EC-ENG 231 (Spring 2024)

C Programming Part 1

Fatima Anwar
fanwar@umass.edu
UMass Amherst

A lightning introduction to C programming:

- This is a ~2 lecture “crash course” in C programming

- **Prerequisite:** familiarity with a high-level programming language such as Python. Understanding of basic programming concepts such as variables, loops, if statements, representing values in different bases, etc…

- We won’t get through all the slides in this crash course. But you’ll have them for reference

- **Book and Tutorial**:
  - The C book: https://publications.gbdirect.co.uk//c_book/ also can be found here: https://github.com/wardvanwanrooij/thecbook
  - C Tutorial: C made easy https://www.cprogramming.com/tutorial.html

easily-understandable, human readable instructions

Higher-level

machine details abstracted away

Python

Java

Translation: source code file interpreted every time program runs, into machine-understandable code of the host machine

C

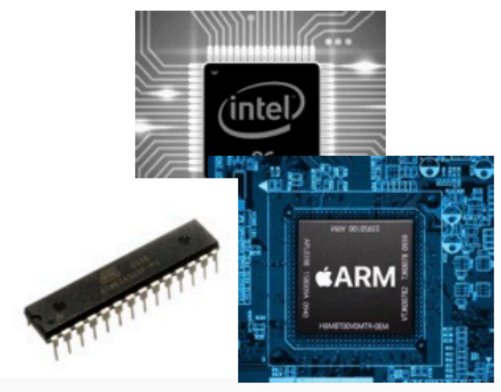
access to memory & other machine-level features

Translation: Source code compiled into a new executable machine code file.

Assembly

Lower-level

machine code(binary/hex) understood by the CPU of the host machine



Interesting Facts about C

- C++ is a superset/extension of C

- What is written in C? “System code” & applications:
  - Interpreters, operating systems (parts of Windows, macOS, Linux, iOS, etc..)
  - Interpreted languages: Python, Ruby, …
  - Embedded processor control
  - DSP processors

C Features

- **Access to memory locations, control over processor & hardware** (different from Java, Python, which “hide” the machine)
- Portable code – run on different machines. Common source code compiles down to code understandable by a specified target machine (CPU)
- **Efficient – few keywords, small code size, runs fast**
- Modular code – can effectively build larger programs from modules with encapsulated details
- **External standard library – handles I/O, math, string manipulations, and other capabilities**
- “Never doubts the programmer” – bugs in your code can result in strange behaviors; contrast higher languages that deliberately reduce impact of simple mistakes
- “C is quirky, flawed, and an enormous success.” - Ritchie

Embedded Development Tools & Environment

- Integrated Development Environments (IDE's) – 10's GB disk space
  - Examples:
    - Visual Studio (Microsoft, Windows 10 & MacOS)
    - Xcode (Apple, MacOS)
    - Eclipse
  - Integrated text editor with compiler, debugger
  - Convenient for large programs, steep learning curve

- IDEs for Embedded Programming
  - Microchip Studio, MPLABX (~10 GB)– AVR & PIC MCUS
  - Arduino 500-800 MB (for hobbyists, rapid prototypes, artists, etc…)

- Lighter weight option:
  - text editor, toolchain, command-line
    - less disk space
    - less time spent learning the software tool, more time learning to code
    - better understanding of what's actually happening to code

Programming Environment for C

- Editor:
  - Visual Studio Code (Win, macOS)

- Command line interface/shell:
  - Terminal (macOS)
  - Command Prompt (Windows)

- Toolchains (compiler & supporting bin utils)
  - gcc compiler (for use with console) - to compile code to run on your computers
    - clang (macOS) >> gcc --version, clang --version
    - MinGWx64 (Windows 10) >> gcc -v

Online C Compiler

ONLINE CPP

C

main.c

```c
/*****************************************************
hello_main.c  -- this program prints "Hello World!"
to the display monitor.
Compile using gcc
Date        Author       Revision
02/05/2024  Fatima Anwar initial version
*****************************************************/

#include <stdio.h>   //  We need this to use the printf function

int main(void){

    printf("Hello World!\n");
    return 0;
}
```

Ln: 15, Col: 2

Run | Share | Command Line Arguments

```
Hello World!


** Process exited - Return Code: 0 **
```

https://www.online-cpp.com/online_c_ide

C Program Structure

what does a C program look like?

