CMSC 21
Fundamentals of Programming

2nd Semester 2011-2012

How much do you recall?

REVIEW OF CMSC 11

The Basics

CREATING, COMPILING, AND RUNNING C PROGRAMS

How to create a C Program?

- Create a new file and save it as *filename.c*
- *filename* is your desired name of the file
- The extension *.c* indicates that the file is a C program

How to locate your C program?

- Open a terminal
- To know the current working directory, type pwd (in most systems, the default is `/home/user`)
- Use the cd command to navigate the file system and to find the file
  - type `cd ..` to return to the parent directory
  - type `cd folderName` to go to the subdirectory
- Type `ls` to list all the files within a directory

How to compile your C program?

- The gcc compiler is used in compiling C programs
- Once the file is located, you can now generate an executable file by typing:

```
gcc –o executableFileName fileName.c
```

- If `executableFileName` is not specified, the default file name for the executable file is `a.out`

How to run your program?

- If an executable file is specified, just type

```
./executableFileName
```

- If not, type

```
./a.out
```

Things that must be in your program

BUILDING C PROGRAMS

Preprocessor Command

- Comes at the beginning of the program and begins with a pound (#) sign
- `#include <filename>` or `#include "filename"`
  - Tells the compiler to include a specific library file in the program
  - In order to use the first notation, the file must be in `/usr/include`
  - `#include <stdio.h>` includes the header file `stdio.h` in the program

Preprocessor Command

- `#define`
  - Defines a name for a constant value
  - All subsequent occurrences of that name is replaced with its equivalent value
  - `#define MAXVALUE 1000000` replaces all subsequent occurrences of `MAXVALUE` with `1000000`

`main` Function

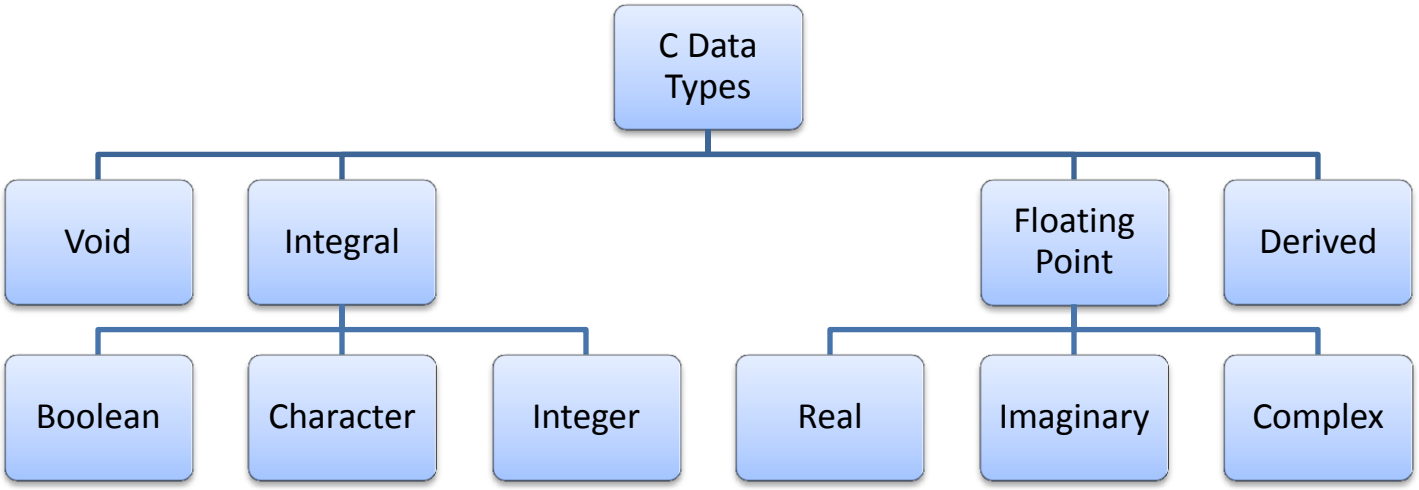
- First function executed when the program is run
- Every C program must have a `main` function
- The program terminates when the execution of the `main` function is finished

Data types, variables, constants…

DATA REPRESENTATION

Data Types

- Set of values and set of operations that can be applied on those values



Data Types

Data Type	Keyword in C	Description
Void	void	No values and operations
Boolean	bool	Stored in memory as 0 (false) or 1 (true)
Character	char	Any value that can be represented in the computer's character set
Integer	int, short int, long int, long long int	A number without the fraction part
Real	float, double, long double	Values with integral and fractional part

Variable Declarations

- Declaring Variables

```
datatype variableName;
```

- Variables can also be initialized upon declaration

```
datatype variableName = value;
```

Variable Declarations

- Multiple variables of the same data type can be declared in one command

```
int x, y = 10, z;
```

- Declaring constant variables

```
const datatype variableName = value;
```

- Values of constants cannot change during the program's execution

How to communicate to the user

INPUT AND OUTPUT

Header File

- stdio.h is the standard input-output library. Include this library to use the built-in input-output functions

```
#include <stdio.h>
```

