# C Lab

Assembly code file types?

In assembly language programming, different file types are commonly used to denote various stages of the development and compilation process. Here are the primary types:

### 1. ****.asm****

* **Description**: This is the standard file extension for assembly language source code. It contains the human-readable assembly instructions.
* **Usage**: Written by programmers and contains the assembly code.

### 2. ****.s****

* **Description**: Some assemblers (like GNU Assembler) use this extension for assembly files, especially when the code is in a format ready for assembly.
* **Usage**: Can be generated from C code by using the -S option in GCC.

### 3. ****.o****

* **Description**: This is an object file that contains machine code produced by the assembler. It is not yet linked into a complete executable.
* **Usage**: Generated after assembling the .asm or .s file. It's used in the linking stage.

### 4. ****.exe****

* **Description**: This is a common extension for executable files on Windows.
* **Usage**: Contains the final compiled program that can be run directly by the operating system.

### 5. ****.out****

* **Description**: This is often used as a default output name for executables on Unix-like systems.
* **Usage**: The output file produced when compiling a program without specifying an output name.

### 6. ****.elf****

* **Description**: The Executable and Linkable Format, used for executable files and shared libraries in Unix-like systems.
* **Usage**: Contains all the information needed for the program to run, including headers, sections, and program segments.

### Summary

* **Source Code**: .asm, .s
* **Object Files**: .o
* **Executable Files**: .exe, .out, .elf

**A) Compiling and Running**

1. **Create the C file**: Save the above code in a file named print\_name.c.
2. **Compile the C code**: Open a terminal and navigate to the directory where you saved the file. Then run:

gcc print\_name.c -o print\_name

1. **Run the executable**: After compilation, run the program with:

./print\_name

You should see the output:

ANIL

### B) Steps to Convert .c to .asm

1. **Create your C file**: Save the following code in a file named print\_name.c:

#include <stdio.h>

int main() {

printf("ANIL\n");

return 0;

}

1. **Open a terminal** and navigate to the directory where your print\_name.c file is located.
2. **Compile to assembly**: Use the following command:

gcc -S print\_name.c

1. **Check the output**: This command generates an assembly file named print\_name.s. You can view the contents of this file using:

cat print\_name.s

### Output

The print\_name.s file will contain the assembly code corresponding to your C program. It will include sections for the .text segment (code), and possibly data segments for the strings used in the printf function.

**C) Example Assembly Code (print\_name.asm)**

.section .data

msg: .asciz "ANIL\n" # The string to be printed

.section .text

.globl \_start

\_start:

# Write the string to stdout

mov rax, 1 # syscall: write

mov rdi, 1 # file descriptor: stdout

mov rsi, msg # pointer to the string

mov rdx, 5 # length of the string

syscall # invoke syscall

# Exit the program

mov rax, 60 # syscall: exit

xor rdi, rdi # exit status 0

syscall # invoke syscall

**Explanation**

* **.section .data**: This section is for initialized data. We define a string msg that contains "ANIL\n".
* **.section .text**: This section contains the code. The \_start label marks the entry point of the program.
* **syscall**: This is used to make system calls. The write syscall (1) outputs to stdout (file descriptor 1), and the exit syscall (60) terminates the program.

**How to Assemble and Run**

1. **Save the code** in a file named print\_name.asm.
2. **Assemble the code** using nasm (the Netwide Assembler):

nasm -f elf64 print\_name.asm

1. **Link the object file** to create an executable:

ld -o print\_name print\_name.o

1. **Run the executable**:

./print\_name

You should see the output:

ANIL

**Steps to Generate an Assembly File from a C File**

1. **Create a C file**: Save the following C code in a file named sample.c:

#include <stdio.h>

int main() {

printf("ANIL\n");

return 0;

}

1. **Open a terminal** and navigate to the directory where you saved sample.c.
2. **Run GCC with the -S option**: Use the following command to generate an assembly file:

gcc -S sample.c

1. **Check the output**: After running the command, you will see a new file named sample.s in the same directory. You can view its contents with:

cat sample.s

**Output**

The sample.s file will contain the assembly code corresponding to your C program. It will include sections for the .text segment (code) and possibly data segments for the strings used in the printf function.

