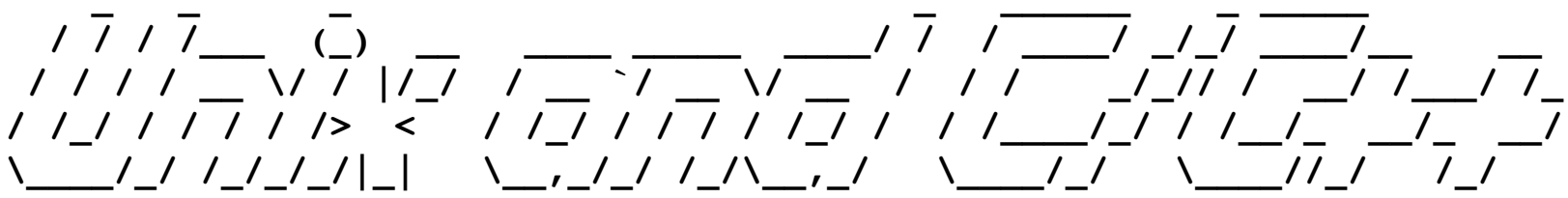
UNIVERSITY OF OREGON

CIS 330:



Lecture 7: enum, structs, and unions

April 20th, 2016

Hank Childs, University of Oregon

Announcements

- Projects
  - 4A assigned: due in class on Weds
  - 2B prompt available last Saturday AM, due this Friday
  - 2C prompt assigned today, due Tuesday
- No class on Weds April 27
  - will be a short YouTube replacement lecture

Grading (1/2)

- code isn't always portable
  - Scenario #1
    - your compiler is smart and glosses over a small problem
    - my compiler isn't as smart, and gets stuck on the problem
    - (fix: gcc –pedantic)
  - Scenario #2
    - memory error doesn't trip you up
    - memory error does trip me up
    - (fix: run valgrind before submitting)

Grading (2/2)

- Graders will run your code
  - If it works for them, then great
  - If not, they may ding you
- → how to resolve?
  - ix.cs.uoregon.edu is my reference machine
  - if it works there, you have a very strong case for getting credit back

Outline

- Grade 4A
- Review
- Project 2B
- Enum
- Struct
- Unions
- Project 2C

Let’s Grade 4A

Outline

- Grade 4A
- Review
- Project 2B
- Enum
- Struct
- Unions
- Project 2C

Memory Errors

- Array bounds read

```
int main()
{
    int var;
    int arr[3] = { 0, 1, 2 };
    var=arr[3];
}
```

- Array bounds write

```
int main()
{
    int var = 2;
    int arr[3];
    arr[3]=var;
}
```

Great question from last class

- int *X = 0xDEADBEEF:
- int Y = *X;

- what error code fits?

- I missed two:
  - Invalid Pointer Read (IPR)
  - Invalid Pointer Writer (IPW)

File I/O: streams and file descriptors

- Two ways to access files:
  - File descriptors:
    - Lower level interface to files and devices
      - Provides controls to specific devices
    - Type: small integers (typically 20 total)
  - Streams:
    - Higher level interface to files and devices
      - Provides uniform interface; easy to deal with, but less powerful
    - Type: FILE *

Streams are more portable, and more accessible to beginning programmers. (I teach streams here.)

File I/O

- Process for reading or writing
  - Open a file
    - Tells Unix you intend to do file I/O
    - Function returns a “FILE *
      - Used to identify the file from this point forward
    - Checks to see if permissions are valid
  - Read from the file / write to the file
  - Close the file

Example

```
C02LN00GFD58:330 hank$ cat rw.c
#include <stdio.h>
#include <string.h>

int main(int argc, char *argv[])
{
   char *hello = "hello world: file edition\n";
   FILE *f = fopen("330", "w");
   fwrite(hello, sizeof(char), strlen(hello), f);
   fclose(f);
}
C02LN00GFD58:330 hank$ gcc rw.c
C02LN00GFD58:330 hank$ ./a.out
C02LN00GFD58:330 hank$ cat 330
hello world: file edition
```

```c
#include <stdio.h>
#include <printf.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
    FILE *f_in, *f_out;
    int buff_size;
    char *buffer;

    if (argc != 3)
    {
        printf("Usage: %s <file1> <file2>\n", argv[0]);
        exit(EXIT_FAILURE);
    }

    f_in = fopen(argv[1], "r");
    fseek(f_in, 0, SEEK_END);
    buff_size = ftell(f_in);
    fseek(f_in, 0, SEEK_SET);

    buffer = malloc(buff_size);
    fread(buffer, sizeof(char), buff_size, f_in);

    printf("Copying %d bytes from %s to %s\n", buff_size, argv[1], argv[2]);

    f_out = fopen(argv[2], "w");
    fwrite(buffer, sizeof(char), buff_size, f_out);

    fclose(f_in);
    fclose(f_out);

    return 0;
}
```

