CMSC 313
COMPUTER ORGANIZATION
&
ASSEMBLY LANGUAGE PROGRAMMING

LECTURE 10, FALL 2012

TOPICS TODAY

- C Programming Overview
- C Input/Output

C PROGRAMMING OVERVIEW

Different Kinds of Languages

- **Java is an object-oriented programming (OOP) language**
  - **Problem solving centers on defining classes**
  - **Classes encapsulate data and code**

- **C is a procedural language**
  - **Problem solving centers on functions**
  - **Functions perform a single service**
  - **Data is global or passed to functions as parameters**
  - **No classes**

Adapted from Dennis Frey CMSC 313 Spring 2011

Libraries

Java libraries consist of predefined classes:

- ArrayList, Scanner, Color, Integer

C libraries consists of predefined functions:

- Char/string functions (strcpy, strcmp)
- Math functions (floor, ceil, sin)
- Input/Output functions (printf, scanf)

Adapted from Dennis Frey CMSC 313 Spring 2011

Documentation

On-line C/Unix manual — the “man” command

Description of many C library functions and Unix commands

Usage:

```
man <function name>
man <command name>
```

Examples:

```
man printf
man dir
man –k malloc
man man
```

Adapted from Dennis Frey CMSC 313 Spring 2011

The C Standard

The first standard for C was published by the American National Standards Institute (ANSI) in 1989 and is widely referred to as “ANSI C” (or sometimes C89)

A slightly modified version of the ANSI C standard was adopted in 1990 and is referred to as “C90”. “C89" and "C90" refer to essentially the same language.

In March 2000, ANSI adopted the ISO/IEC 9899:1999 standard. This standard is commonly referred to as C99, and it is the current standard for the C programming language.

The C99 standard is not fully implemented in all versions of C compilers.

Adapted from Dennis Frey CMSC 313 Spring 2011

C99 on GL

The GNU C compiler on the GL systems (gcc versions 4.1.2 & 4.4.5) appears to support several useful C99 features.

These notes include those C99 features supported by gcc on GL since our course use that compiler.

These features will be noted as C99 features when presented.

Hello World

This source code is in a file such as hello.c

```
/*
   file header block comment
*/
#include <stdio.h>

int main() {

   // print the greeting (C99)
   printf( “Hello World\n”);

   return 0;
}
```

Adapted from Dennis Frey CMSC 313 Spring 2011

Compiler Options

We will use `gcc` to compile C programs on GL.

`-c`

Compile only (create a .o file), don’t link (create an executable)

```
gcc -c hello.c
```

`-o filename`

Name the executable filename instead of a.out

```
gcc -o hello hello.c
```

`-Wall`

Report all warnings

```
gcc -Wall hello.c
```

`-ansi`

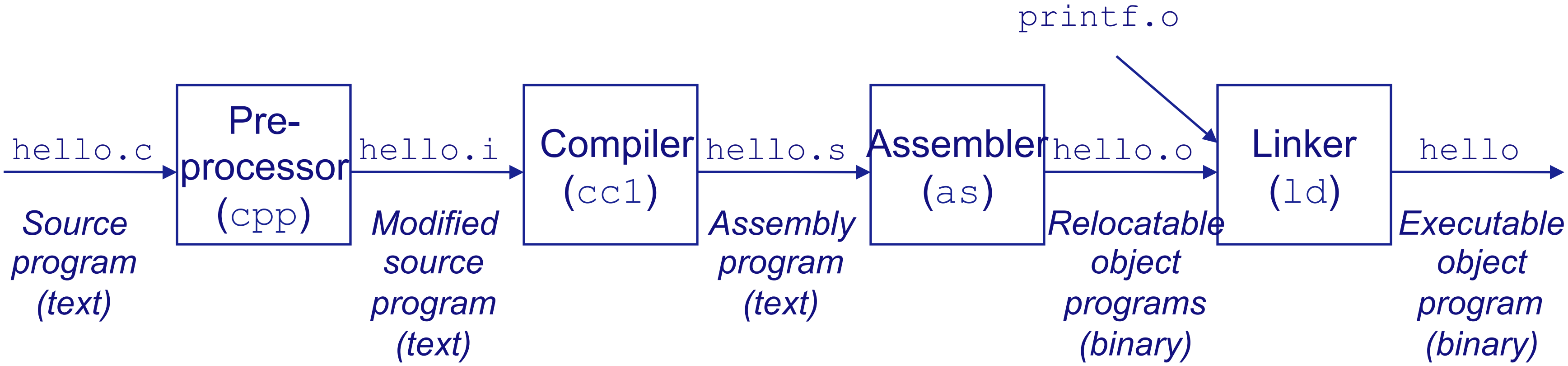
enforces the original ANSI C standard and disables C99 features.

```
gcc -ansi hello.c
```