what are the key parts?

```c
/*****************************************************
hello_main.c  -- this program prints "Hello World!"
to the display monitor.
Compile using gcc

Date        Author       Revision
02/05/2024  Fatima Anwar initial version
*****************************************************/

#include <stdio.h>  //  We need this to use the printf function

int main(void){

    printf("Hello World!\n");
    return 0;
}

/************* end of file ************************/
```

BLOCK COMMENT

Preprocessor Directive #

Main Function
Every C program should have it
“int”as return value

output & input from the **standard I/O** devices (monitor, keyboard). These are function calls. Requires <stdio.h>

Programming Style

```c
/*****************************************************
hello_main.c  -- this program prints "Hello World!"
to the display monitor.
Compile using gcc

Date        Author       Revision
02/05/2024  Fatima Anwar initial version
*****************************************************/

#include <stdio.h>   //  We need this to use the printf function

int main(void){

    printf("Hello World!\n");
    return 0;
}

/************ end of file ************************/
```

programming style matters.
/* Comments */
{...} // Comments
white space

```c
#include <stdio.h>
int main(void){printf("Hi World!\n");return 0;}
```

Same code without white space

Don’t Do This!

Structure of a C program

```
/* about this program*/
#include
#define
function declarations;
```

block comments
pre-processor directives &
function declarations

```
int main(void)
{ statement;
statement;
...
return 0; }
```

main function header
{main function definition}

```
function a()
{ statement;
statement;
...
return; }
```

function a header
{function a
definition}

```
function b()
{ statement;
statement;
...
return; }
```

function b header
{function b definition}

4 types of C statements:

declaration
assignment
function call
control

Structure of a C program: FUNCTION

```
/* A function comment states what the function does,
not how it does it. Typically describe function inputs &
output */

return_type function_name(arg1, arg2, ...){
        statement;
        statement;
        statement;
        return  (something);
}
```

function definition

function header – specifies function name, return type, # & types of arguments

function body

```
#include <stdio.h>

Function A();
Function B();

int main() {
    A();
    B();
}
```

examples of function headers:
int main(void)
float square_root(float)
char answer()

All C programs are required to have a main() function. Program execution starts here.

Function return type

Function Declaration

```c
#include <stdio.h>

/* Tell the compiler that we intend to use a function called show_message.
 * It has no arguments and returns no value. This is the "declaration" */
void show_message(void);

/* Another function, but this includes the body of
 * the function. This is a "definition" */
int main() {
  int count;
  count = 0;
  while(count < 10){
    show_message();
    count = count + 1;
  }
  return 0;
}

/* The body of the simple function. This is now a "definition" */
void show_message(void) {
    printf("hello\n");
}
```

Input Arguments

Function Definition

```c
/*****************************************
Arithmetic library

Date          Author        Revision
02/05/2024    Fatima Anwar  initial ver
*****************************************
#include <stdio.h>
// Function to perform addition
int add(int a, int b) {
    return a + b;
}
// Function to perform subtraction
int subtract (int a, int b) {
    return a - b;
}
// Function to perform multiplication
int multiply(int a, int b) {
    return a * b;
}
// Function to perform division
float divide (int a, int b) {
    if (b != 0) {
        return (float)a / b;
    } else {
        printf("Error: Division by ze
        return 0;
    }
}
// Even/Odd check function
int isEven (int number) {
    return number % 2 == 0;
}
```

```c
int main() {

    int num1, num2;

    printf("Enter two numbers: ");
    scanf("%d %d", &num1, &num2);
    printf("Sum: %d\n", add(num1, num2));
    printf("Difference: %d\n", subtract(num1, num2));
    printf("Product: %d\n", multiply(num1, num2));
    printf("Quotient: %.2f\n", divide (num1, num2));
    if (isEven (num1)) {
        printf("The first number is even\n");
    }
    else {
        printf("The first number is odd\n");
    }
    return 0;
}
```

```
fatimas-mbp:C_programs fatimanwar$ ./a.out
Enter two numbers: -5 -7
Sum: -12
Difference: 2
Product: 35
Quotient: 0.71
The first number is odd
```

