Lecture 10:
Building Large Projects,
Beginning C++,
C++ and structs
Outline

• Building Large Projects
• Beginning with C++
• Changes in C++ for structs
• Bonus topics (if time)
Outline

• Building Large Projects
• Beginning with C++
• Changes in C++ for structs
• Bonus topics (if time)
struct Rectangle;
void InitializeRectangle(struct Rectangle *r, double v1, double v2, double v3, double v4);

#include <prototypes.h>

int main()
{
    struct Rectangle r;
    InitializeRectangle(r, 0, 1, 0, 1.5);
}
Review on compilation

• gcc –c: build an object file (.o), i.e., binary code that can directly run on the architecture
• Then the binary can be generated from the object files.
• Libraries are a mechanism for grouping up a bunch of related object files
  – They are assembled together using a program called an archiver (ar)
• You can also just use object files directly when linking.
Makefiles

• Consists of rules
• Rule syntax:
  target: dependency1 dep2 ... depN
  <tab>command1
  <tab>command2

Quiz: write down a Makefile for a program called proj2B. Again, the file names are prototypes.h, driver.c, rectangle.c
Makefile for prototypes.h, rectangle.c, driver.c

proj2B: driver.c rectangle.c prototypes.h
    gcc -I. -c rectangle.c
    gcc -I. -c driver.c
    gcc -o proj2B driver.o rectangle.o

Is this a good Makefile?
What’s the problem with it?
proj2B: rectangle.o driver.o
   gcc -o proj2B driver.o rectangle.o

driver.o: prototypes.h driver.c
   gcc -I -c driver.c

rectangle.o: rectangle.c
   gcc -I -c rectangle.c
Definition of Rectangle in rectangle.c
Why is this a problem?

prototypes.h

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

rectangle.c

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

```c
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;
}
```

driver.c

```c
#include <prototypes.h>

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

“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.

```c
#include <prototypes.h>

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

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

```c
struct Rectangle;

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

This is it. If the compiler can’t figure out how to make object file with this, then it has to give up.
Preprocessor

• Preprocessor:
  – takes an input program
  – produces another program (which is then compiled)

• C has a separate language for preprocessing
  – Different syntax than C
  – Uses macros (“#”)

macro (“macroinstruction”): rule for replacing input characters with output characters
Preprocessor Phases

• Resolve #includes
  – (we understand #include phase)
• Conditional compilation
• Macro replacement
• Special macros
## Define Compilation

```
#define compile

This is an example of macro replacement.
```
#define via gcc command-line option

```c
#define via gcc command-line option

C02LN00GFD58:330 hank$ cat defines.c
int main()
{
    return RV;
}
```

```
C02LN00GFD58:330 hank$ gcc -DRV=4 defines.c
C02LN00GFD58:330 hank$ ./a.out
C02LN00GFD58:330 hank$ echo $?
4
```
Conflicting –D and #define

C02LN00GFD58:330 hank$ cat defines.c
#define RV 2
int main()
{
    return RV;
}
C02LN00GFD58:330 hank$ gcc -DRV=4 defines.c
defines.c:1:9: warning: 'RV' macro redefined
#define RV 2
^
<command line>:1:9: note: previous definition is here
#define RV 4
^
1 warning generated.
C02LN00GFD58:330 hank$ ./a.out
C02LN00GFD58:330 hank$ echo $?
2
Conditional compilation

```c
C02LN00GFD58:330 hank$ cat conditional.c
#define USE_OPTION 1

int main()
{
    DoMainCode();
    if (defined USE_OPTION)
        UseOption();
    endif
    DoCleanupCode();
}
```
Conditional compilation controlled via compiler flags

```c
#include <stdio.h>

int main()
{
    #ifdef DO_PRINTF
        printf("I am doing PRINTF!!\n");
    #endif
}
```

```bash
C02LN00GFD58:330 hank$ cat conditional_printf.c
#include <stdio.h>

int main()
{
    #ifdef DO_PRINTF
        printf("I am doing PRINTF!!\n");
    #endif
}
```

```bash
C02LN00GFD58:330 hank$ gcc conditional_printf.c
C02LN00GFD58:330 hank$ ./a.out
C02LN00GFD58:330 hank$ gcc -DDO_PRINTF conditional_printf.c
C02LN00GFD58:330 hank$ ./a.out
I am doing PRINTF!!
```

This is how configure/cmake controls the compilation.
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

```c
#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++. 
There is more to macros...

• Macros are powerful & can be used to generate custom code.
  – Beyond what we will do here.
• Two special macros that are useful:
  – __FILE__ and __LINE__

```c
#include <stdio.h>

int main()
{
    printf("This print happens on line %d of file %s\n", __LINE__, __FILE__);
    printf("But this print happens on line %d\n", __LINE__);
}
```

(Do an example with __LINE__, __FILE__)
Outline

• Building Large Projects
• Beginning with C++
• Changes in C++ for structs
• Bonus topics (if time)
Relationship between C and C++

• C++ adds new features to C
  – Increment operator!

