Review of CMSC 11

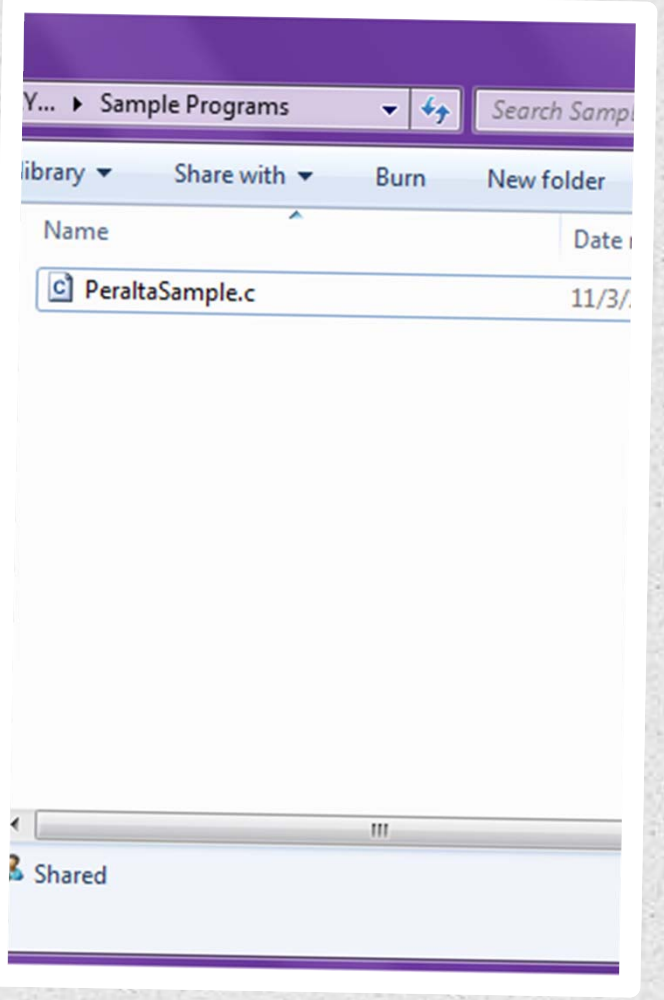
What do you remember?

Some Basics

Compiling and executing C programs.

Creating your C Program

Save your program as *filename.c*, where your desired filename replaces *filename*.



Locating your C Program

- Open the Terminal
- The default working directory should be /home/user
- Use the cd command to navigate the file system and find your file
- To return to the parent directory, use

  cd ..
- To move to a subdirectory, use

  cd *foldername*

Compiling your C Program

- To compile C programs, use the gcc compiler
- To generate an executable file from a C program, type

  gcc -o *executablename programname.c*
- If *executablename* is not specified, the default filename for the executable file is a.out

Running your Program

- To run your program, type

  ./executablename

Building Your Program

The things that should never be absent from your program.

Preprocessor Commands (1)

- #include<*filename*> or #include "*filename*"
  - Includes the specifed file in the program
  - In order to use the first notation, file must be in /usr/include
  - Ex. #include<stdio.h> includes the header file stdio.h in the program

Preprocessor Commands (2)

- #define
  - Defines a name for a constant value. All subsequent occurrences of that name is replaced with its equivalent value
  - Ex. #define DIVISOR 2.0
    - All subsequent occurrences of DIVISOR will be replaced with 2.0

The `main` Function

- First function executed when a program is run
- Every C program must have one
- When the main function ends, the program ends

Data Representation

Data types, variables, constants, etc…

Data Types

- Integer
- Float
- Double
- Character

Variable Declarations (1)

- Declaring variables

  datatype variablename;

- Variables can also be initialized upon declaration

  datatype variablename = value;

Variable Declarations (2)

- Multiply variables of the same data type can be declared in one command, for example,

  int x, y = 10, z;

- Declaring constant variables

  const *datatype variablename* = *value*;

  - Values of constant values can not change during the duration a program's execution

Input and Output

How to ask the user for input and how to respond to the user.

