

CC-112L

Programming Fundamentals

Laboratory 14

Miscellaneous Topics

Version: 1.0.0

Release Date: 28-10-2022

Department of Information Technology

University of the Punjab

Lahore, Pakistan

Contents:

- Learning Objectives
- Required Resources
- General Instructions
- Background and Overview
 - Variable Length Argument List
 - Command Line Arguments
 - extern Modifier
 - exit Function
 - atexit Function
- Activities
 - Pre-Lab Activity
 - Variable Length Argument List
 - Task 01: Variable Length Argument
 - In-Lab Activity
 - Command Line Arguments
 - Compiling Multi Source File Program
 - Global Variables in Other Files
 - Restricting Scope with static
 - exit() Function
 - atexit() Function
 - Suffixes for Integer and Floating-Point Literals
 - Task 01: File Writing
 - Task 02: Second Largest
 - Task 03: Triplet Product
- Submissions
- Evaluations Metric
- References and Additional Material
- Lab Time and Activity Simulation Log

Learning Objectives:

- Variable Length Argument Lists
- Command Line Arguments
- Compilation of Multi Source Files
- Program Termination in C
- Literals Suffixes

Resources Required:

- Desktop Computer or Laptop
- Microsoft ® Visual Studio 2022

General Instructions:

- In this Lab, you are **NOT** allowed to discuss your solution with your colleagues, even not allowed to ask how is s/he doing, this may result in negative marking. You can **ONLY** discuss with your Teaching Assistants (TAs) or Lab Instructor.
- Your TAs will be available in the Lab for your help. Alternatively, you can send your queries via email to one of the followings.

Teachers:		
Course Instructor	Prof. Dr. Syed Waqar ul Qounain	swjaffry@pucit.edu.pk
Lab Instructor	Madiha Khalid	madiha.khalid@pucit.edu.pk
Teacher Assistants	Usman Ali	bitf19m007@pucit.edu.pk
	Saad Rahman	bsef19m021@pucit.edu.pk

Background and Overview:

Variable Length Argument List:

A variable-length argument is a feature that allows a function to receive any number of arguments. There are situations where a function handles a variable number of arguments according to requirements, such as: Sum of given numbers.

Command Line Arguments:

Command-line arguments are given after the name of the program in command-line shell of Operating Systems. To pass command line arguments, we typically define `main()` with two arguments : first argument is the number of command line arguments and second is list of command-line arguments.

extern Modifier:

`extern "C"` specifies that the function is defined elsewhere and uses the C-language calling convention. The `extern "C"` modifier may also be applied to multiple function declarations in a block. In a template declaration, `extern` specifies that the template has already been instantiated elsewhere.

exit Function:

The `exit()` function is used to terminate a process or function calling immediately in the program. It means any open file or function belonging to the process is closed immediately as the `exit()` function occurred in the program. The `exit()` function is the standard library function of the C, which is defined in the `stdlib.h` header file. So, we can say it is the function that forcefully terminates the current program and transfers the control to the operating system to exit the program. The `exit(0)` function determines the program terminates without any error message, and then the `exit(1)` function determines the program forcefully terminates the execution process.

© <https://www.javatpoint.com/exit-function-in-c>

atexit Function:

The function pointed by `atexit()` is automatically called without arguments when the program terminates normally. In case more than one function has been specified by different calls to the `atexit()` function, all are executed in the order of a stack (i.e. the last function specified is the first to be executed at exit). A single function can be registered to be executed at exit more than once.

© <https://www.geeksforgeeks.org/atexit-function-in-c-c/>

Activities:

Pre-Lab Activities:

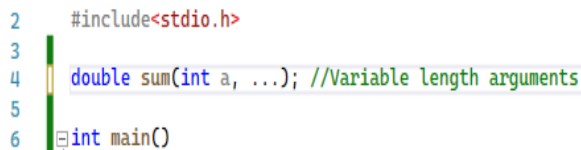
Variable Length Argument Lists:

Most programs in the text have used the standard-library function `printf`. At a minimum, `printf` must receive a string as its first argument, but `printf` can receive any number of additional arguments. The function prototype for `printf` is

`int printf(const char *format, ...);`

The “**ellipsis (...)**” in the function prototype indicates that the function receives a variable number of arguments of any type.

The ellipsis must be the last parameter. Placing the ellipsis in the middle of the parameter list is a syntax error. The following figure shows function with variable length argument list.



```

2  #include<stdio.h>
3
4  double sum(int a, ...); //Variable length arguments
5
6  int main()

```

Fig 01. (Variable Length Argument List)

The following table contains the variable arguments header’s macros and definitions for building functions with variable-length argument lists:

Identifier	Description
va_list	A type for holding information needed by macros <code>va_start</code> , <code>va_arg</code> and <code>va_end</code> . To access the arguments in a variable-length argument list, an object of type <code>va_list</code> must be defined.
va_start	A macro that you must invoke before accessing a variable-length argument list’s arguments. This macro initializes the object declared with <code>va_list</code> for use by the <code>va_arg</code> and <code>va_end</code> macros.
va_arg	A macro that expands to the variable-length argument list’s next argument value. The value has the type you specify as the macro’s second argument. Each use of <code>va_arg</code> modifies the object declared with <code>va_list</code> to point to the next argument.
va_end	A macro that facilitates a normal return from a function whose variable-length argument list was referred to by the <code>va_start</code> macro.

