C Language I

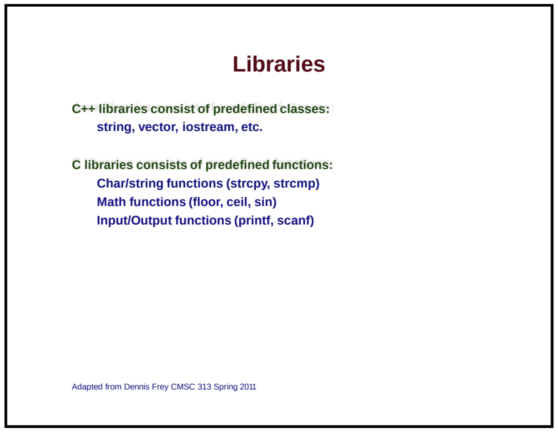
CMSC 313
Sections 01, 02

C Programming Overview

Different Kinds of Languages

- C++ 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

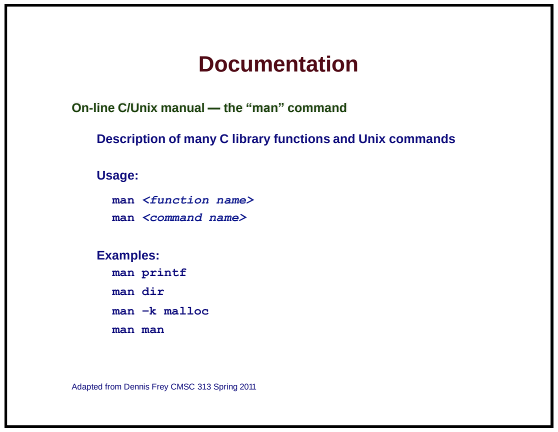


Libraries

C++ libraries consist of predefined classes:
- string, vector, iostream, etc.

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

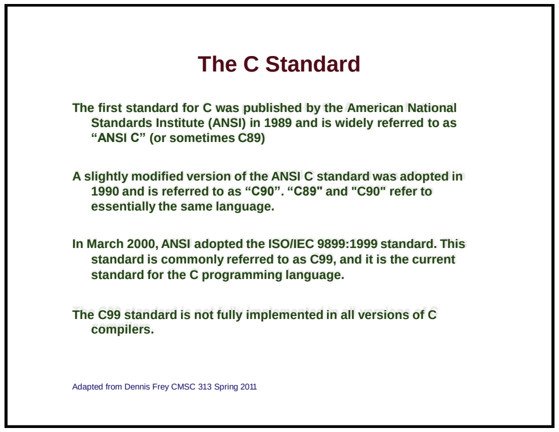
Usage:

```
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 version 4.4) appears to support several useful C99 features.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.

Adapted from Dennis Frey CMSC 313 Spring 2011

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
```

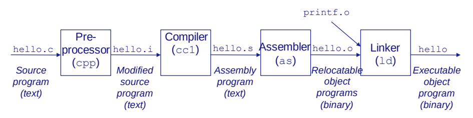
`-Wall`
Report all warnings
```
gcc -Wall hello.c
```