C Program Structure - Summary

- All C programs must have a main() function
- Preprocessor directives begin with #. They are not C statements
- You need to declare the names of things (variables, functions, structures, …) before you can use them
- Types of statements in C that end with a semicolon ;
  - Function call: `printf("hello\n"); square(5.5);`
  - Declaration: `int x;`
  - Assignment: `x=x+1;`
  - Control: `if (x > 100) printf("Happy Birthday\n");`
- Coding w/o comments is cruel. It's allowed in C. But not in ECE-231
- Use good names for variables, functions, etc…. (`int x` vs `int current_time`)
- Function declaration:

```
int square(int);
```

- Function definition:

```
int square(int x){
    return (x*x);
}
```

Data Types and Declarations

Variables and Constants
Types of data
Size of data

Names of functions and variables

- Requirements:
  - Names must be declared before they are used
    - `int x;` before `x=..`
    - `int myFunction(int);` before `x=myFunc(5);`
  - Names are case sensitive: myage, myAge, MYAGE are all different
  - First character must be _ or letter
  - May not assign a C keyword (eg, int, float, struct, return, etc…)

- Good programming style:
  - Use descriptive names
  - Don’t begin with _ to avoid confusion with special identifiers
  - Use ALLCAPS for symbolic constants (eg, #define PI 3.14)

Data types in C

- numeric
  - <u>int</u>eger (4, -36, -9909) – no fractional part
  - <u>float</u>ing point (98.6, 6.02e-23) – decimal point

- <u>char</u>acters
  - ‘a’, ‘b’ etc... also small integers -128, -127, ...0, 1, ... 127

- arrays of integers, floats, characters (strings)

- 4, -36, 09909, 98.6, 6.02e-23. ‘a’, ‘z’ are all called literals.
  (Sometimes also called “constants” since they have fixed value in a program)

Variables and Constants

```
#define PI 3.14                  // Macro or symbolic constant
int x,y;                         // Int variables declared
float z;                         // Float variable declared
int x=50;                        // Int declared & initialized
float z = 98.6                   // Float declared & initialized
char a = 'G';                    // Declare, init a as char
const int x = 50;                // Declare, init x as constant int
short int a,b,c;                 // Declare some short ints
unsigned int d,e;                // Declare some unsigned ints
char my[] = "Hi There";          // Declare & initialize char array
```

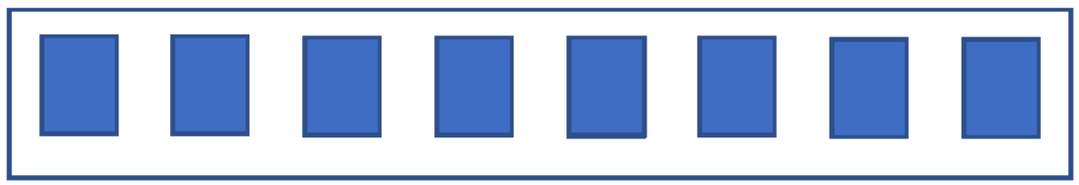
```c
/*****************************************************
data_types.c  -- this program prints different data
types to display monitor.

Date          Author        Revision
02/05/2024    Fatima Anwar  initial version
*****************************************************/

#include <stdio.h>

int main(void){

    char ch = 'A';
    int i = 0;
    float f = 1.1;
    double ff = 3.14159;

    printf("ch = %c, i = %d\n", ch, i);
    printf("f = %f, ff=%.15f\n", f, ff);

    return 0;
}
/************* end of file ************************/
```

```
fatimas-mbp:C_programs fatimanwar$ ./a.out
ch = A, i = 0
f = 1.100000, ff=3.141590000000000
```

Integer declaration with initialization

```
int x=100;

signed int x= -100;

unsigned int x=100;

short int x= -100;

unsigned short int x=100;

long x= -100;

unsigned long x=100;
```

