and adds a new method: DecrementCount
Virtual functions

• Virtual function: function defined in the base type, but can be re-defined in derived type.
• When you call a virtual function, you get the version defined by the derived type.
```c
#include <stdio.h>

struct SimpleID
{
    int id;
    virtual int GetIdentifier() { return id; }
};

struct ComplexID : SimpleID
{
    int extraId;
    virtual int GetIdentifier() { return extraId*128+id; }
};

int main()
{
    ComplexID cid;
    cid.id = 3;
    cid.extraId = 3;
    printf("ID = %d\n", cid.GetIdentifier());
}
```

Virtual functions: example
Virtual functions:

You get the method furthest down in the inheritance hierarchy

Example:

```c
#include <stdio.h>

struct SimpleID {
    int id;
    virtual int GetIdentifier() { return id; }
};

struct ComplexID : SimpleID {
    int extraId;
    virtual int GetIdentifier() { return extraId*128-id; }
};

struct C3 : ComplexID {
    int extraExtraId = 3;
    virtual int GetIdentifier() { return extraExtraId; }
};

C3 cid;

int main() {
    cid.id = 3;
    cid.extraId = 3;
    printf("ID = %d\n", cid.GetIdentifier());
}
```
Virtual functions: example

You can specify the method you want to call by specifying it explicitly.
Access controls and inheritance

C02LN00GFD58:330 hank$ cat inheritance.C
struct A { int x; };
struct B : A { int y; };
struct C : public A { int y; };
struct D : private A { int y; };

int main()
{
    C c;
c.x = 2;
    D d;
d.x = 2;
}

B and C are the same. public is the default inheritance for structs.

Public inheritance: derived types get access to base type’s data members and methods.

Private inheritance: derived types don’t get access.
One more access control word: protected

• Protected means:
  – It cannot be accessed outside the object
    • Modulo “friend”
  – But it can be accessed by derived types
    • (assuming public inheritance)
Public, private, protected

<table>
<thead>
<tr>
<th>Accessed by derived types*</th>
<th>Accessed outside object</th>
</tr>
</thead>
<tbody>
<tr>
<td>Public</td>
<td>Yes</td>
</tr>
<tr>
<td>Protected</td>
<td>Yes</td>
</tr>
<tr>
<td>Private</td>
<td>No</td>
</tr>
</tbody>
</table>

* = with public inheritance
More on virtual functions upcoming

- “Is A”
- Multiple inheritance
- Virtual function table
- Examples
  - (Shape)
Bonus Topics
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!!
Web pages

• ssh –l <user name> ix.cs.uoregon.edu
• cd public_html
• put something in index.html
• → it will show up as

  [link]
  http://ix.cs.uoregon.edu/~<username>
Web pages

• You can also exchange files this way
  – scp file.pdf <username>@ix.cs.uoregon.edu:~/public_html
  – point people to http://ix.cs.uoregon.edu/~<username>/file.pdf

Note that ~/public_html/dir1 shows up as
http://ix.cs.uoregon.edu/~<username>/dir1

(“~/dir1” is not accessible via web)