Adapted from Dennis Frey CMSC 313 Spring 2011

Compiling and Running a C Program

```
unix> gcc -Wall -o hello hello.c
```



Execute your program by typing the name of the executable at the Unix prompt

```
unix> hello
```

Adapted from Dennis Frey CMSC 313 Spring 2011

Language Commonality

- C and Java syntax have much in common
  - Some Data Types
  - Arithmetic operators
  - Logical Operators
  - Control structures
  - Other Operators

- We assume that you are proficient in Java

Adapted from Dennis Frey CMSC 313 Spring 2011

Integral Data Types

- C integer data types:

`int`	(the basic integer data type)
`short int`	(typically abbreviated just as `short`)
`long int`	(typically abbreviated just as `long`)
`long long int`	(C99)
`char`	(C does not have "`byte`")

- mostly use `int`

- use `char` for ASCII

- `char` uses 1 byte

- other sizes system dependent

Adapted from Dennis Frey CMSC 313 Spring 2011

Signed vs Unsigned

- integer types may be signed (default) or unsigned:

```
signed        (positive, negative, or zero)
unsigned      (positive or zero only)
```

- Examples:

```
int age;
signed int age = -33;
long area = 123456;
short int height = 4;
unsigned char IQ = 102;
unsigned int length = 8282;
unsigned long int SATscore = 800;
```

Adapted from Dennis Frey CMSC 313 Spring 2011

Floating Point Data Types

- C floating point types:

`float`	(small)
`double`	(normal)
`long double`	(bigger)

- Examples:

```
float avg = 10.6 ;
double median = 88.54 ;
double homeCost = 10000 ;
```

Adapted from Dennis Frey CMSC 313 Spring 2011

sizeof( )

- C does not specify data sizes.
- `sizeof(`*`type`*`)` returns # of bytes used by *type*.
- Use `sizeof()` for portability.
- On GL,
  - sizeof(short) = 2
  - sizeof(int) = sizeof(long) = 4
  - sizeof (long long) = 8
  - sizeof(float) = 4
  - sizeof(double) = 8

Adapted from Dennis Frey CMSC 313 Spring 2011

const

- Use `const` qualifier to indicate constants:

```
const double PI = 3.1415;
const int myAge = 39;
```

- Compiler complains if code modifies `const` variables.

- `const` variables must be initialized when declared.

Adapted from Dennis Frey CMSC 313 Spring 2011

Variable Declaration

- ANSI C requires that all variables be declared at the beginning of the “block” in which they are defined, before any executable line of code.

- C99 allows variables to be declared anywhere in the code (like Java and C++)

- In any case, variables must be declared before they can be used.