Header Files

- To use built-in input and output functions, include the header file stdio.h

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

Format Codes

- %d or %i – integer value
- %f – float value
- %c – char value
- %s – string
- %lf – double value

Output: `printf` (1)

- The format of the printf function is

  printf("*formatstring*", *argumentlist*);

- Parameters
  - Format String: Contains the string to be outputted to the user as well as format codes to insert variable values in the string
  - Argument List: List of variables that correspond to format codes specified in the format string

Input: scanf (1)

- The format of the input function is

  scanf("*formatstring*", *argumentlist*);

- Parameters:
  - Format String: Mostly contains format codes to determine what data type the input will be stored as.
  - Argument List: Same as the argument list for printf. However, variables of atomic data types should be preceded by an &.

Output: `printf` (2)

- Examples
  - printf(“Hello world!”);
  - printf(“Your age is %d.”, age);
    - Since %d indicates an integer value is to be inserted, the variable age should be of data type int

Input: scanf (2)

- Examples
  - scanf("%d", &age);
  - scanf("%c", &gender);
  - scanf("%s", name);
    - The format code %s indicates a string data type which is not atomic (it can be broken down to characters)

Data Processing

Arithmetic and other operations

Assignment Statements

- Used to assign values to variables

  identifier = *expression*;

- Identifier = variable name
- Data type of Identifier = Data type of expression result
- If data type is not the same, use typecasting

Typecasting

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

```
int x; //x is an integer
float y; //y is a floating-point number

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

Arithmetic Operations

- Addition
- Subtraction
- Multiplication
- Division
- Modulus

Integer Division (1)

- Occurs when the divisor is an integer
- Fractional part is truncated
- Use a float constant or variable, or typecast the divisor to float to avoid this

Integer Division (2)

- Example
  - 10 / 3 = 3
  - 10 / 4 = 2
  - 1 / 2 = 0
  - 10 / 3.0 = 3.333333333
  - 10 / 4.0 = 2.5

Increment Operation

- Increment
  - Increasing the value of a variable by 1
  - Notation: *variable*++ is the same as

    variable = *variable* + 1
  - Can also be expressed as ++*variable*

Decrement Operation

- Decrement
  - Decreasing the value of a variable by 1
  - Notation: variable-- is the same as

    variable = *variable* – 1
  - Can also be expressed as --variable

Before or After?

- The position of the ++ or -- can affect the precedence of operators in an expression
- Example

```
x = 7
y = x++;
y = ?
```

```
x = 7
y = ++x;
y = ?
```

Other Shortcut Operators

- The expression

  variable <operator>= constant;

  is equivalent to

  variable = variable <operator> constant;

- Examples
  - n = n * 2 ⟷ n *= 2
- Other operators:
  - /=, -=, +=

Nested Statements

- Operations can be nested within each other
- Possible nested statements:
  - Increment/decrement within an assignment statement
  - Assignment statement within another assignment statement
  - Arithmetic operations within relational operations
  - Arithmetic operations within input/output calls
  - Assignment statements within relational operations
  - And so on…

Rules of Precedence

- +, -, *, /, %
  - Highest: *, /, %
  - Lowest: +, -
- To override precedence rules, use parentheses

Control Structures

Controlling the flow of execution of a program

Types of Control Flow

- Sequential
  - Default control flow
  - Statements executed in order
- Conditional/Selection
  - May cause a statement/group of statements to be skipped subject to certain conditions
- Iterative
  - A statement/group of statements are executed repeatedly subject to certain conditions

Conditional Statements (1)

- Condition is in the form of a BOOLEAN EXPRESSION
- Condition evaluated to either TRUE or FALSE
  - TRUE = non-zero value
  - FALSE = zero
- Rules for precedence in expressions apply
- Employ parentheses to clarify expressions and override precedence

Conditional Statements (2)

- Code blocks
  - Grouped statements enclosed in curly braces

Types of Selection

- Two-Way
  - `if-else` statements
  - ternary operators
- Multi-Way
  - `switch` statements
  - `if-else-if` statements

Logical Operators - NOT

- Unary operator: ! x
- Inverts the truth value of the operand

x	!x
false	true
true	false

Logical Operators - AND

- Binary operator: x && y
- Results in true if BOTH x and y are true
- Otherwise, results in false

x	y	x && y
true	true	true
true	false	false
false	true	false
false	false	false

Logical Operators - OR

- Binary operator: x || y
- Results in false if both operands are false
- Otherwise, results in true

x	y	x && y
true	true	true
true	false	true
false	true	true
false	false	false

Evaluating Logical Expressions

- Two general methods
  1. Evaluate the expression completely in order to get the result
  2. Stop evaluating the expression once the result is known for sure
     - C uses this method

Quiz (1/4)

- Evaluate the following logical expression. (True, False, or Cannot Be Determined) Given x=10, y=30, z=17.