Printing to terminal and reading from terminal

- In Unix, printing to terminal and reading from terminal is done with file I/O
- Keyboard and screen are files in the file system!
  - (at least they were …)

Standard Streams

- Wikipedia: "preconnected input and output channels between a computer program and its environment (typically a text terminal) when it begins execution"
- Three standard streams:
  - stdin (standard input)
  - stdout (standard output)
  - stderr (standard error)

What mechanisms in C allow you to access standard streams?

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);

fprintf

- Just like printf, but to streams
- fprintf(stdout, “helloworld\n”);
  - → same as printf
- fprintf(stderr, “helloworld\n”);
  - prints to “standard error”
- fprintf(f_out, “helloworld\n”);
  - prints to the file pointed to by FILE *f_out.

buffering and printf

- Important: printf is buffered
- So:
  - printf puts string in buffer
  - other things happen
  - buffer is eventually printed
- But what about a crash?
  - printf puts string in buffer
  - other things happen … including a crash
  - buffer is never printed!

Solutions: (1) fflush, (2) fprintf(stderr) always flushed

Outline

- Grade 4A
- Review
- Project 2B
- Enum
- Struct
- Unions
- Project 2C

Project 2B

(which means submitted by 6am on April 24th, 2015)
Worth 4% of your grade

Assignment: Write a program that reads the file "2E_binary_file". This file contains a two-dimensional array of integers, that is 10x10. You are to read in the 5x5 bottom left corner of the array. That is, the values 0-4, 10-14, 20-24, 30-34, and 40-44. You may only read 25 integers total. Do not read all 100 and throw some out. You will then write out the new 5x5 array. Please write this as strings, one integer per line (25 lines total). You should be able to "cat" the file afterwards and see the values.

Use Unix file streams for this project (i.e., fopen, fread, fseek, fprintf). Your program will be checked for good programming practices. (Close your file streams, use memory correctly, etc. I am not referring to style, variable initialization, etc.)

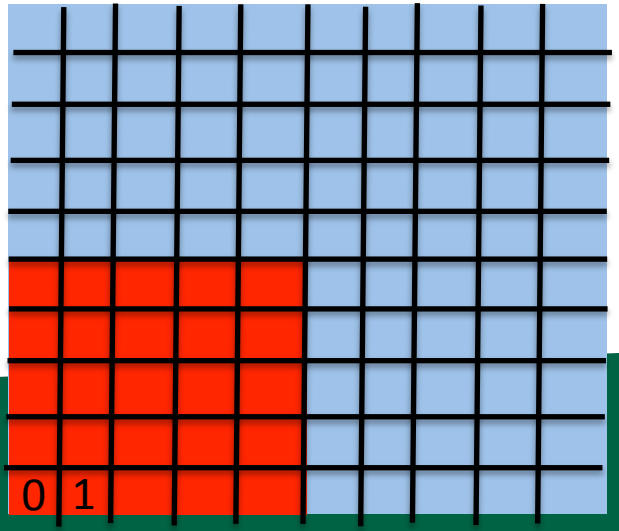
Also, add support for command line arguments (argc and argv).

Your program should run as:
./<prog_name> <input_name> <output_name>

(The input_name will be 2E_binary_file, unless you change it.)

Finally, note that I am handing you a binary file. I think we are all little endian, and so it will be fine. But, if it is big endian, then we will have a problem. You can check if it is little endian by printing the first two values of the file. They should be "0" and "1".

Please submit a tarball with (1) a Makefile (should be simple), (2) your source code, and (3) the output ASCII file from running your program, with the name "ASCII_output".