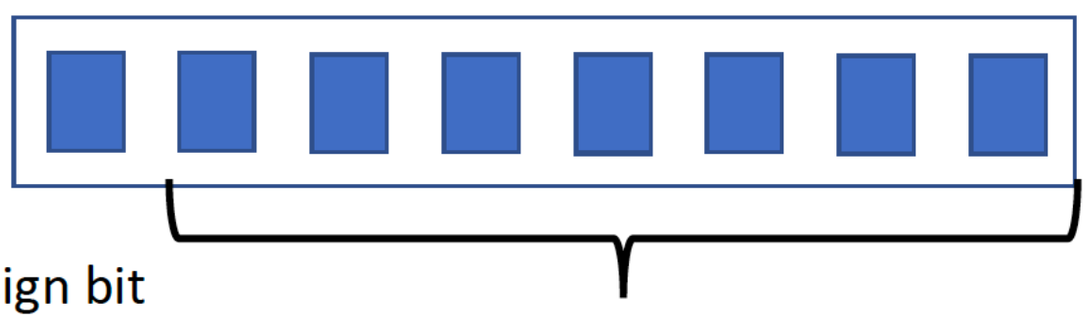
8Bits = 1 byte; 2^8=256 combinations (representations)



unsigned binary representation

an unsigned 8 bit int can represent: 0 to 255

a signed 8 bit int can represent: -128 to + 127



2's compliment representation for signed binary

sign bit
1=negative
0=positive

7 bits for magnitude
2^7 = 128

Integer Types

1 byte; 8 bits; 2^8 = 256 different values

- signed: -128 to +127
- unsigned: 0 to 255

2 bytes; 16 bits; 2^{16} = 65,636

- signed: -32,768 to +32,767
- unsigned: 0 to 65, 535

4 bytes: 32 bits; 2^{32} = 4,294,967,296

8 bytes: 64 bits; 2^{64} = 1.844674×10^{19}

standard types

char
signed char
unsigned char
short int
signed short int
unsigned short int
signed int
unsigned int
long int
signed long int
unsigned long int

different widths (bytes) & sign representations means different min & max values.

Type range and formatting

Data type	Size(bytes)	Range	Format String
char	1	-128 to 127	%c
unsigned char	1	0 to 255	%c
short	2	-32,768 to 32,767	%d
unsigned short	2	0 to 65535	%u
int	2	32,768 to 32,767	%d
unsigned int	2	0 to 65535	%u
long	4	-2147483648 to +2147483647	%ld
Unsinged long	4	0 to 4294967295	%lu
float	4	-3.4e-38 to +3.4e-38	%f
double	8	1.7 e-308 to 1.7 e+308	% lf
long double	10	3.4 e-4932 to 1.1 e+4932	%lf

Use `sizeof` function call to find the size of a variable, variable type, and literal for a particular C installation e.g. `sizeof(double)`

```c
/*****************************************************
This program records one time value and prints in
seconds and nanoseconds

Date          Author        Revision
02/05/2024    Fatima Anwar  initial version
*****************************************************/
#include<stdio.h>
#include<time.h>

int main(){
    struct timespec now1, now2;

    clock_gettime(CLOCK_REALTIME, &now1);
    clock_gettime(CLOCK_MONOTONIC, &now2);

    printf("Current real time is: %lu sec and %lu nanoseconds\n", now1.tv_sec, now1.tv_nsec);
    printf("Current monotonic time is: %lu sec and %lu nanoseconds\n", now2.tv_sec,
now2.tv_nsec);
}
```

```c
struct timespec {
    time_t tv_sec;
    long tv_nsec;
};
```

```
fatimas-mbp:C_programs fatimanwar$ ./a.out
Current real time is: 1707273943 sec and 200157000 nanoseconds
Current monotonic time is: 30382 sec and 291603000 nanoseconds
```

why do we care?

```
long double temperature[1000];   // Declares an array of 1000 long doubles.  8kB if
                                 // each long double is 8 bytes wide

short int temperature[1000];     // Declares an array of 1000 short ints.
                                 // 1kB if each short int is 1 byte wide
```

your laptop has GB of RAM
your ATmega328P chip has 2kB RAM

the issue: different compilers are free to set their own widths, provided widths follow a general specification:
short ≤ int ≤ long; int ≥ 16 bits; long ≥ 32 bits, etc...