Arithmetic Operators

Arithmetic operators are the same as Java

`=`	(assignment)
`+ -`	(plus, minus)
`* / %`	(times, divide, mod)
`++ --`	(increment, decrement)

Combine with assignment:

```
+= -= *= /= %=
```

Adapted from Dennis Frey CMSC 313 Spring 2011

Boolean Data Type

- ANSI C has no Boolean type
- The C99 standard supports the Boolean data type
- To use `bool`, `true`, and `false`, include `<stdbool.h>`

```
#include <stdbool.h>

bool isRaining = false;
if ( isRaining )
  printf( “Bring your umbrella\n”);
```

Adapted from Dennis Frey CMSC 313 Spring 2011

Type casting

- C provides both implicit and explicit type casting

- Type casting creates value with new type (assuming conversion is possible):

```
int age = 42;
long longAge;
char charAge;

longAge = (long) age; // explicit type cast to long
charAge = age;        // implicit type conversion
```

Adapted from Dennis Frey CMSC 313 Spring 2011

Logical Operators

- Logical operators are the same in C and Java and result in a Boolean value.

`&&`	(and)
`||`	(or)
`== !=`	(equal, not equal)
`< <=`	(less than, less than or equal)
`> >=`	(greater than, greater than or equal)

- Integral types may also be treated as Boolean expressions
  - Zero is considered “false”
  - Any non-zero value is considered “true”

Adapted from Dennis Frey CMSC 313 Spring 2011

Control Structures

Both languages support these control structures which function the same way in C and Java

- `for` loops
  - But NOT: `for (int i = 0; i < size; i++)`
- `while` loops
- `do-while` loops
- `switch` statements
- `if` and `if-else` statements
- `braces` ({, }) are used to begin and end blocks

Other Operators

These other operators are the same in C and Java

- **?: (tri-nary “hook colon”)**

  ```
  int larger = (x > y ? x : y);
  ```

- **<<, >>, &, |, ^ (bit operators)**

- **<<=, >>=, &=, |=,^=**

- **[ ] (brackets for arrays)**

- **( ) parenthesis for functions and type casting**

Arrays

- Array indexing starts with 0.
- ANSI C requires that the size of the array be a constant
- Declaring and initializing arrays

```
int grades[44];
int areas[10] = {1, 2, 3};
long widths[12] = {0};
int IQs[ ] = {120, 121, 99, 154};
```

Adapted from Dennis Frey CMSC 313 Spring 2011

Variable Size Arrays

- C99 allows the size of an array to be a variable

```
int nrStudents = 30;
...
int grades[nrStudents];
```

- Use carefully!!!
- Lifetime = enclosing block.
- Uses lots of stack memory if placed in a loop.
- Not supported by all C compilers.

Adapted from Dennis Frey CMSC 313 Spring 2011

2D Arrays

- Subscripting is provided for each dimension

- For 2D arrays, the first dimension is the number of “rows”, the second is the number of “columns” in each row

```
int board[4][5];      // 4 rows, 5 columns
int x = board[0][0];  // 1st row, 1st column
int y = board[3][4];  // row 4 (last), col 5 (last)
```

Adapted from Dennis Frey CMSC 313 Spring 2011

#define

- `#define` used for macros.

- Preprocessor replaces every instance of the `macro` with the text that it represents.

- Note that there is no terminating semi-colon

```
#define MIN_AGE 21
    ...
 if (myAge > MIN_AGE)
    ...
#define PI 3.1415
    ...
    double area = PI * radius * radius;
    ...
```

Adapted from Dennis Frey CMSC 313 Spring 2011

#define vs const

- #define
  - Pro: no memory is used for the constant
  - Con: cannot be seen when code is compiled since they are removed by the pre-compiler
  - Con: are not real variables and have no type
- const variables
  - Pro: are real variables with a type
  - Pro: can be examined by a debugger
  - Con: take up memory

Adapted from Dennis Frey CMSC 313 Spring 2011

typedefs

- Define new names for existing data types (NOT new data types)

```
typedef int Temperature;
typedef int Row[3];
...
Temperature t ;
Row R ;
```

- Give simple names to complex types.
- `typedefs` make future changes easier.