**Summary**

* **Command**: gcc -S sample.c generates sample.s.
* **File Types**:
  + .c: C source file
  + .s: Assembly source file

**Additional Tips**

* To compile with debugging information (useful for debugging), add the -g flag:

gcc -g -o sample sample.c

* To see more compiler warnings, you can use the -Wall option:

gcc -Wall -o sample sample.c

### UNIT I

### WEEK 1

### Objective: Getting familiar with the programming environment on the computer and writing the first program. Suggested Experiments/Activities:

### Tutorial 1: Problem-solving using Computers.

### Lab1: Familiarization with programming environment

### Basic Linux environment and its editors like Vi, Vim & Emacs etc.

### Exposure to Turbo C, gcc

### Writing simple programs using printf(), scanf()

### Ans:-

### Objective

To get familiar with the programming environment on the computer and write your first program.

### Suggested Experiments/Activities

#### Tutorial 1: Problem-Solving Using Computers

1. **Understanding Problem-Solving**:
   * Discuss the process of problem-solving: Define the problem, plan a solution, write the program, and test it.
2. **Algorithm Development**:
   * Learn to develop algorithms for simple problems (e.g., finding the maximum of three numbers).

#### Lab 1: Familiarization with Programming Environment

##### i) Basic Linux Environment and Editors

1. **Using the Terminal**:
   * Open the terminal and familiarize yourself with basic commands:
     + ls: List files in the current directory.
     + cd: Change directory.
     + mkdir: Create a new directory.
     + rm: Remove files or directories.
2. **Text Editors**:
   * **Vi**:
     + Open a file: vi filename.c
     + Switch to insert mode: Press i
     + Save and exit: Press Esc, then type :wq
   * **Vim**:
     + Similar to Vi, but offers more features.
     + Use vim filename.c to open.
     + Learn basic navigation and commands.
   * **Emacs**:
     + Open a file: emacs filename.c
     + Use Ctrl + x, Ctrl + s to save and Ctrl + x, Ctrl + c to exit.

##### ii) Exposure to Turbo C and GCC

1. **Turbo C**:
   * Usually used in educational settings for learning C.
   * Learn to navigate the IDE and write simple programs.
2. **GCC (GNU Compiler Collection)**:
   * Open the terminal.
   * To compile a C program: gcc -o program\_name filename.c
   * Run the program: ./program\_name

##### iii) Writing Simple Programs Using printf() and scanf()

1. **Simple Program Example**: Create a C file named hello.c with the following content:

#include <stdio.h>

int main() {

char name[50];

printf("Enter your name: ");

scanf("%s", name);

printf("Hello, %s!\n", name);

return 0;

}

1. **Steps to Compile and Run**:
   * Open a terminal and navigate to the directory containing hello.c.
   * Compile the program:

gcc -o hello hello.c

* + Run the program:

./hello

* + Enter your name when prompted and observe the output.

**WEEK 2**

**Objective:** Getting familiar with how to formally describe a solution to a problem in a series of finite steps both using textual notation and graphic notation.

**Suggested Experiments /Activities:**

**Tutorial 2:** Problem-solving using Algorithms and Flow charts.

**Lab 1:** Converting algorithms/flow charts into C Source code. Developing the algorithms/flowcharts for the following sample programs

1. Sum and average of 3 numbers
2. ii) Conversion of Fahrenheit to Celsius and vice versa
3. iii) Simple interest calculation

**Ans:-**

### Objective

To get familiar with how to formally describe a solution to a problem in a series of finite steps using both textual and graphic notation.

### Suggested Experiments/Activities

#### Tutorial 2: Problem-Solving Using Algorithms and Flowcharts

1. **Understanding Algorithms**:
   * An algorithm is a step-by-step procedure to solve a problem. It should be clear, unambiguous, and finite.
2. **Understanding Flowcharts**:
   * Flowcharts are graphical representations of algorithms. They use various symbols to denote different types of operations (e.g., processes, decisions).

