Advanced Programming
COEN 11

Lecture 1

C - Overview

- C Structure
- Control Structures
- Modular Programming with Functions

C Structure - Overview

- program structure
- constants and variables
- assignment statements
- standard input and output
- library functions
- system limitations

Program Structure: General Form

```
preprocessing directives

int main (void)
{
   declarations
   statements
}
```

Program Structure

- Every C program contains one function named main
- The body of the each function is enclosed by braces, { }

Program Structure

- Comments
  - Across lines
    - begin with the characters /*
    - end with the characters */
  - Same line
    - // starts the comment which ends at the end of the line

Program Structure

- Preprocessor directives give instructions to the compiler

```
#include <stdio.h>
#define CONSTANT constant_value
```

Program Structure

- Functions contain two types of commands
  - Declarations and statements
    - End with a semicolon (;)
- Preprocessor directives
  - Do NOT end with a semicolon
- To exit the program
  - Use a return 0; statement

Program Structure: First Program

```
/*****************************************************************/
/*  Program 1 – Yi Fang– September 2017                          */
/*                                                               */
/*  This program computes the sum two numbers                    */
/*****************************************************************/

#include <stdio.h>

int main (void)
{
     //  Declare and initialize variables
     double number1 = 1.234, number2 = 5.678, sum;

     //  Calculate sum
     sum = number1 + number2;

     //  Print the sum
     printf ("The sum is %f\n", sum);

     //  Exit program
     return 0;
}
```

Constants and Variables

- A constant is a specific value
- A variable is a memory location that is assigned a name or an identifier
  - A variable is associated with a data type
  - Variables must be declared before the variable can be used

Variables

- An identifier is used to reference a memory location.
- Rules for selecting a valid identifier
  - must begin with an alphabetic character or underscore
  - may contain only letters, digits and underscore (no special characters)
  - case sensitive
  - cannot use keywords as identifiers

C Data Types

- Integers
  - short: 16 bits
  - int: depends on the machine
  - long: 32 bits
- Floating-Point Values
  - float: 32 bits
  - double: 64 bits
  - long double: depends on the compiler
- Characters
  - char: 1 byte

Symbolic Constants

- Defined with a preprocessor directive
- Compiler replaces each occurrence of the directive identifier with the constant value in all statements that follow the directive
- Example
  - #define PI 3.141593

Assignment Statements

- Used to assign a value to a variable
- General Form:

  identifier = expression;
- Example 1

  double sum = 0;
- Example 2

  int x;

  x=5;

Assignment Statements

- Example 3

```
char ch;
ch = 'a';
```

- Example 4

```
int x, y, z;
x = y = 0;
z = 2;
```

- Example 5

```
y = x + z;
```

Arithmetic Operators

- Addition +
- Subtraction -
- Multiplication *
- Division /
- Modulus %
  - Modulus returns remainder of division between two integers
  - Example
    - 5%2 returns a value of 1

Integer Division

- Division between two integers results in an integer.
- The result is truncated, not rounded
- Example:
  - 5/3 is equal to 1
  - 3/6 is equal to 0

Precedence of Operators

- Parentheses Inner most first
- Unary operators Right to left
  - (+ -)
- Binary operators Left to right
  - (* / %)
- Binary operators Left to right
  - (+ -)

Increment and Decrement Operators

- Increment Operator ++
  - post increment x++;
  - pre increment ++x;
- Decrement Operator --
  - post decrement x--;
  - pre decrement --x;

Abbreviated Assignment Operator

- operator statement — example — equivalent
  - += — x+=2; — x=x+2;
  - -= — x-=2; — x=x-2;
  - *= — x*=y; — x=x*y;
  - /= — x/=y; — x=x/y;
  - %= — x%=y; — x=x%y;

Standard Output

- printf Function
  - prints information to the screen
  - requires one or more arguments
    - Req: control string with conversion specifiers
    - Opt: values that correspond to the specifiers in the control string
- Example

```
float angle = 45.5;
printf("Angle = %.2f degrees \n", angle);
```

- Output

```
Angle = 45.50 degrees
```

Standard Input

- scanf Function
  - inputs values from the keyboard
  - required arguments
    - control string with conversion specifiers
    - memory locations that correspond to the specifiers in the control string
- Example:

```
int distance;
char unit_length;
scanf("%d%c", &distance, &unit_length);
```

Standard Input

- scanf function
  - It is very important to use a specifier that is appropriate for the data type of the variable

Standard Input and Output

- Examples of specifiers
  - int – %d, %i
  - short – %hd, %hi
  - long – %ld, %li
  - char – %c
  - float – %f
  - double - %lf

Library Functions

Math Functions

- fabs(x)
  - Absolute value of x.
- sqrt(x)
  - Square root of x, where x>=0.
- pow(x,y)
  - Exponentiation, x^y.
- ceil(x)
  - Rounds x to the nearest integer toward ∞ (infinity).
  - Example, ceil(2.01) is equal to 3.