C Operators

Arithmetic
Relational
Logical
Bitwise

Arithmetic Operators

- =, +, -, *, / // Just what you'd expect
- % // Modulo 4%2 is 0; 4%3 is 1

- Examples:

```
a = b+50;      // Set a = result of b+50
c = a*b;       // Set c = a*b
x = x+1;       // Increment x
x++;           // Shorthand for x=x+1
x = x-1;       // Decrement x
x--;           // Shorthand for x=x-1
a = 4/2;       // a = 2
a = 4%2;       // a = 0
a = 4%3;       // a = 1
```

Shorthand Arithmetic Operators

```
x++;        // x=x+1;
x--;        // x=x-1;
x+=50;      // x=x+50;
x-=a;       // x=x-a;
x*=a;       // x=x*a;
x/=a;       // x=x/a;
x%=a;       // x=x%a;
y=x++;      // y=x; x=x+1; postfix increment
y=x--;      // y=x; x=x-1; postfix decrement
y=++x;      // x=x+1; x=y; prefix increment
y=--x;      // x=x-1; x=y; prefix increment
```

Practice Exercises

```c
#include <stdio.h>

int main() {
  int a,b;
  a = b = 5;
  printf("%d\n", ++a+5);
  printf("%d\n", a);
  printf("%d\n", b++ +5);
  printf("%d\n", b);
  return 0;
}
```

```c
#include <stdio.h>

#define BOILING 212     /* degrees Fahrenheit */

int main() {
  float f_var; double d_var; long double l_d_var;
  int i;
  i = 0;
  printf("Fahrenheit to Centigrade\n");
  while(i <= BOILING){
    l_d_var = 5*(i-32);
    l_d_var = l_d_var/9;
    d_var = l_d_var;
    f_var = l_d_var;
    printf("%d %f %f %Lf\n", i, f_var, d_var, l_d_var);
    i = i+1;
  }
  return 0;
}
```

Mixed type Arithmetic

```
int n_int=5
int d_int=2;
int q_int = n_int/d_int;          // 5/2 = 2
int m_int = n_int%d_int;          // 5%2 = 1 (modulo)

float n_float = 2.0;
float d_float = 5.0;
float q_float = n_float/d_float;  // 5.0/2.0 = 2.5

q_int = n_float / d_float;        // 5.0/2.0=2.5; q_int = 2

// Implicit type conversion:
q_float = n_int/d_float;          // 5/2.0 = 5.0/2.0 = 2.5

// Explicit type conversion:
q_float = (float) n_int/d_float   // 5.0/2 = 5.0/2.0 = 2.5
```

```c
/*****************************************************
This program measures time difference of a sleep
event

Date          Author        Revision
02/05/2024    Fatima Anwar  initial version
*****************************************************/

#include<stdio.h>
#include <unistd.h>
#include<time.h>

int main(){
    struct timespec start;
    struct timespec end;
    long elapsed_time_in_nanosec;
    long elapsed_nanos;
    double elapsed_time_in_sec;
    long elapsed_time;

    clock_gettime(CLOCK_REALTIME, &start);
    sleep(5); // sleeps for specified seconds
    clock_gettime(CLOCK_REALTIME, &end);

    elapsed_nanos = end.tv_nsec - start.tv_nsec;
    elapsed_time_in_nanosec = (end.tv_sec - start.tv_sec)*1000000000 + elapsed_nanos;
    elapsed_time_in_sec =  elapsed_time_in_nanosec / 1000000000.0;
    elapsed_time = (long) elapsed_time_in_sec;
    printf("Elapsed time is: %lf seconds and %lu nanoseconds\n", elapsed_time_in_sec, elapsed_nanos);
    printf("Elapsed time is: %lu seconds and %lu nanoseconds\n", elapsed_time, elapsed_nanos);

    return 0;
}
```

```
fatimas-mbp:C_programs fatimanwar$ ./a.out
Elapsed time is: 5.003983 seconds and 3983000 nanoseconds
Elapsed time is: 5 seconds and 3983000 nanoseconds
```