### Lab 1: Converting Algorithms/Flowcharts into C Source Code

#### Sample Programs

##### i) Sum and Average of 3 Numbers

**Algorithm**:

1. Start
2. Input three numbers: num1, num2, num3
3. Calculate sum = num1 + num2 + num3
4. Calculate average = sum / 3
5. Output sum and average
6. End

**Flowchart**:

* Start -> Input Numbers -> Calculate Sum -> Calculate Average -> Output -> End

**C Code**:

#include <stdio.h>

int main() {

float num1, num2, num3, sum, average;

printf("Enter three numbers: ");

scanf("%f%f%f", &num1, &num2, &num3);

sum = num1 + num2 + num3;

average = sum / 3;

printf("Sum: %.2f\n", sum);

printf("Average: %.2f\n", average);

return 0;

}

##### ii) Conversion of Fahrenheit to Celsius and Vice Versa

**Algorithm**:

1. Start
2. Input temperature and conversion choice (F to C or C to F)
3. If choice is F to C:
   * Calculate C = (F - 32) \* 5/9
4. Else if choice is C to F:
   * Calculate F = (C \* 9/5) + 32
5. Output the converted temperature
6. End

**Flowchart**:

* Start -> Input Temperature and Choice -> Decision (F to C or C to F) -> Calculate -> Output -> End

**C Code**:

#include <stdio.h>

int main() {

float temperature, converted;

char choice;

printf("Enter temperature (C/F): ");

scanf("%f%c", &temperature, &choice);

if (choice == 'F' || choice == 'f') {

converted = (temperature - 32) \* 5 / 9;

printf("Temperature in Celsius: %.2f\n", converted);

} else if (choice == 'C' || choice == 'c') {

converted = (temperature \* 9 / 5) + 32;

printf("Temperature in Fahrenheit: %.2f\n", converted);

} else {

printf("Invalid choice!\n");

}

return 0;

}

##### iii) Simple Interest Calculation

**Algorithm**:

1. Start
2. Input principal amount, rate of interest, and time
3. Calculate SI = (Principal \* Rate \* Time) / 100
4. Output SI
5. End

**Flowchart**:

* Start -> Input Principal, Rate, Time -> Calculate SI -> Output SI -> End

**C Code**:

#include <stdio.h>

int main() {

float principal, rate, time, simple\_interest;

printf("Enter principal, rate, and time: ");

scanf("%f%f%f", &principal, &rate, &time);

simple\_interest = (principal \* rate \* time) / 100;

printf("Simple Interest: %.2f\n", simple\_interest);

return 0;

}

**WEEK 3**

**Objective:** Learn how to define variables with the desired data-type, initialize them with appropriate values and how arithmetic operators can be used with variables and constants.

**Suggested Experiments/Activities:**   
**Tutorial 3:** Variable types and type conversions:

**Lab 3:** Simple computational problems using arithmetic expressions.

1. Finding the square root of a given number
2. Finding compound interest
3. Area of a triangle using heron’s formulae
4. Distance travelled by an object

Ans:-

### Week 3

#### Objective

Learn how to define variables with the desired data type, initialize them with appropriate values, and use arithmetic operators with variables and constants.

### Suggested Experiments/Activities

#### Tutorial 3: Variable Types and Type Conversions

1. **Data Types in C**:
   * **int**: Integer type (e.g., int a = 5;)
   * **float**: Single precision floating point (e.g., float b = 5.5;)
   * **double**: Double precision floating point (e.g., double c = 10.1234;)
   * **char**: Character type (e.g., char d = 'A';)
2. **Type Conversions**:
   * Implicit conversion (automatic by C) and explicit conversion (using casting).
   * Example of implicit conversion:

int a = 5;

float b = a; // a is automatically converted to float

* + Example of explicit conversion:

float a = 5.5;

int b = (int)a; // a is explicitly converted to int

### Lab 3: Simple Computational Problems Using Arithmetic Expressions

#### i) Finding the Square Root of a Given Number

**C Code**:

#include <stdio.h>

#include <math.h> // Include math.h for sqrt function

int main() {

double number, square\_root;

printf("Enter a number: ");

scanf("%lf", &number);

square\_root = sqrt(number); // Calculate square root

printf("Square root of %.2f is %.2f\n", number, square\_root);

return 0;

}

#### ii) Finding Compound Interest

**C Code**:

#include <stdio.h>

#include <math.h>

int main() {

double principal, rate, time, compound\_interest;

printf("Enter principal amount, rate of interest, and time (in years): ");

scanf("%lf %lf %lf", &principal, &rate, &time);

compound\_interest = principal \* pow((1 + rate / 100), time) - principal; // CI formula

printf("Compound Interest: %.2f\n", compound\_interest);

return 0;