`-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
```

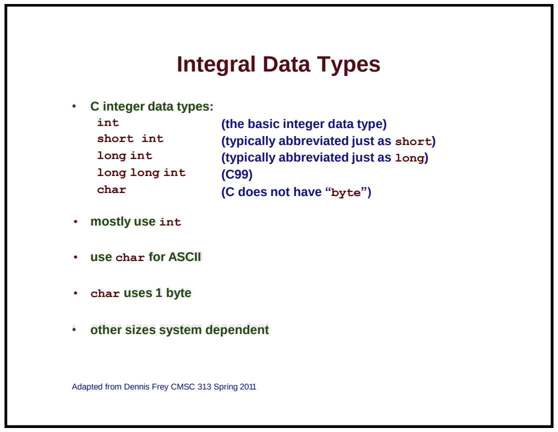
Adapted from Dennis Frey CMSC 313 Spring 2011

C/C++ Comparison

- Most of you know C++ from CMSC 202
- C is essentially C++, but without the classes
- You used to invoke "g++"—now you'll use "gcc"
- Arithmetic and logical operators, control structures, are almost identical
- Except for missing bool, primitive types are the same
- C++ has classes, C has structs

We assume that you have basic knowledge of C++

Whirlwind Review of C/C++



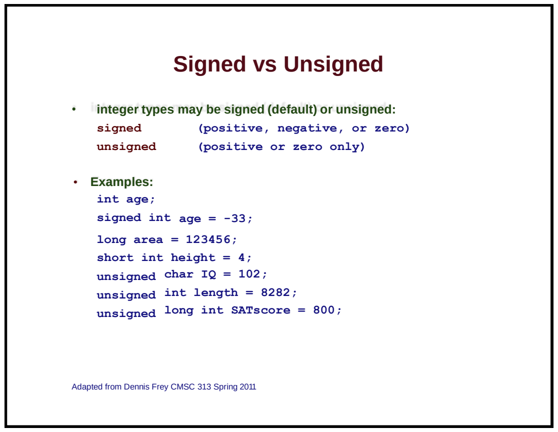
Integral Data Types

- 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

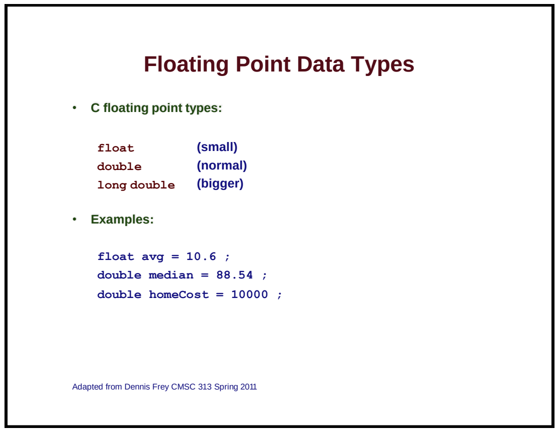
- integer types may be signed (default) or unsigned:

```
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

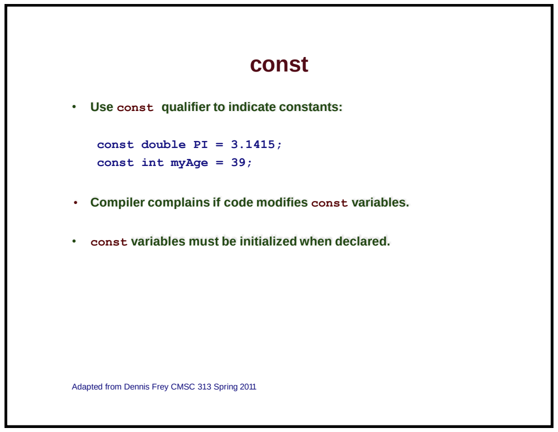
- C floating point types:

`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

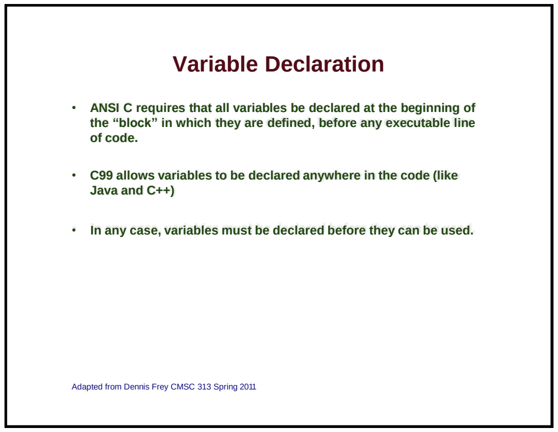


const

- Use `const` qualifier to indicate constants:

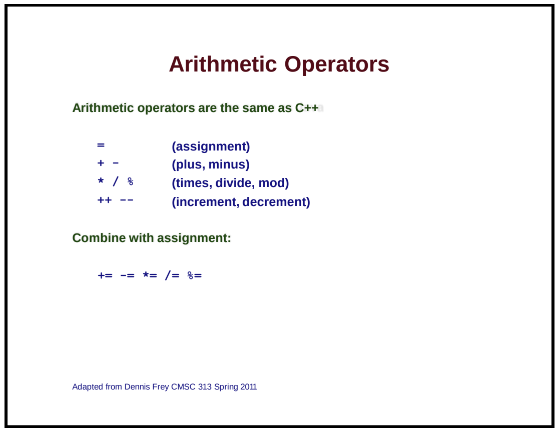
```
const double PI = 3.1415;
const int myAge = 39;
```

- Compiler complains if code modifies `const` variables.
- `const` variables must be initialized when declared.