```
((x%2==0&&x%5==0)||x>z)&&
((y%3==0&&y %5==0)||y>z)
```

`if-else` Statements (1)

- Syntax

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

- Code blocks can be used instead of statements
- The else part is optional

`if-else` Statements (2)

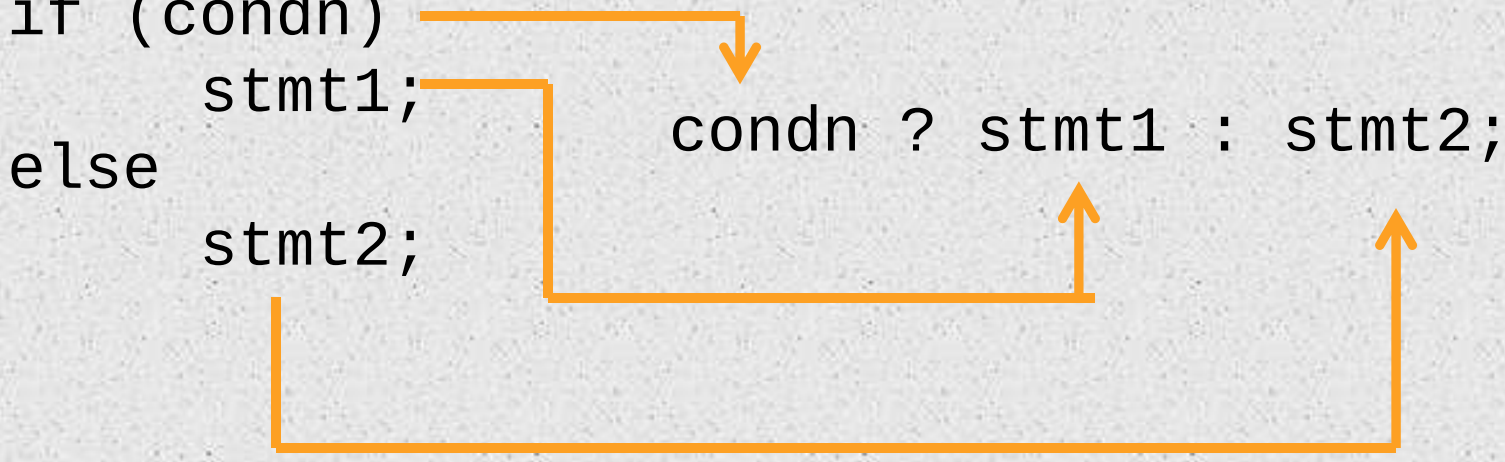