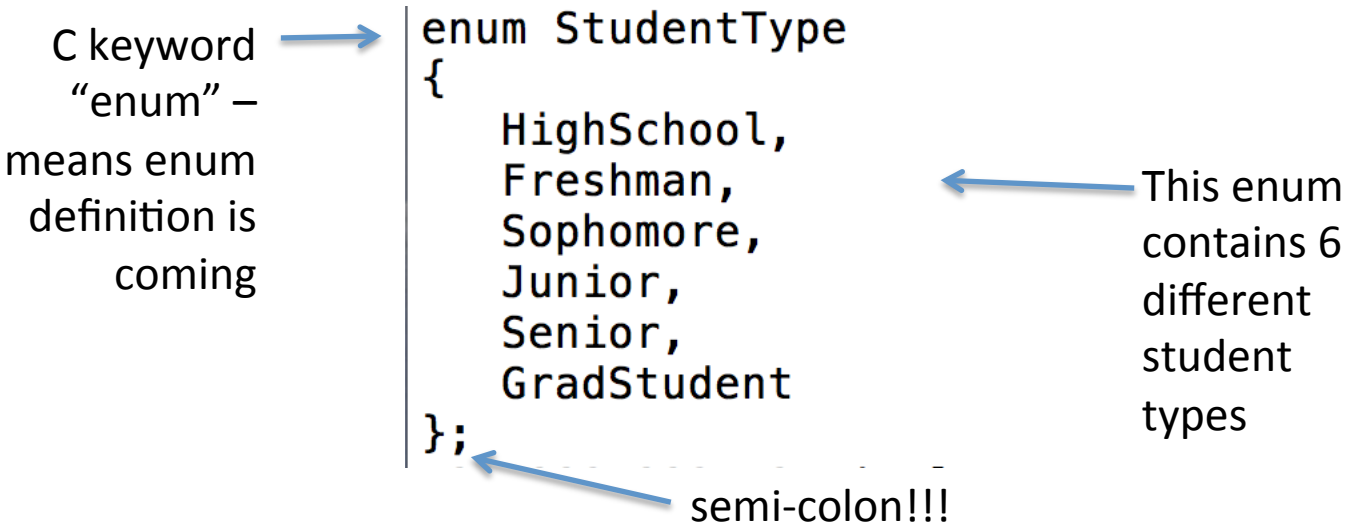
Outline

- Grade 4A
- Review
- Project 2B
- Enum
- Struct
- Unions
- Project 2C

Enums

- Enums make your own type
  - Type is “list of key words”
- Enums are useful for code clarity
  - Always possible to do the same thing with integers
- Be careful with enums
  - … you can “contaminate” a bunch of useful words

enum example



enum example

```
int AverageAge(enum StudentType st)
{
    if (st == HighSchool)
       return 16;
    if (st == Freshman)
       return 18;
    if (st == Sophomore)
       return 19;
    if (st == Junior)
       return 21;
    if (st == Senior)
       return 23;
    if (st == GradStudent)
       return 26;

    return -1;
}
```

enums translate to integers … and you can set their range

```
128-223-223-72-wireless:330 hank$ cat enum2.c
#include <stdio.h>

enum StudentType
{
   HighSchool = 105,
   Freshman,
   Sophomore,
   Junior,
   Senior,
   GradStudent
};

int main()
{
   printf("HighSchool = %d, GradStudent = %d\n", HighSchool, GradStudent);
}
128-223-223-72-wireless:330 hank$ gcc enum2.c
128-223-223-72-wireless:330 hank$ ./a.out
HighSchool = 105, GradStudent = 110
```

But enums can be easier to maintain than integers

```
enum StudentType
{
    HighSchool,
    Freshman,
    Sophomore,
    Junior,
    Senior,
    PostBacc,
    GradStudent
};
```

```
int AverageAge(enum StudentType st)
{
    if (st == HighSchool)
        return 16;
    if (st == Freshman)
        return 18;
    if (st == Sophomore)
        return 19;
    if (st == Junior)
        return 21;
    if (st == Senior)
        return 23;
    if (st == PostBac)
        return 24;
    if (st == GradStudent)
        return 26;

    return -1;
```

If you had used integers, then this is a bigger change and likely to lead to bugs.

Outline

- Grade 4A
- Review
- Project 2B
- Enum
- Struct
- Unions
- Project 2C

Data types

- float
- double
- int
- char
- unsigned char

All of these are simple data types

Structs: a complex data type

- Construct that defines a group of variables
  - Variables must be grouped together in contiguous memory
- Also makes accessing variables easier … they are all part of the same grouping (the struct)

struct syntax

C keyword "struct" – means struct definition is coming

```
struct Ray
{
    double origin[3];
    double direction[3];
};

int main()
{
    struct Ray r;
    r.origin[0] = 0;
    r.origin[1] = 0;
    r.origin[2] = 0;
    r.direction[0] = 1;
    r.direction[1] = 0;
    r.direction[2] = 0;
}
```

This struct contains 6 doubles, meaning it is 48 bytes

semi-colon!!!