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.



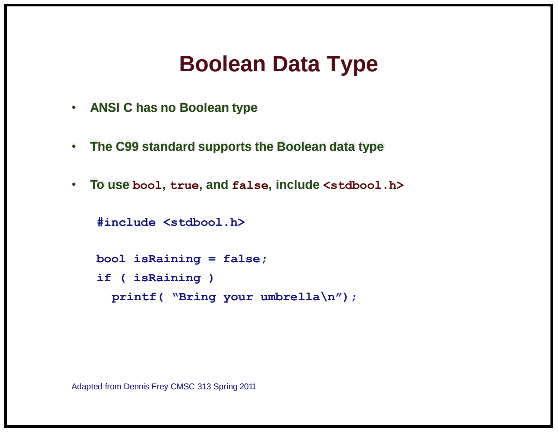
Arithmetic Operators

Arithmetic operators are the same as C++

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

Combine with assignment:

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

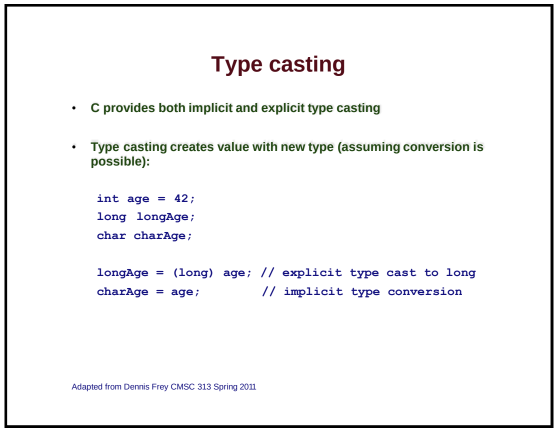


Boolean Data Type

- 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

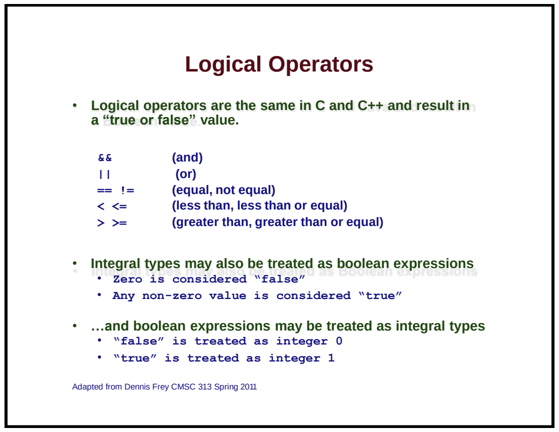


Type casting

- 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



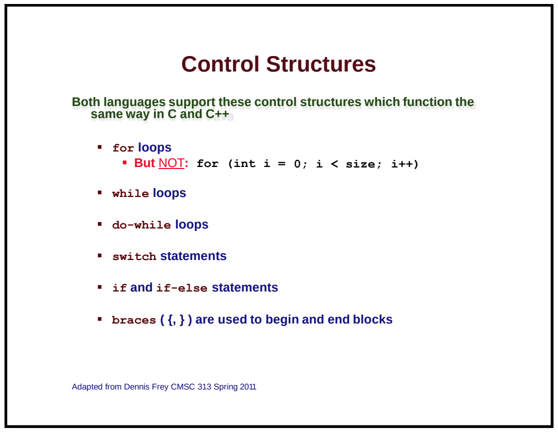
Logical Operators

- Logical operators are the same in C and C++ and result in a "true or false" 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"`
- ...and boolean expressions may be treated as integral types
  - `"false" is treated as integer 0`
  - `"true" is treated as integer 1`