- An if part can exist without an else partner, but an else part can not exist without a corresponding if
- A statement/code block might not be executed depending on the result of the condition
- Condition = TRUE, if clause is executed
- Otherwise, else clause is executed

Ternary Operators

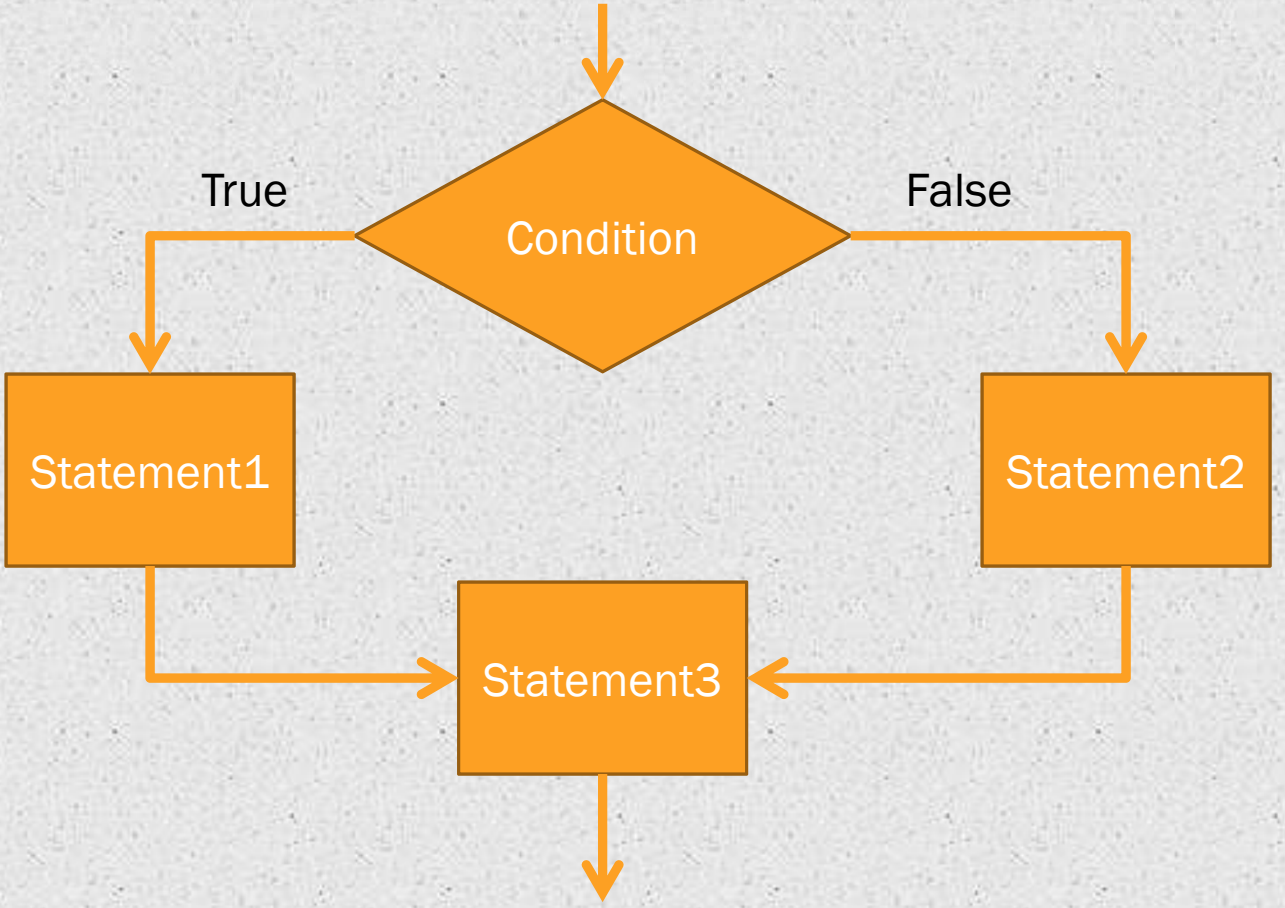
- A shorter version of the if-else statement