Adapted from Dennis Frey CMSC 313 Spring 2011

Enumeration Constants

- `enum` = a list of named constant integer values (starting at 0)
- Behave like integers
- Names in `enum` must be distinct
- Better alternative to `#define`
- Example

```
enum months { JAN = 1, FEB, MAR, APR, MAY, JUN, JUL,
  AUG, SEP, OCT, NOV, DEC };
  ...
enum months thisMonth;
thisMonth = SEP;    // preferred usage
thisMonth = 42;     // unfortunately, also ok
```

Adapted from Dennis Frey CMSC 313 Spring 2011

Functions vs. Methods

- Java classes have methods.
- Accessibility of methods controlled by class definition.

- C functions do not belong to any class.
- C functions can have global scope or file scope.
  - global scope = used by anyone
  - file scope = used only by code in same file

- Java methods & C functions both:
  - have a name
  - have a return type
  - may have parameters

Adapted from Dennis Frey CMSC 313 Spring 2011

More Functions

- Function declaration = function *prototype* (aka signature)

```
int add3 (int) ;
```

- Functions must be declared before use.
- Function definition = implementation (code) of function

```
int add3 (int n) {
   return n + 3 ;
}
```

- Function definition also declares the function.
- Functions can be declared in one place and defined (implemented) elsewhere.
- Cannot overload function name in C.

Adapted from Dennis Frey CMSC 313 Spring 2011

A Simple C Program

```
#include <stdio.h>
typedef double Radius;
#define PI 3.1415

/* given the radius, calculates the area of a circle */
double calcCircleArea( Radius radius )
{
   return ( PI * radius * radius );
}

// given the radius, calcs the circumference of a circle
double calcCircumference( Radius radius )
{
   return (2 * PI * radius );
}

int main( )
{
   Radius radius = 4.5;
   double area = circleArea( radius );
   double circumference = calcCircleCircumference( radius );

   // print the results
   return 0;
}
```

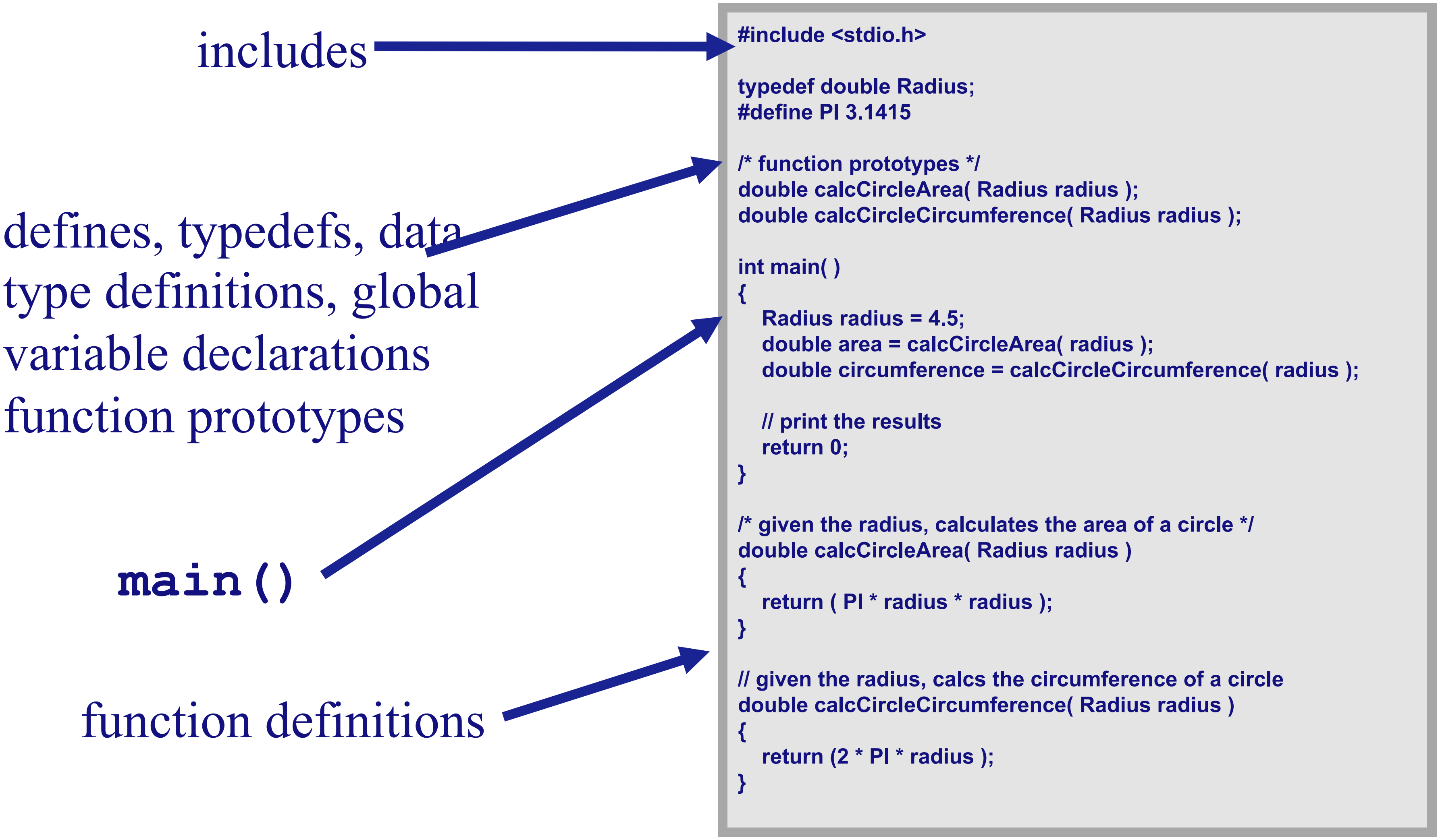
Adapted from Dennis Frey CMSC 313 Spring 2011