Format Codes

`%d or %i`	integer value
`%f`	float value
`%c`	character value
`%s`	string
`%lf`	double value

Input: scanf

```
scanf (“format_string”, variable_parameters);
```

- `format_string` – contains the format codes to specify the type/s of input/s to be accepted by the program
- `variable_parameters` – list of varialbes that correspond to the format codes specified on the format_string. Variables of atomic data type should pre preceded by &

Ouput: printf

```
printf (“format_string”, [variable_parameters]);
```

- `format_string` – the string and the format codes of the variables to be printed on the screen
- `variable_parameters` – same as with scanf but variables of atomic data type should not be preceded by &

Operators, Precedence, Expressions and Assignment Statements

DATA PROCESSING

Assignment Statements

- Used to assign values to variables

```
variable_name = expression;
```

- The data type of the `variable_name` should be the same as the data type of the `expression`
- In case the two are different, use typecasting

Typecasting

- Used to force a value into s certain data type
- Example:

```
int x;          //x is an integer
float y;        //y is a float

X = (int) y;    //in order to assign the
          //value of y
                //to x, a typecast must be
          //done
```

Arithmetic Operators

- An expression is a code fragment in C that when evaluated results to a value
- Arithmetic operators are used for expressions that result to integer, floating point or character values

Arithmetic Operators

- Binary Operators

  +, -, *, /(integer/floating point division), %

- Unary Operators

  sizeof, unary plus, unary minus, typecast

- Example:

```
z = x + y;        //binary
+a;               //unary
```

Arithmetic Operators

- Precedence rule in binary operators

  `*, /, %`

  `+, -`

- All are evaluated from left to right

Relational and Logical Operators

- Operators involved in the expressions that result to boolean (true or false) values
- Logical operators

  `&&, ||, !`
- Relational Operators

  `>, <, >=, <=, ==, !=`
- Note: relational Operators have a lower precedence than arithmetic operators

Bitwise Logical Operators

- Perform the operations per bit

`&`	bitwise and
`|`	bitwise inclusive or
`<<`	shift left
`>>`	shift right
`~`	one's complement (unary)

Bitwise Logical Operators

- Example:

  n = 143 (0000 0000 1000 1111 in binary)

  n&0177 0000 0000 1000 1111 &
  0000 0000 0111 1111
  0000 0000 0000 1111 = 15

  n|0177 0000 0000 1000 1111 |
  0000 0000 0111 1111
  0000 0000 1111 1111 = 255

Bitwise Logical Operators

- Example:

  n = 143 (0000 0000 1000 1111 in binary)

  n^0177 0000 0000 1000 1111 ^
  0000 0000 0111 1111
  0000 0000 1111 0000 = 240

  n<<2 0000 0000 1000 1111
  0000 0010 0011 1100 = 572

Bitwise Logical Operators

- Example

  n = 143 (0000 0000 1000 1111 in binary)

  n>>2 0000 0000 1000 1111>>2

  0000 0000 0010 0011 = 35

  ~n ~0000 0000 1000 1111

  1111 1111 0111 0000 = 65392

Increment and Decrement Operators

- Increment operator adds one to the value of the operand

```
++operand; OR operand++;
```

- Decrement operator subtracts one to the value of the operand

```
--operand; OR operand--;
```

Increment Operators

- `operand++` and `++operand` are equivalent to `operand = operand + 1`
- However, if you are to assign each to another variable, the two expressions become different

  `++operand` means increment the operand before using it

  `operand++` means assigning the value of the operand to another variable before incrementing it

Increment Operators

- Example:

```
int x=5, y=3, a, b;
++x;            //means x=x+1=5+1=6
y++;            //means y=y+1=3+1=4
a = x++;        //what will be a?
                //what will be x?
b = ++y;        //what will be b?
                //what will be y?
```

Decrement Operators

- `operand--` and `--operand` are equivalent to `operand = operand - 1`
- However, if you are to assign each to another variable, the two expressions become different

  `++operand` means decrement the operand before using it

  `operand++` means assigning the value of the operand to another variable before decrementing it

Decrement Operators

- Example:

```
int x=5, y=3, a, b;
--x;            //means x=x-1
y--;            //means y=y-1
a = x--;        //what will be a?
                //what will be x?
b = --y;        //what will be b?
                //what will be y?
```

Other Shortcut Operators

The expression

```
variable <operator> = constant
```

is equivalent to

```
variable=variable <operator> constant
```

Example:

```
int x = 4;
x += 4;   //equivalent to x=x+4
```

Notes

- When operands of different types appear in expressions, the types of some sub expressions are converted using some rules:
  1. When integers and floating point values are found in the expression, the integer type is converted automatically to floating point
  2. Chars and ints may be freely mixed in the expression, with the char type automatically converted to the int type