Adapted from Dennis Frey CMSC 313 Spring 2011

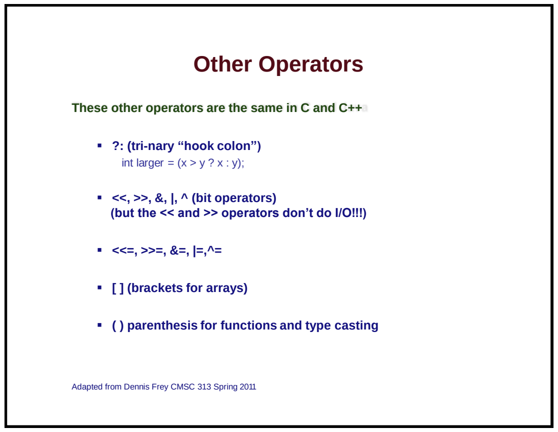


Control Structures

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

- `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

Adapted from Dennis Frey CMSC 313 Spring 2011

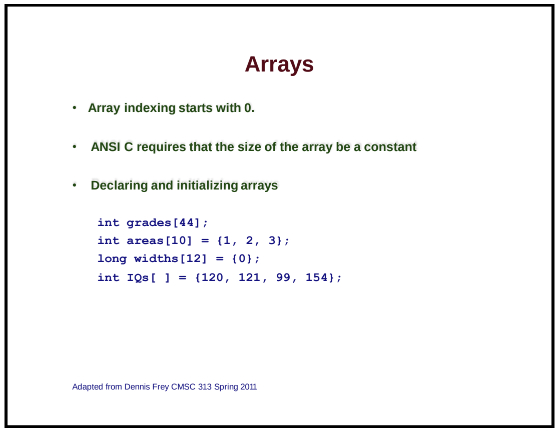


Other Operators

These other operators are the same in C and C++

- ?: (tri-nary "hook colon")
  int larger = (x > y ? x : y);
- <<, >>, &, |, ^ (bit operators)
  (but the << and >> operators don't do I/O!!!)
- <<=, >>=, &=, |=,^=
- [ ] (brackets for arrays)
- ( ) parenthesis for functions and type casting