Relational Operators

a	b
(2*5+4/x)	75

- Equality ==
- inequality !=
- <, ≤, >, ≥

a == b
a != b
a < b

produce a boolean result (1 if true 0 if false)

```c
#include <stdio.h>

int main() {
  int ch;

  ch = getchar();
  while(ch != 'a'){
    if(ch != '\n')
      printf("ch was %c, value %d\n", ch, ch);
    ch = getchar();
  }
  return 0;
}
```

```
fatimas-mbp:C_programs fatimanwar$ ./a.out
c
ch was c, value 99
d
ch was d, value 100
g
ch was g, value 103
t
ch was t, value 116
a
```

```c
/* program that generates prime numbers.*/
#include <stdio.h>

int main() {
  int this_number, divisor, not_prime;
  this_number = 3;

  while(this_number < 10){
    divisor = this_number / 2;
    not_prime = 0;
    while(divisor > 1) {
      if(this_number % divisor == 0){
        not_prime = 1;
        divisor = 0;
      }
      else
        divisor = divisor-1;
    }
    if(not_prime == 0)
      printf("%d is a prime number\n", this_number);
    this_number = this_number + 1;
  }
  return 0;
}
```

```
fatimas-mbp:C_programs fatimanwar$ ./a.out
3 is a prime number
5 is a prime number
7 is a prime number
```

Logical Operators

combine results of relational operations

- && means AND 1&&1=1; 1&&10=1; 10&&0=0
- || means OR 1||1=1; 1||0=1; 10||0=1
- ! means NOT !1=0; !0=1; !10=0

Control Structures

Conditionals
Loops
Statement

Control Structures

- if
- if-else
- for loop
- while loop
- do while loop
- switch statement

if block

0 is false
non-zero is true

```
if (condition)
(statement)
```

single statement doesn't require braces. But most style guides recommend them.

```
if (condition)
{
        . . .
}
```

```
if (condition){
    ...
}
```

curly braces vertically aligned; easier to read.

first curly brace in line with condition; saves space but harder to read.

which style to use? Either. Pick one and be consistent with it in your coding.

if/else control structure

```
if (condition)
{
      ...
}
else
{
          . . .
}
```

```
if (condition){
   ...
} else {
   ...
}
```

curly braces vertically aligned – easier to read.

first curly brace in line with condition. Saves space but harder braces to read.

which style to use? Either. Pick one and be consistent with it in your coding.

if/else-if/else control structure

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

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

```c
/*****************************************************
if-statement.c  -- this program perform arithmetic
operations based on conditional statements and
prints their results

Date          Author        Revision
02/05/2024    Fatima Anwar  initial version
*****************************************************/

#include <stdio.h>

int main() {
    int a = 0, b = 0, x, xx, yy;
    // Assuming a and b are already defined
    // or you can take input for them
    if (a >= b) {
        x = 0;
    }
    if (a >= b + 1){
        xx = 0;
        yy = -1;
    }
    else {
        xx = 100;
        yy = 200;
    }
    // Display the values of x, xx, and yy
    printf("x = %d, xx = %d, yy= %d\n", x, xx, yy);
    return 0;
}
```

```
fatimas-mbp:C_programs fatimanwar$ ./a.out
x = 0, xx = 100, yy= 200
```

while loop

```
while (condition) {
     ...
}
```

keep looping as long condition is true (true means condition evaluates to a non-zero value)

```
example:
i = 0;
while (i < 10) {
     printf("This is iteration %d\n", i);
     i++;
}
```

what happens if we don't have i++ ?
while (1) { ;} ? Infinite loops

do loop

```
do {
     ...
} while (condition)
```

keep looping as long condition is true (true means condition evaluates to a non-zero value)

This structure will always execute the {...} block at least once.

```
example:
i = 0;
do {
     printf("This is iteration %d\n", i);
     i++;
} while (i < 10)
```

Here’s the mistake many people make:

```
a = 10
b = 20
if (a=b) {
   print(“They’re equal!”);
} else {
   print(“They’re not equal\n)
}
```

C language philosophy: trust the programmer.
C will let you really screw up!

for loop

```
for (initialization; condition; update) {
    ...
}
```

example:

```
for (i = 1; i < 10; i++) {
    printf("This is iteration %d\n", i);
}
```

initialization, condition, update are all optional !