Alternate Sample

```c
#include <stdio.h>
typedef double Radius;
#define PI 3.1415

/* function prototypes */
double calcCircleArea( Radius radius );
double calcCircleCircumference( Radius radius );

int main( )
{
    Radius radius = 4.5;
    double area = calcCircleArea( radius );
    double circumference = calcCircleCircumference( radius );

    // print the results
    return 0;
}

/* given the radius, calculates the area of a circle */
double calcCircleArea( Radius radius )
{
    return ( PI * radius * radius );
}

// given the radius, calcs the circumference of a circle
double calcCircleCircumference( Radius radius )
{
    return (2 * PI * radius );
}
```

Adapted from Dennis Frey CMSC 313 Spring 2011

Typical C Program



includes

defines, typedefs, data type definitions, global variable declarations function prototypes

main()

function definitions

```
#include <stdio.h>

typedef double Radius;
#define PI 3.1415

/* function prototypes */
double calcCircleArea( Radius radius );
double calcCircleCircumference( Radius radius );

int main( )
{
   Radius radius = 4.5;
   double area = calcCircleArea( radius );
   double circumference = calcCircleCircumference( radius );

   // print the results
   return 0;
}

/* given the radius, calculates the area of a circle */
double calcCircleArea( Radius radius )
{
   return ( PI * radius * radius );
}

// given the radius, calcs the circumference of a circle
double calcCircleCircumference( Radius radius )
{
   return (2 * PI * radius );
}
```

Adapted from Dennis Frey CMSC 313 Spring 2011

C INPUT/OUTPUT

stdin, stdout, stderr

C opens three input/output devices automatically:

stdin

The “standard input” device, usually your keyboard

stdout

The “standard output” device, usually your monitor

stderr

The “standard error” device, usually your monitor

Some C library I/O functions automatically use these devices

Adapted from Dennis Frey CMSC 313 Fall 2011

Formatted Console Output

- `printf( )` outputs formatted text to `stdout`

```
printf( format, arg1, arg2, … );
```

- Example:

```
int n = 3 ;
printf (“Value = %d\n”, n) ;
```

- `format` is a string containing
  - conversion specifications
  - literals to be printed

Adapted from Dennis Frey CMSC 313 Fall 2011

printf( ) conversions

Conversions specifications begin with % and end with a conversion character.