Math Functions

- floor(x)
  - Rounds x to the nearest integer toward -∞ (negative infinity).
  - Example, floor(2.01) is equal to 2.
- exp(x)
  - Computes the value of e^x.
- log(x)
  - Returns ln x, the natural logarithm to the base e.
  - Errors occur if x<=0.
- log10(x)
  - Returns $\log_{10}x$, logarithm to the base 10.
  - Errors occur if x<=0.

Character Functions

- toupper(ch)
  - If ch is a lowercase letter, this function returns the corresponding uppercase letter
  - otherwise, it returns ch
- isdigit(ch)
  - Returns a nonzero value if ch is a decimal digit
  - otherwise, it returns a zero.
- islower(ch)
  - Returns a nonzero value if ch is a lowercase letter
  - otherwise, it returns a zero.

Character Functions

- isupper(ch)
  - Returns a nonzero value if ch is an uppercase letter;
  - otherwise, it returns a zero.
- isalpha(ch)
  - Returns a nonzero value if ch is an uppercase letter or a lowercase letter
  - otherwise, it returns a zero.
- isalnum(ch)
  - Returns a nonzero value if ch is an alphabetic character or a numeric digit
  - otherwise, it returns a zero.

System Limitations

System Limitations

- SHRT_MAX
- INT_MAX
- LONG_MAX
- FLT_DIG
- FLT_MAX_10_EXP
- FLT_MAX

Control Structures

Control Structures - Overview

- algorithm development
  - conditional expressions
  - selection statements
  - loop structures

Algorithm Development

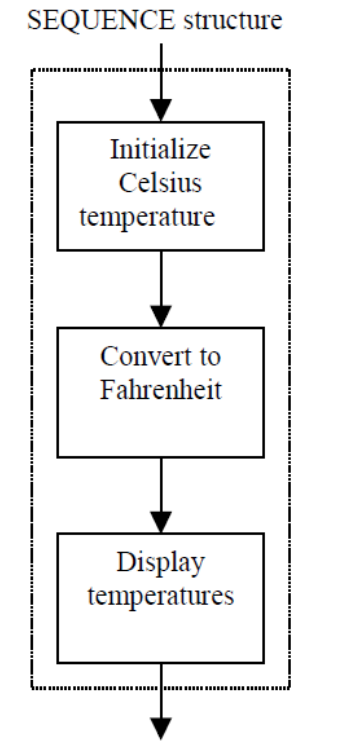
Structured Programming

- Sequence
- Selection
- Repetition

Sequence - example

```
…
int m, x, y;
scanf ("%d%d", &x, &y);
m = x * y;
printf ("Multiplication is %d\n", m);
…
```

Flowchart Example




```
x = 10;

y = 9*x/5+32;

DispTemp(x,y);
```

Selection Statements

- if
- if else
- if else if
- switch

If statement

```
if (condition)          //single statement
        statement;
```

```
if (condition)          //more than one statement
{
   statement 1;
        ...
   statement n;
}
```

If statement - examples

```
if (x > 0)
    k++;
```

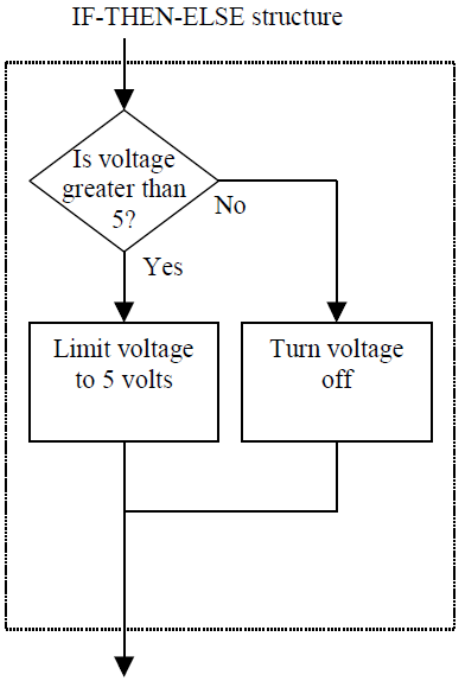
```
if (x > 0)
{
    k++;
    j--;
}
```

if - else statement

```
if (condition)
    statement 1;
else
    statement 2;
```

```
if (condition)
{
    statement block 1
}
else
{
    statement block 2
}
```