}

#### iii) Area of a Triangle Using Heron's Formula

**C Code**:

#include <stdio.h>

#include <math.h>

int main() {

double a, b, c, s, area;

printf("Enter lengths of the three sides of the triangle: ");

scanf("%lf %lf %lf", &a, &b, &c);

s = (a + b + c) / 2; // Semi-perimeter

area = sqrt(s \* (s - a) \* (s - b) \* (s - c)); // Heron's formula

printf("Area of the triangle: %.2f\n", area);

return 0;

}

#### iv) Distance Travelled by an Object

**C Code**:

#include <stdio.h>

int main() {

double speed, time, distance;

printf("Enter speed (in km/h) and time (in hours): ");

scanf("%lf %lf", &speed, &time);

distance = speed \* time; // Distance = Speed × Time

printf("Distance travelled: %.2f km\n", distance);

return 0;

}

**UNIT II**

**WEEK 4**

**Objective:** Explore the full scope of expressions, type-compatibility of variables & constants and operators used in the expression and how operator precedence works.

**Suggested Experiments/Activities:**   
**Tutorial4:** Operators and the precedence and as associativity:

**Lab4:** Simple computational problems using the operator’ precedence and associativity

i) Evaluate the following expressions.

a. A+B\*C+(D\*E) + F\*G

b. A/B\*C-B+A\*D/3

c. A+++B---A

d. J= (i++) + (++i)

ii) Find the maximum of three numbers using conditional operator

iii) Take marks of 5 subjects in integers, and find the total, average in float

Ans:

Plan for Week 4 that focuses on expressions, type compatibility, operators, and operator precedence in C programming.

### Week 4

#### Objective

Explore the full scope of expressions, type compatibility of variables and constants, operators used in expressions, and how operator precedence and associativity work.

### Suggested Experiments/Activities

#### Tutorial 4: Operators, Precedence, and Associativity

1. **Operators in C**:
   * **Arithmetic Operators**: +, -, \*, /, %
   * **Relational Operators**: ==, !=, <, >, <=, >=
   * **Logical Operators**: &&, ||, !
   * **Bitwise Operators**: &, |, ^, ~, <<, >>
   * **Assignment Operators**: =, +=, -=, \*=, /=, %=
   * **Increment/Decrement Operators**: ++, --
2. **Operator Precedence**:
   * Operators have different precedence levels that determine the order of operations in expressions.
   * Example: \* and / have higher precedence than + and -.
3. **Associativity**:
   * Determines the order in which operators of the same precedence are evaluated.
   * Left-to-right: Most arithmetic operators.
   * Right-to-left: Assignment operators.

### Lab 4: Simple Computational Problems Using Operator Precedence and Associativity

#### i) Evaluate the Following Expressions

To evaluate the expressions programmatically, create a C file named expressions.c:

**C Code**:

#include <stdio.h>

int main() {

int A = 5, B = 10, C = 2, D = 3, E = 4, F = 6, G = 1;

int i = 2, J;

// Expression a

int result\_a = A + B \* C + (D \* E) + F \* G;

printf("Result of (a) A + B \* C + (D \* E) + F \* G: %d\n", result\_a);

// Expression b

int result\_b = A / B \* C - B + A \* D / 3;

printf("Result of (b) A / B \* C - B + A \* D / 3: %d\n", result\_b);

// Expression c

int result\_c = A++ + B-- - A;

printf("Result of (c) A+++B---A: %d\n", result\_c); // Note: A and B will change after this

// Expression d

J = (i++) + (++i);

printf("Result of (d) J = (i++) + (++i): %d\n", J);

return 0;

}

#### ii) Find the Maximum of Three Numbers Using the Conditional Operator

**C Code**:

#include <stdio.h>

int main() {

int a, b, c, max;

printf("Enter three numbers: ");

scanf("%d %d %d", &a, &b, &c);

max = (a > b && a > c) ? a : (b > c ? b : c);

printf("Maximum of the three numbers: %d\n", max);

return 0;

}

#### iii) Take Marks of 5 Subjects in Integers, and Find the Total and Average in Float

**C Code**:

#include <stdio.h>

int main() {

int marks[5];

int total = 0;

float average;

printf("Enter marks of 5 subjects: ");

for (int i = 0; i < 5; i++) {

scanf("%d", &marks[i]);

total += marks[i];

}

average = total / 5.0; // Use 5.0 to ensure float division

printf("Total: %d\n", total);

printf("Average: %.2f\n", average);

return 0;

}