Iterations and Conditional Statements

CONTROL FLOW

Blocks

- A group of declaration and statements enclosed with {}
- A block can be composed of several blocks

```
int main () {
        …
        if (…) { … }
        else { … }
}
```

Program Control Flow

- Sequential
  - Default control flow in C
  - Statements are executed in the order that they are written
- Conditional
  - A statement or a block may not be executed at all
- Iterative
  - A statement or block is executed repeatedly

if-else, switch, ternary operator

CONDITIONAL STATEMENTS

Conditional Statements

- Deal with boolean values (TRUE or FALSE)
- TRUE – numerical non-zero
- FALSE – numerical zero
- Use logical and relational operators

Two-Way Selection

```
if (condition)
    statement1;
else
    statement2;
```

Two-Way Selection

- `condition` determines the execution of statements
- `else` part is optional

Two-Way Selection

- if-else statements can be
  - nested

```
if (condition1) {
   if (condition2) { … }
   else { … }
} else { … }
```

Two-Way Selection

- if-else statements can be
  - ladderized

```
if (condition1) { … }
else if (condition2) { … }
else if (condition3) { … }
```

Two-Way Selection

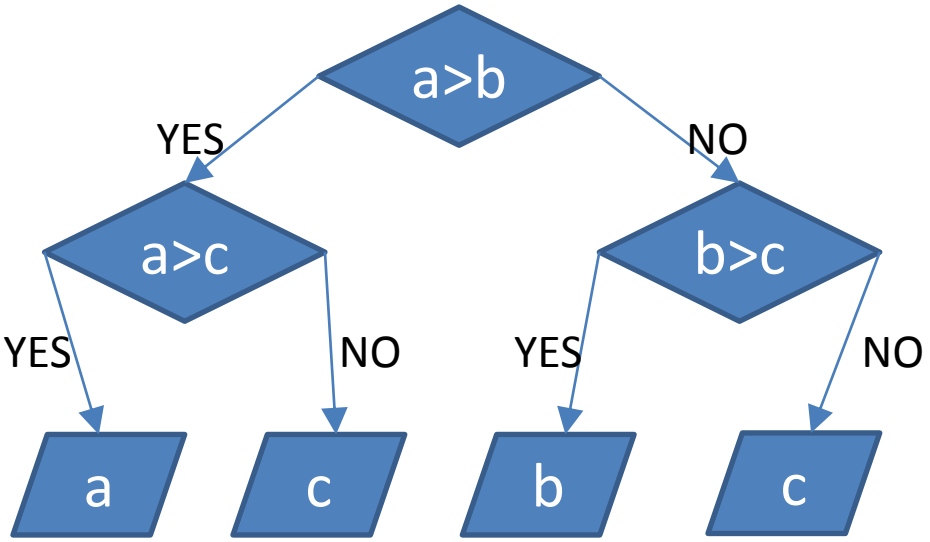
- if-else statements can be
  - created using the ternary operator

```
expression1 ? expression2 : expression3
```

Evaluate `expression1`. If it is TRUE, then the value of the whole expression is evaluated to `expression2`. If FALSE, the expression is evaluated to `expression3`

Two-Way Selection

- Example: Find the maximum among three numbers




```
if (a > b) {
    if (a > c)
        max = a;
    else
        max = b;
} else {
    if (b > c)
        max = b;
    else
        max = c;
}
```

Two-Way Selection

- Example: Find the smaller between two numbers a and b

```
min = (a<b) ? a : b;
```

Multi-Way Selection

- Executes 1 out of n+1 statements
- Has an optional default statement
- Uses a `break` statement
- Relates to ladderized if-else statements

Multi-Way Selection

```
switch (expression) {
   case const1:   statement1;
                  break;
   case const2:   statement2;
                  break;
   …
   case constn:   statementn;
                  break;
   default:       statementn+1;
}
```

Multi-Way Selection

- Example

```
switch (num) {
   case 1:     printf (“one\n”);
               break;
   case 2:     printf (“two”);
               break;
   case 3:     printf (“three”);
               break;
   default:    printf (“%d”, num);
}
```

Multi-Way Selection

- Another example

```
switch (num) {
   case 1:
   case 3:
   case 5:     printf (“odd\n”);
               break;
   case 2:
   case 4:     printf (“even\n”);
               break;
   default:    printf (“%d”, num);
}
```

Iterative Statements

- Statements that are executed repeatedly
- Can be
  - Test before (while loop)
  - Test after (do-while loop)
  - Indexed (for loop)

while Loop

```
while (condition)
   statement

i=0;
while (i>=2) {
   printf (“Hello %d!\n”, i);
   i-=2;
}
```

do-while Loop

```
do
   statement;
while (condition);

i=10;
do {
   printf (“Hello %d!\n”, n);
   i-=2;
} while (i>=2);
```

`for` Loop

```
for (initialization; condition; update)
     statement;
```

```
   for (i=10; i>=2; i-=2)
     printf ("Hello %d\n", i);
```

`break` and `continue`

- `break` – exit the loop
- `continue` – ignore other statements and proceed to the next iteration