Flowchart Example




```
if ( x > 5 )
{
      x = 5;
}
else
{
      x = 0;
}
```

if – else-if statement

```
if (condition1)
   statement 1;
else if (condition2)
   statement 2;
else if (condition3)
        statement 3;
else
   statement 4;
```

if – else - if statement

```
if (condition1)
{
    statement block 1
}
else if (condition2)
{
    statement block 2
}
else
{
    statement block 3
}
```

nested if-else

```
if (x > y)
   if (y < z)
        a++;
   else
        b++;
else
   c++;
```

Practice!

```
int x=10, y=9, z=8, a=0, b=0, c=0;

if (x > y)
    if (y < z)
            a++;
    else
            b++;
else
    c++;
```

What are the values of a, b and c?

Boolean (or Conditional) Expressions

- result is 0 or 1
  - 1 is used as true
  - 0 is used as false
- use relational and logical operators

Relational Operators

- == equality
- != non equality
- < less than
- > greater than
- <= less than equal to
- >= greater than equal to

Logical Operators

- ! not
- && and
- || or

Operator Precedence

- ( )
- !
- < <= > >=
- == !=
- &&
- ||

Switch Statement

```
switch (expression)
{
    case constant:
        statement(s);
        break;
    case constant:
        statement(s);
        break;
    default:            // default is optional
        statement(s);
}
```

Switch Statement

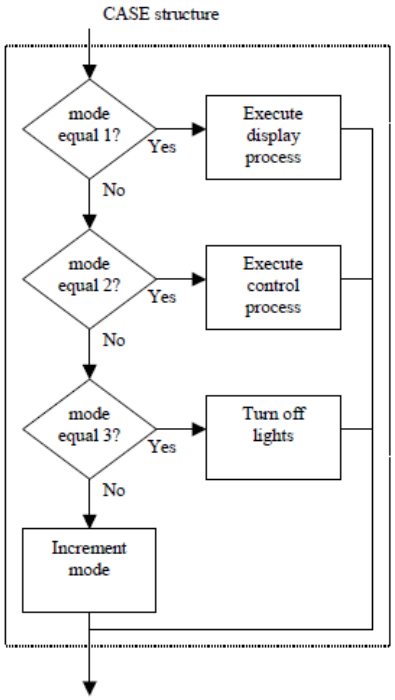
- Expression must be of type integer or character
- The keyword case must be followed by a constant
- break statement is required unless you want all subsequent statements to be executed.

Practice!

- Convert the following if/else statement to a switch statement:

```
if (choice == 1)
        printf ("First Choice\n");
else if (choice == 2)
          printf ("Second Choice\n");
else if (choice == 3)
            printf ("Third Choice\n");
else
         printf ("Default Choice\n");
```

Flowchart Example




```
switch (mode) {
case 1:
      /* Display process */
      break;
case 2:
      /* Control process */
      break;
case 3:
      red_light = 0;
      blue_light = 0;
      break;
default:
      mode = mode + 1;
      break;
}
```

Structured Programming

- Repetition

Loop

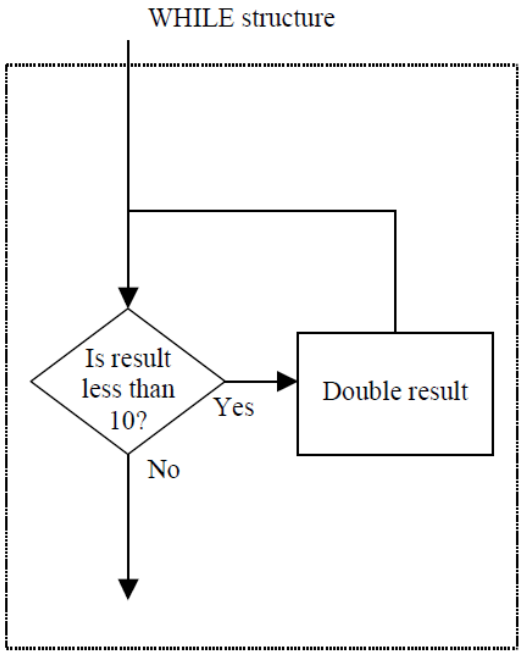
- while statement
- do while statement
- for statement

while statement

```
while (condition)
        statement;
```

```
while (condition)
{
        statement 1;
        ...
        statement n;
}
```