Between the % and the conversion character MAY be, in order

- A minus sign specifying left-justification
- The minimum field width
- A period separating the field width and precision
- The precision that specifies
  - Maximum characters for a string
  - Number of digits after the decimal for a floating point
  - Minimum number of digits for an integer
- An h for “short” or an l (letter ell) for long

man printf for more documentation.

Adapted from Dennis Frey CMSC 313 Fall 2011

Common printf( ) Conversions

`%d`	print integer as a decimal number (base 10)
`%u`	print integer as unsigned number
`%s`	print string
`%f`	print double as a floating point number
`%x`	print integer in hexadecimal (base 16)
`%c`	print integer as ASCII character
`%p`	print pointer in hexadecimal (implementation dependent)

Adapted from Dennis Frey CMSC 313 Fall 2011

printf( ) Examples

```
int anInt = 5678;
double aDouble = 4.123;
#define NAME “Bob”

/* what is the output from each printf( ) */
printf (“%d is a large number\n”, anInt);
printf (“%8d is a large number\n”, anInt);
printf (“%-8d is a large number\n”, anInt);
printf (“%10.2f is a double\n”, aDouble);
printf( “The sum of %d and %8.4f is %12.2f\n”,
   anInt, aDouble, anInt + aDouble);
printf (“Hello %s\n”, NAME);
```

Adapted from Dennis Frey CMSC 313 Fall 2011

Formatted Output Example

Use field widths to align output in columns

```
int i;
for (i = 1 ; i < 5; i++)
  printf("%2d %10.6f %20.15f\n", i,sqrt(i),sqrt(i));
```

```
12 1234567890 12345678901234567890
 1   1.000000    1.000000000000000
 2   1.414214    1.414213562373095
 3   1.732051    1.732050807568877
 4   2.000000    2.000000000000000
```

Adapted from Dennis Frey CMSC 313 Fall 2011

Keyboard Input

- `scanf` reads user input from `stdin.`
- Syntax for `scanf( )` is similar to `printf( )`
  `scanf( format, arg1, arg2, ... )`
- The format string similar structure to `printf( ).`
- The arguments must be *addresses* of the variables.

Adapted from Dennis Frey CMSC 313 Fall 2011

scanf( ) format string

The `scanf( )` format string usually contains conversion specifications that tell `scanf( )` how to interpret the next "input field". An input field is a string of non-whitespace characters.

The format string usually contains

- Blanks or tabs which are ignored
- Ordinary characters which are expected to match the next (non-whitespace) character input by the user
- Conversion specifications usually consisting
  - % character indicating the beginning of the conversion
  - An optional h, l (ell) or L
  - A conversion character which indicates how the input field is to be interpreted.

Adapted from Dennis Frey CMSC 313 Fall 2011

Common scanf( ) conversions

`%d`	a decimal (integer) number
`%u`	an unsigned decimal (integer) number
`%x`	a hexadecimal number
`%f`	a floating point number with optional sign, decimal point, and exponent
`%s`	a string delimited by white space, NOT an entire line
`%c`	a single character (possibly a whitespace char)

Adapted from Dennis Frey CMSC 313 Fall 2011

scanf( ) examples

```
int age;
double gpa;
char initial;

printf(“ input your middle initial: “);
scanf (“%c”, &initial );        // note &
printf(“Input your age: “);
scanf( “%d”, &age );
printf(“ input your gpa: “);
scanf (“%lf”, &gpa );
```

Adapted from Dennis Frey CMSC 313 Fall 2011

Unix I/O redirection

- Redirect input (read from `infile` instead of keyboard):

  ```
  a.out < infile
  ```