• For the most part, C++ is a superset of C
  – A few invalid C++ programs that are valid C programs

• Early C++ “compilers” just converted programs to C
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.
Variable declaration (1/2)

• You can declare variables anywhere with C++!

```cpp
void line_C(double X1, double X2, double Y1, double Y2) {
    double slope;
    double intercept;

    slope = (Y2-Y1)/(X2-X1);
    intercept = Y1-slope*X1;
}

void line_CPP(double X1, double X2, double Y1, double Y2) {
    double slope = (Y2-Y1)/(X2-X1);
    double intercept = Y1-slope*X1;
}
```
Variable declaration (2/2)

- You can declare variables anywhere with C++!

```c
int C_fun(void)
{
    int sum += i;
    ~
    =
}
```

Why is this bad?

```c
C02LN00GFD58:L10  hank$  g++  t.C
```

```
t.C:16:17:  error: invalid '+=' at end of declaration; did you mean '='?
    int sum += i;
       ^
```

```
t.C:18:12:  error: use of undeclared identifier 'sum'
    return sum;
       ^
```

2 errors generated.

```c
int CPP_fun(void)
{
    int sum = 0;
    for (int i = 0 ; i < 10 ; i++)
    {
        sum += i;
    }
    return sum;
}
```

What compiler error would you get?
C-style Comments

/* Here is a single line comment */

/*
   Here is a multi-line comment */

/
   * Here is a
   * multi-line comment
   * that makes it clearer
   * that each line is a
   * comment
   * ... because of the *'s
   */
C++-style comments

// this is a comment

/* this is still a comment */

// this is a
// multi-line C++ comment

When you type “//”, the rest of the line is a comment, whether you want it to be or not.
Valid C program that is not a valid C++ program

• We have now learned enough to spot one (the?) valid C program that is not a valid C++ program
  – ... but it is tricky?
  – ... any thoughts?

```c
int main()
{
    int y = 2;
    int x = 3  // 2 * y;
}
```
The next 6 slides are review slides on “mangling”
Problem with C...

```c
float doubler(float f) { return 2*f; }
#include <stdio.h>

int doubler(int);

int main()
{
    printf("Doubler of 10 is %d\n", doubler(10));
}
```

```
C02LN00GFD58:330  hank$  cat doubler.c
C02LN00GFD58:330  hank$  gcc -c doubler.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...

```
C02LN00GF58:330  hank$ nm doubler.o
00000000000000048  s  EH_frame0
00000000000000000  T  _doubler
00000000000000060  S  _doubler.eh
C02LN00GF58:330  hank$ nm doubler
  doubler.c  doubler_example  doubler_example.o
doubler.o  doubler_example.c  doubler_user.o
C02LN00GF58:330  hank$ nm doubler_example.o
00000000000000068  s  EH_frame0
0000000000000032  s  L__str
U  _doubler
U  _main
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++...
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.

```cpp
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", 
         CIS610::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
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.
Reference vs address of

• References come up when you declare variables.
  – Otherwise “address of”

• int &x = y; // reference ... with the variable
• int *x = &y; // address of ... the value of the variable
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);
}
```

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

```
#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
Different Misc C++ Topic: initialization during declaration using parentheses

```c
#include <stdio.h>

int main()
{
    int x(3);
    printf("X is %d\n", x);
}
```

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

• Building Large Projects
• Beginning with C++
• Changes in C++ for structs
• Bonus topics (if time)
Learning classes via structs

• structs and classes are closely related in C++
• I will lecture today on changes on how “structs in C++” are different than “structs in C”
• ... at the end of the lecture, 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
(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
#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());
}
```

C02LN00GFD58:330 hank$ cat tallycounter.C
C02LN00GFD58:330 hank$ g++ tallycounter.C
C02LN00GFD58:330 hank$ ./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
Method for constructor has same name as struct Constructor is called automatically when object is instantiated (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
More traditional file organization

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”

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

public:
    TallyCounter(int c) {
        count = c;
    }
    ~TallyCounter() {
        delete 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.

```cpp
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();
};
```

- Default constructor
- Parameterized constructor
- Copy constructor
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

```cpp
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

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

    private:
        int count;

    public:
        void Reset();
        int GetCount();
        void IncrementCount();
};
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
make: *** [main.o] Error 1
```

```
128-223-223-72-wireless:TC hank$ 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

We will discuss inheritance next week.
Outline

• Building Large Projects
• Beginning with C++
• Changes in C++ for structs
• Bonus topics (if time)
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)
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!!
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
memcpy

MEMCPY(3) BSD Library Functions Manual MEMCPY(3)

NAME
memcpy -- copy memory area

LIBRARY
Standard C Library (libc, -lc)

SYNOPSIS
#include <string.h>

void *
memcpy(void *restrict dst, const void *restrict src, size_t n);

DESCRIPTION
The memcpy() function copies n bytes from memory area src to memory area
dst. If dst and src overlap, behavior is undefined. Applications in
which dst and src might overlap should use memmove(3) instead.

RETURN VALUES
The memcpy() function returns the original value of dst.