Flowchart Example




```
while ( x < 10 )
{
      x = 2*x;
}
```

while statement - examples

```
int   sum = 0;
int   i = 1;

while (i < 10)
{
    sum = sum + i;
    i += 2;
}
```

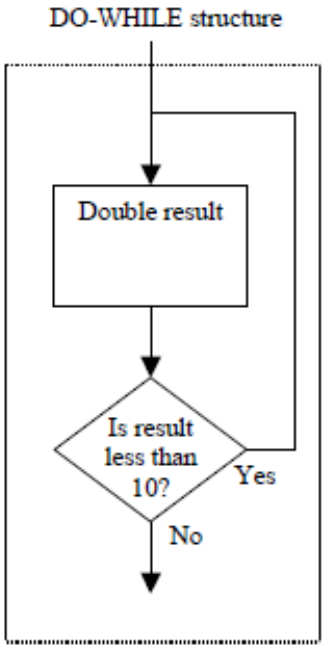
do while statement

```
do
        statement;
while (expression);
```

```
do
{
        statement 1;
        ...
        statement n;
} while (condition);
```

- note - the expression is tested *after* the statement(s) are executed, so statements are executed *at least once*.

Flowchart Example




```
do
{
        x = 2*x;
}
while ( x < 10 );
```

for statement

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

```
for (initialization; condition; update)
{
        statement 1;
        ...
        statement n;
}
```

for statement - examples

```
int    sum = 0;
int    i;
for (i = 1; i < 10; i += 2)
   sum = sum + i;
```

```
int    fact = 1;
int    n;
for (n = 5; n > 1; n--)
   fact = fact * n;
```

break statement

- break;
  - terminates loop
  - execution continues with the first statement following the loop

continue statement

- continue;
  - forces next iteration of the loop, skipping any remaining statements in the loop

Example

```
for(i=0;i<=5;i++)
  {
     if(i==3){
     break;
     }
     printf("%d\n",i)
  }
```

Modular Programming with Functions

Modularity

- Execution of a program begins in the main function
- The main function can call other functions
  - Functions defined in the same file
  - Functions defined in other files or libraries
- Functions are also referred to as modules
  - A module is a set of statements that performs a task or computes a value

Advantages of using modules

- Modules can be written and tested separately
- Large projects can be developed in parallel
- Reduces length of program, making it more readable
- Promotes the concept of abstraction

Functions

- What do functions do?
  - Perform a task
  - May also
    - Return a single value to the calling function
    - Change value of the function arguments

Functions

- Pre-defined
  - standard libraries
- Programmer defined

Example

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

int
main (void)
{
    double        x, y;

    scanf ("%lf%lf", &x, &y);

    printf ("%lf\n", pow (x, y));
}
```

Function Terminology

- Function prototype or declaration
  - Describes how a function is called
    - the types of the arguments received
    - the type of the value returned
- Function calls
  - Specify where in the code each function is executed with actual parameters

Function Terminology

- Function definition
  - Code for the actual function
  - Defined with
    - Formal parameters
      - must match with actual parameters in order, number, and data type
    - Returned value

Functions: Value Returned

- Function returns a single value to the calling program
- Function definition declares the type of value to be returned
- A return (expression); statement is required in the function definition

Example – function defintion

```
int fact (int);
...
int
fact (int n)
{
    int   fact = 1;
    while (n > 1)
    {
          fact = fact * n;
          n--;
    }
    return (fact);
}
```

Prototype or declaration

Type of the value returned

Parameters received

Value returned

void Functions

- A void function may be called to
  - perform a particular task
  - modify data
  - perform input and output
- A void function does not return a value to the calling program
- A return; statement is used
  - no value is returned

Example of void function definition

```
void
print_date (int mo, int day, int year)
{
   /*output formatted date */
    printf("%i-%i-%i\n", mo , day , year );
   return;
}
```

Parameter Passing

- Call by value
  - formal parameter receives the value of the actual parameter
  - functions cannot change the value of the actual parameter
- Call by reference
  - actual parameters are addresses

Scope

- Scope
  - refers to the portion of the program in which it is valid to reference a function or a variable
- Storage class
  - refers to the lifetime of a variable

Scope

- Local scope
  - a local variable is declared within a function or a block and can be accessed only within the function or block that declares it
- Global scope
  - a global variable is declared outside the functions and can be accessed by any function within the program file

Lifetime

- Local variables
  - Generally only active while the function in which it was declared is active
- Global variables
  - Active throughout the execution of the program

Exception

- Static local variables
  - Local scope
  - Global lifetime