- Redirect output (write to `outfile` instead of screen):

  ```
  a.out > outfile
  ```

- Redirect both:

  ```
  a.out < infile > outfile
  ```

- Redirect `stdout` and `stderr` to `outfile`

  ```
  a.out >& outfile
  ```

- Redirect `stdout` to `outfile` and `stderr` to `errfile`

  ```
  (a.out > outfile) >& errfile
  ```

Adapted from Dennis Frey CMSC 313 Fall 2011

Text File I/O

- Use `fprintf()` and `fscanf()` functions instead of `printf()` and `scanf()`.
- Must open file before reading/writing: `fopen()`
- Must close file after all done: `fclose()`
- Use file handle to specify file.
- File handle returned by `fopen()`:

```
FILE *myFile ;
myFile = fopen (“bob.txt”, “r”) ;
if (myFile == NULL) {
    /* handle the error */
}
```

Adapted from Dennis Frey CMSC 313 Fall 2011

fopen( )

`fopen( )` requires two parameters

1. The name of the text file to be opened
2. The text file open “mode”

`“r”`	open the file for reading only
`“w”`	create the file for writing; delete existing file
`“a”`	append; open or create the file for writing at the end
`“r+”`	open the file for reading and writing
`“w+”`	create the file for reading & writing; deletes existing file
`“a+”`	open or create the file for reading or writing at the end

Adapted from Dennis Frey CMSC 313 Fall 2011

fscanf.c

```
#include <stdio.h>
#include <stdlib.h>  /* for “exit” */
int main ( )
{
  double x ;
  FILE *ifp ;

  /* try to open the file for reading, check if successful */
  /* if it wasn't opened exit gracefully */
  ifp = fopen("test_data.dat", "r") ;
  if (ifp == NULL) {
    printf ("Error opening test_data.dat\n");
    exit (-1);
  }
  fscanf(ifp, "%lf", &x) ;    /* read one double from the file */
  fclose(ifp);       /* close the file when finished  */

  /* check to see what you read */
  printf("x = %.2f\n", x) ;
  return 0;
}
```

Adapted from Dennis Frey CMSC 313 Fall 2011

Detecting end-of-file with fscanf

When reading an unknown number of data elements from a file using `fscanf( )`, we need a way to determine when the file has no more data to read, i.e, we have reached the "end of file".

Fortunately, the return value from `fscanf( )` holds the key. `fscanf( )` returns an integer which is the number of data elements read from the file. If end-of-file is detected the integer return value is the special value `EOF`

Adapted from Dennis Frey CMSC 313 Fall 2011

EOF example code

```
/* code snippet that reads an undetermined number of integer
   student ages from a file and prints them out as an example
   of detecting EOF
*/
FILE *inFile;
int age;

inFile = fopen( “myfile”, “r” );
if (inFile == NULL) {
    printf ("Error opening myFile\n");
    exit (-1);
   }

while ( fscanf(inFile, “%d”, &age ) != EOF ) {
   printf( “%d\n”, age );
}

fclose( inFile );
```

Adapted from Dennis Frey CMSC 313 Fall 2011

fprintf.c

```c
#include <stdio.h>
#include <stdlib.h>  /* exit */
int main ( )
{
  double pi = 3.14159 ;
  FILE *ofp ;

  /* try to open the file for writing, check if successful */
  ofp = fopen("test.out", “w") ;
  if (ofp == NULL) {
    printf ("Error opening test.out\n");
    exit (-1);
  }

  /* write to the file using printf formats */
  fprintf(ofp, “Hello World\n”);
  fprintf(ofp, “PI is defined as %6.5lf\n”, pi);

  fclose(ofp);       /* close the file when finished reading */

  return 0;
}
```

Adapted from Dennis Frey CMSC 313 Fall 2011

NEXT TIME

- Characters & Strings
- Structs