Declaring an instance

"." accesses data members for a struct

Nested structs

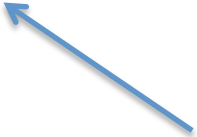
```
struct Origin
{
    double originX;
    double originY;
    double originZ;
};

struct Direction
{
    double directionX;
    double directionY;
    double directionZ;
};

struct Ray
{
    struct Origin ori;
    struct Direction dir;
};
```

```
int main()
{
    struct Ray r;
    r.ori.originX = 0;
    r.ori.originY = 0;
    r.ori.originZ = 0;
    r.dir.directionX = 0;
    r.dir.directionY = 0;
    r.dir.directionZ = 0;
}
```

accesses dir part of Ray



accesses directionZ part of Direction (part of Ray)

typedef

- typedef: tell compiler you want to define a new type

```
struct Ray
{
    double origin[3];
    double direction[3];
};

int main()
{
    struct Ray r;
    r.origin[0] = 0;
    r.origin[1] = 0;
    r.origin[2] = 0;
    r.direction[0] = 1;
    r.direction[1] = 0;
    r.direct
}
```

```
typedef struct
{
    double origin[3];
    double direction[3];
} Ray;

int main()
{
    Ray r;
    r.origin[0] = 0;
    r.origin[1] = 0;
    r.origin[2] = 0;
    r.direction[0] = 1;
    r.direction[1] = 0;
    0;
```

saves you from having to type “struct” every time you declare a struct.

Other uses for typedef

- Declare a new type for code clarity
  - typedef int MilesPerHour;
    - Makes a new type called MilesPerHour.
    - MilesPerHour works exactly like an int.
- Also used for enums & unions
  - same trick as for structs … typedef saves you a word

Outline

- Grade 4A
- Review
- Project 2B
- Enum
- Struct
- Union
- Project 2C

Unions

- Union: special data type
  - store many different memory types in one memory location

```
typedef union
{
    float x;
    int   y;
    char  z[4];
} cis330_union;
```

When dealing with this union, you can treat it as a float, as an int, or as 4 characters.

This data structure has 4 bytes

Unions

```
128-223-223-72-wireless:330 hank$ cat union.c
#include <stdio.h>

typedef union
{
    float x;
    int   y;
    char  z[4];
} cis330_union;


int main()
{
    cis330_union u;
    u.x = 3.5;  /* u.x is 3.5,     u.y and u.z are not meaningful */
    u.y = 3;    /* u.y is 3,   now u.x and u.z are not meaningful */
    printf("As u.x = %f, as u.y = %d\n", u.x, u.y);
}
128-223-223-72-wireless:330 hank$ gcc union.c
128-223-223-72-wireless:330 hank$ ./a.out
As u.x = 0.000000, as u.y = 3
```

Why are unions useful?

Unions Example

```
typedef struct
{
   int firstNum;
   char letters[3];
   int endNums[3];
} CA_LICENSE_PLATE;

typedef struct
{
   char  letters[3];
   int  nums[3];
} OR_LICENSE_PLATE;

typedef struct
{
   int  nums[6];
} WY_LICENSE_PLATE;

typedef union
{
   CA_LICENSE_PLATE ca;
   OR_LICENSE_PLATE or;
   WY_LICENSE_PLATE wy;
} LicensePlate;
```

Unions Example

```
typedef struct
{
   int firstNum;
   char letters[3];
   int endNums[3];
} CA_LICENSE_PLATE;

typedef struct
{
   char  letters[3];
   int  nums[3];
} OR_LICENSE_PLATE;

typedef struct
{
   int  nums[6];
} WY_LICENSE_PLATE;

typedef union
{
   CA_LICENSE_PLATE ca;
   OR_LICENSE_PLATE or;
   WY_LICENSE_PLATE wy;
} LicensePlate;
```

```
typedef enum
{
   CA,
   OR,
   WY
} US_State;

typedef struct
{
   char *carMake;
   char *carModel;
   US_State  state;
   LicensePlate lp;
} CarInfo;

int main()
{
    CarInfo c;
    c.carMake = "Chevrolet";
    c.carModel = "Camaro";
    c.state = OR;
    c.lp.or.letters[0] = 'X';
    c.lp.or.letters[1] = 'S';
    c.lp.or.letters[2] = 'Z';
    c.lp.or.nums[0] = 0;
    c.lp.or.nums[1] = 7;
    c.lp.or.nums[2] = 5;
}
```

Why are Unions useful?

- Allows you to represent multiple data types simultaneously
  - But only if you know you want exactly one of them
- Benefit is space efficiency, which leads to performance efficiency

Unions are also useful for abstracting type. We will re-visit this when we talk about C++'s templates.