Adapted from Dennis Frey CMSC 313 Spring 2011

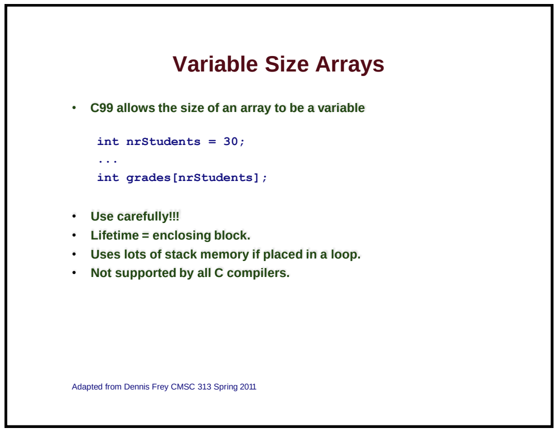


Arrays

- 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



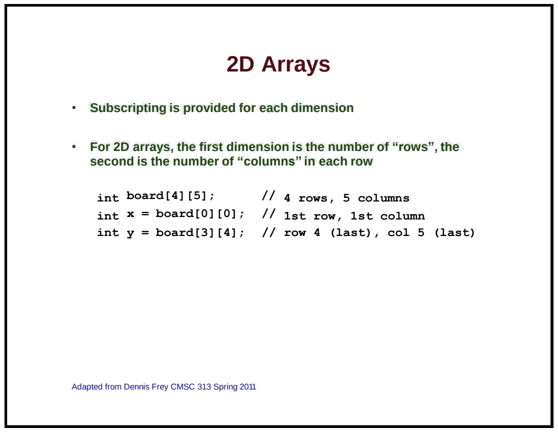
Variable Size Arrays

- 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

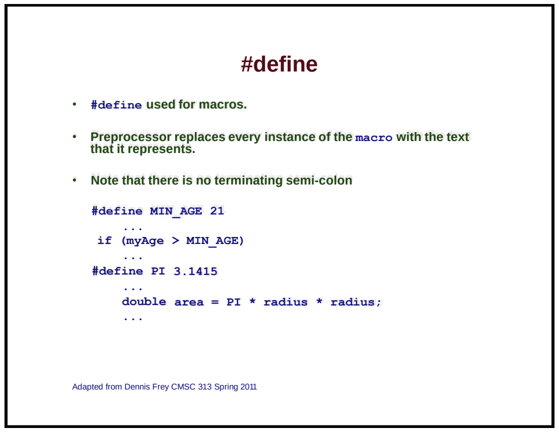


2D Arrays

- 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

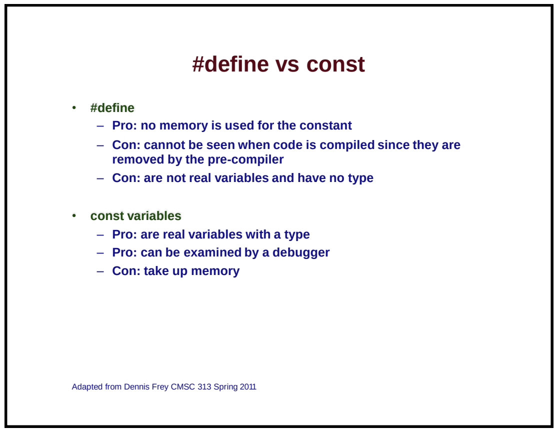


#define

- `#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;
    ...
```

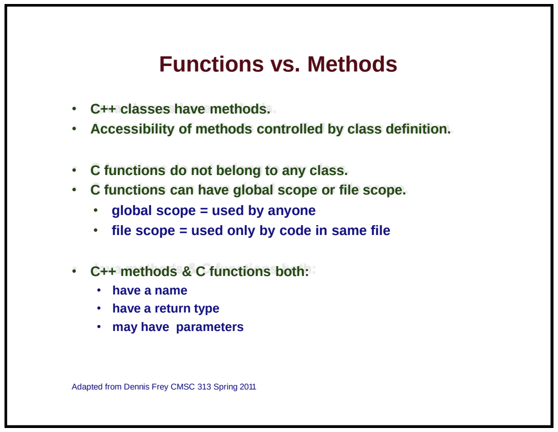
Adapted from Dennis Frey CMSC 313 Spring 2011



#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

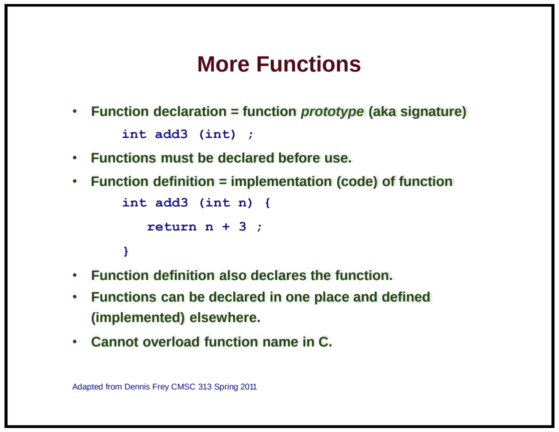
Adapted from Dennis Frey CMSC 313 Spring 2011



Functions vs. Methods

- C++ 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
- C++ 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

Some New (for you) Features

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

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.

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

A Simple C Program

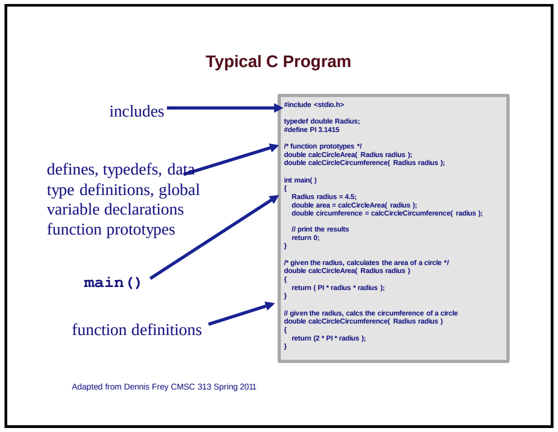
```
#include <stdio.h>
typedef double Radius;
#define PI 3.1415

/* 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

```
#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 =                          radius );
          calcCircleCircumference(
    // 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