```
if (condn)
      stmt1;
else
      stmt2;
```

```
condn ? stmt1 : stmt2;
```



Flowchart for Two-Way Selection



Remarks

- If-else statements may be nested
  - If-else statement can be included within another if-else statement
  - May be hard to understand when the number of levels > 3
- Avoid dangling else situations
  - Else clause is not paired with an if clause
  - Use curly braces to clarify your code
  - Use indentation for more readable code

`if-else-if` Statements (1)

- The else clause may also have a condition

```
if (condition1)
    statement1;
else if (condition2)
    statement2;
```

- Cascading if-else-if statements have numerous else-if clauses

if-else-if Statements (2)

```
if (condition1)
    statement1;
else if (condition2)
    statement2;
else if (condition3)
    statement3;
else if(condition4)
    statement4;
...
```

switch Statements (1)

- Syntax

```
switch (expression) {
    case const1: statement1; break;
    case const2: statement2; break;
    case const3: statement3; break;
    ...
    case constn: statementn; break;
    default: statementn-1;
}
```

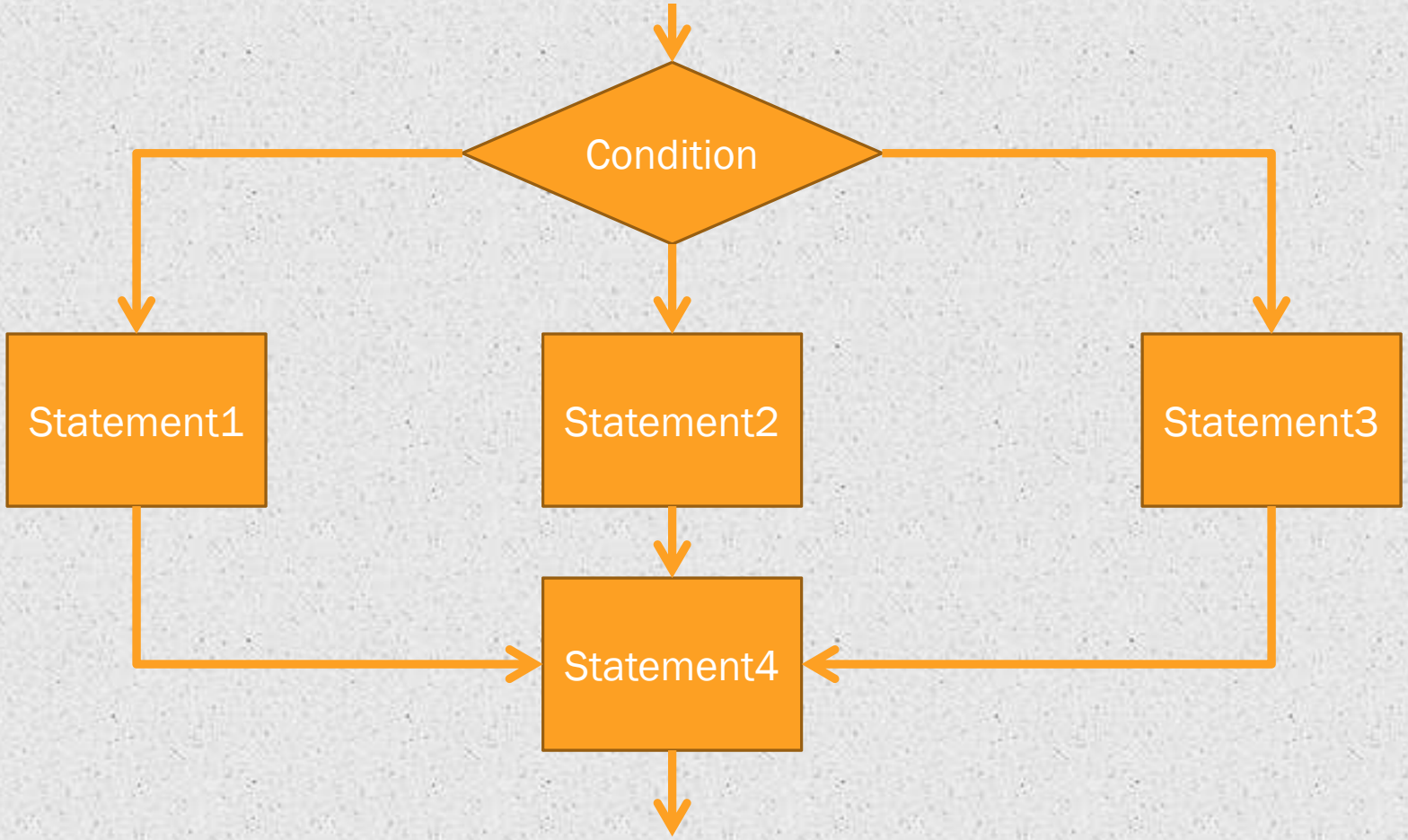
`switch` Statements (2)

- Result of expression compared with each constant case
  - If equal, execute statement starting at that point
  - Once a case has been matched, execution of statements will not terminate until
    - a break statement is reached
    - the end of the `switch` statement is reached
- Similar to `if-else-if` statements

`switch` Statements (3)

- Can only be used if the expression evaluates to either an integer or character
- Constants in cases can only be of data type int or char
- Cases are the possible values of the expression
- Default case is used when none of the other cases matches with the expression's result
  - Default case is OPTIONAL

Flowchart for Multi-Way Selection



Quiz (1/4)

- What will the output of the following code be if `score = 7`?

```
switch(score) {
    case 10:
    case 9: grade = ‘A’;
    case 8: grade = ‘B’;
    case 7: grade = ‘C’;
    case 6: grade = ‘D’
    default: grade = ‘F’;
}
```

Quiz (1/4)

- What will the output of the following code be if `score = 7`?

```
switch(score) {
    case 10:
    case 9: grade = ‘A’; break;
    case 8: grade = ‘B’; break;
    case 7: grade = ‘C’; break;
    case 6: grade = ‘D’ break;
    default: grade = ‘F’;
}
```