Outline

- Grade 4A
- Review
- Project 2B
- Enum
- Struct
- Unions
- Project 2C

Project 2C

CIS 330: Project #2F
Assigned: April 22nd, 2015
Due April 29th, 2015
(which means submitted by 6am on April 30th, 2015)
Worth 4% of your grade

Assignment: You will implement 3 structs and 9 functions. The prototypes for the functions are located in the file prototypes.h (available on the website).

The three structs are Rectangle, Circle, and Triangle, and are described below.

The 3 structs refer to 3 different shapes: Triangle, Circle, and Rectangle.
For each shape, there are 3 functions: Initialize, GetArea, and GetBoundingBox.
You must implement 9 functions total (3*3).

The prototypes for these 9 functions are available in the file prototypes.h

There is also be a driver program, and correct output for the driver program.

Again, your job is to define 3 structs and 9 functions. The comments below clarify the format of the Rectangle, Circle, and Triangle, as well as the convention for GetBoundingBox, and an example of accessing data members for pointers to structs.

== Rectangle ==

The rectangle has corners (minX, minY), (maxX, minY), (minX, maxY), (maxX, maxY).
Its area is (maxX-minX)*(maxY-minY).
Its bounding box is from minX to maxX in X, and minY to maxY in Y.

== Circle ==

The circle has an origin (x and y) and a radius.

Its area is 3.14159*radius*radius.
Its bounding box is from (x-radius) to (x+radius) in X, and (y-radius) to (y+radius) in Y.

== Triangle ==

The triangle always has two points at the minimum Y-value. The third point's Y-value is at the maximum Y-value, and its X-value is at the average of the X's of the other two points. Saying it another way, the first two points form the "base", and the third point is "height" above it.

Thus, the height of the triangle is (pt2X-pt1X)*(maxY-minY)/2;
And the bounding box is from pt1X to pt2X in X, and from minY to maxY in Y.

== GetBoundingBox ==

The GetBoundingBox calls take a double * as an argument. If a shape has its minimum X at "a", its maximum X at "b", its minimum Y at "c", and its maximum Y at "d", then it should do something like:

```
void GetCircleBoundingBox(Circle *, double *bbox)
{
   bbox[0] = a;
   bbox[1] = b;
   bbox[2] = c;
   bbox[3] = d;
}
```

== Working with pointers to structs ==

We reviewed the way to access struct data members in class, which was with the "." operator. We did not review the way to access struct data members when you have a *pointer* to a struct. And the 9 function prototypes all use pointers to structs. It is done with the ->.

So:

```
typedef struct
{
   int X;
} Y;
int main()
{
  Y y;
  Y *y2;
  y2 = &y;
  y.x = 0;
  y2->x = 1;
}
```

Bonus Material

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
0000000000000048 s EH_frame0
0000000000000000 T _doubler
0000000000000060 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
0000000000000068 s EH_frame0
0000000000000032 s L_.str
                 U _doubler
0000000000000000 T _main
0000000000000080 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++…

```
C02LN00GFD58:330 hank$ nm doubler.o
000000000000048 s EH_frame0
000000000000000 T __Z7doublerf
000000000000060 S __Z7doublerf.eh
C02LN00GFD58:330 hank$ nm doubler_example.o
000000000000068 s EH_frame0
000000000000032 s L_.str
                 U __Z7doubleri
000000000000000 T _main
000000000000080 S _main.eh
                 U _printf
C02LN00GFD58:330 hank$
```

```
C02LN00GFD58:330 hank$ nm doubler.o
000000000000048 s EH_frame0
000000000000000 T _doubler
000000000000060 S _doubler.eh
C02LN00GFD58:330 hank$ nm doubler
doubler.c          doubler_example    d
doubler.o          doubler_example.c  d
C02LN00GFD58:330 hank$ nm doubler_examp
000000000000068 s EH_frame0
000000000000032 s L_.str
                 U _doubler
000000000000000 T _main
000000000000080 S _main.eh
                 U _printf
```

Mangling

- Mangling refers to combing information about the return type and arguments and "mangling" it with function name.
  - Way of ensuring that you don't mix up functions.
- 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

```
C02LN00GFD58:330 hank$ cat t.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”

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

Functions or variables within a namespace are accessed with “::”

C++ also gives you access to mangling via “namespaces”

```
C02LN00GFD58:330 hank$ cat cis330.C
```

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

```
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());
}
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$
```

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

  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)

Unix and Windows difference

- Unix:
  - "\n": goes to next line, and sets cursor to far left
- Windows:
  - "\n": goes to next line (cursor does not go to left)
  - "\m": sets cursor to far left

- Text files written in Windows often don't run well on Unix, and vice-versa
  - There are more differences than just newlines

vi: "set ff=unix" solves this