for(; ;) { ; } ? Infinite loop

```c
/*****************************************************
for-statement.c  -- this program performs
operations using for statement and prints the
results

Date          Author        Revision
02/05/2024    Fatima Anwar  initial version
*****************************************************/

#include <stdio.h>

int main() {
    int s, i, n;
    // compute s = 1 + 2 + ... + n
    n = 19;
    s = 0;
    for (i = 1; i <= n; i++){
        s += i;
    }
    // Display the value of s
    printf("The sum from 1 to %d is: %d\n", n, s);
    return 0;
}
```

```
fatimas-mbp:C_programs fatimanwar$ ./a.out
The sum from 1 to 19 is: 190
```

break & continue

- break – control is exited from the construct (loop, struct) immediately
- continue – control is passed to the beginning of the construct (loop statement)

```c
#include <stdio.h>

int main() {
  int i;

  for (i = -10; i < 10; i++) {
    if (i == 0)
      continue;
    printf("%f\n", 15.0/i);
    /*
     * Lots of other statements .
     */
  }
  return 0;
}
```

```
fatimas-mbp:C_programs fatimanwar$ ./a.out
-1.500000
-1.666667
-1.875000
-2.142857
-2.500000
-3.000000
-3.750000
-5.000000
-7.500000
-15.000000
15.000000
7.500000
5.000000
3.750000
3.000000
2.500000
2.142857
1.875000
1.666667
```

Nested for loops and functions

```c
#include <stdio.h>

void pmax(int first, int second);

int main() {
      int i,j;
      for (i = -10; i <= 10; i++) {
            for (j = -10; j <= 10; j++)
                pmax(i,j);
            }
      }
      return 0;
}

void pmax(int a1, int a2) {
  int biggest = (a1 > a2) ? a1: a2;
  printf("largest of %d and %d is %d\n"
     a1, a2, biggest);
}
```

```
fatimas-mbp:C_programs fatimanwar$ ./a
largest of -10 and -10 is -10
largest of -10 and -9 is -9
largest of -10 and -8 is -8
largest of -10 and -7 is -7
largest of -10 and -6 is -6
largest of -10 and -5 is -5
largest of -10 and -4 is -4
largest of -10 and -3 is -3
largest of -10 and -2 is -2
largest of -10 and -1 is -1
largest of -10 and 0 is 0
largest of -10 and 1 is 1
```

Switch statement

```
Switch (expression) { \\expression must be of type int, char
     case label_1:  \\if label matches the expression,
          statement_1 \\execute this statement.
          break
     case label_2:
          statement_2
          .
          .
          .
     default:
          statement_n
}
```

Notes:

Substitute for long if statements
Cases should be unique
break is optional
default case is optional

```
Example:
switch (letter){
   case 'N':
      printf("New York\n");
      break;
   default:
      printf("Somewhere else\n”);
      break;
}
```

```c
/*****************************************
Message printing function

Date          Author        Revision
02/05/2024    Fatima Anwar  initial version
*****************************************/

#include <stdio.h>

void message(int message_number) {
    switch (message_number) {
        case 1:
            printf("Hello World\n");
            break;
        case 2:
            printf("Goodbye World\n");
            break;
        default:
            printf("unknown message\n");
            break;
    }
}

int main() {

    int x = 2;
    message(1); /* What will happen? */
    message(x);
    message(0);

    return 0;
}
```

```
fatimas-mbp:C_programs fatimanwar$ ./a.out
Hello World
Goodbye World
unknown message
```

```c
/* C Program to create a simple calculator using switch
 statement */
#include <stdio.h>

int main(){

    char choice; // switch variable
    int x, y; // operands

    while (1) {
        printf("Enter the Operator (+,-,*,/
        scanf(" %c", &choice);

        printf("Enter the two numbers: ");
        scanf("%d %d", &x, &y);

        // switch case with operation for e
        switch (choice) {
          case '+':
              printf("%d + %d = %d\n", x, y
              break;
          case '-':
              printf("%d - %d = %d\n", x, y, x - y);
              break;
          case '*':
              printf("%d * %d = %d\n", x, y, x * y);
              break;
          case '/':
              printf("%d / %d = %d\n", x, y, x / y);
              break;
          default:
              printf("Invalid Operator Input\n");
        }
    }
    return 0;
}
```

```
fatimas-mbp:C_programs fatimanwar$ ./a.out
Enter the Operator (+,-,*,/)
+
Enter the two numbers: 1 2
1 + 2 = 3
Enter the Operator (+,-,*,/)
-
Enter the two numbers: 4 6
4 - 6 = -2
Enter the Operator (+,-,*,/)
-
Enter the two numbers: -9 -8
-9 - -8 = -1
```