Following code demonstrates the variable length argument list:

```
2  #include <stdarg.h>
3  #include <stdio.h>
4
5  double sum(int i, ...); // ... represents variable arguments
6
7  int main(void) {
8      double a = 33.5, b = 2.5, c = 11.1, d = 10.2;
9      printf("%s%.1f\n%s%.1f\n%s%.1f\n%s%.1f\n", "a = ", a, "b = ", b, "c = ", c, "d = ", d);
10     printf("%s%.3f\n%s%.3f\n%s%.3f\n", "The sum of a and b is ", sum(2, a, b),
11           "The sum of a, b, and c is ", sum(3, a, b, c),
12           "The sum of a, b, c, and d is ", sum(4, a, b, c, d));
13 }
14 double sum(int i, ...) {
15     double total = 0; // initialize total
16     va_list ap; // stores information needed by va_start and va_end
17     va_start(ap, i); // initializes the va_list object
18     for (int j = 1; j <= i; ++j) {
19         total += va_arg(ap, double);
20     }
21     va_end(ap);
22     return total;
23 }
```

Fig 02. (Variable Length Argument List)

The output of the above program is following:



Microsoft Visual Studio Debug Console

```
a = 33.5
b = 2.5
c = 11.1
d = 10.2

The sum of a and b is 36.000
The sum of a, b, and c is 47.100
The sum of a, b, c, and d is 57.300
```

Fig 03. (Variable Length Argument List)

Task 01: Variable Length Arguments**[Estimated 30 minutes / 30 marks]**

Write a C program which:

- Take an integer as an input from the user which tells the number of arguments
- Take arguments as an input from the user
- Find average, minimum, and maximum value from the given arguments
- Display the output on the Console

Input	Output
2 3 4	Average: 3.5 Minimum: 3 Maximum: 4
3 2 3 4	Average: 3 Minimum: 2 Maximum: 4

In-Lab Activities:**Command Line Arguments:**

The main function may receive arguments from a command line if the function's parameter list contains the parameters `int "argc"` and `"char *argv[]"`:

- The `argc` parameter receives the number of command-line arguments that the user has entered.
- The `argv` parameter is an array of strings containing the command-line arguments.

Pass Command Line Argument Using Visual Studio:

- Right Click on Project from **"Solution Explorer"** and Select **"Properties"**.
- In the Project Properties Windows, Navigate to **"Debugging Tab"**
- You will find the text box **"Command Line"**. Type arguments here separated by space.

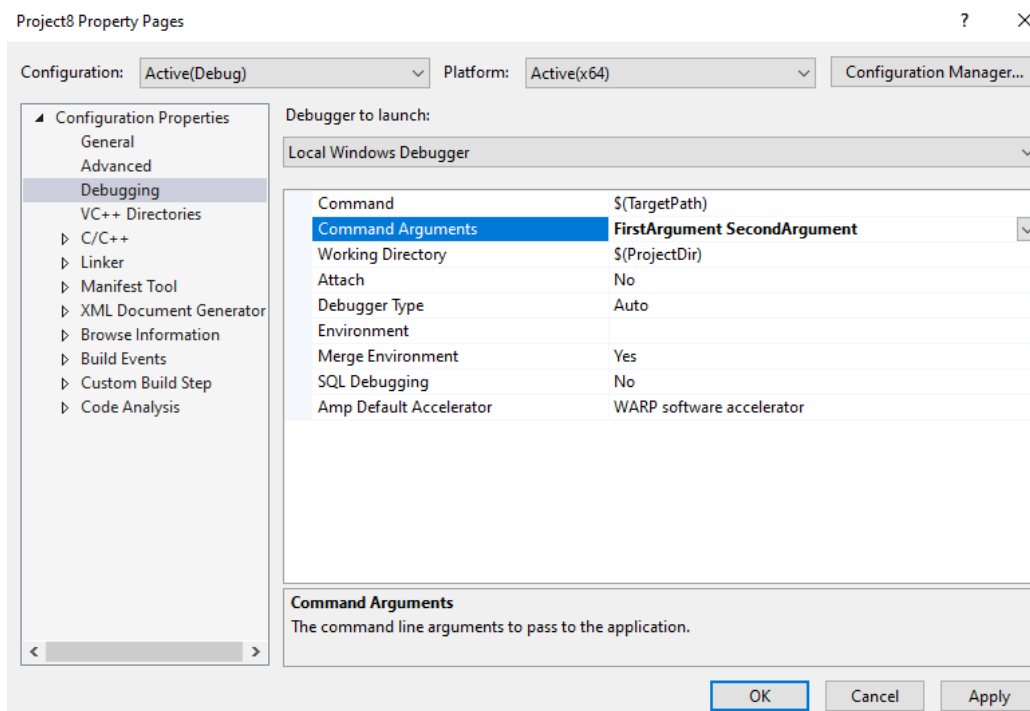


Fig 04. (Command Line Argument)

- Click Apply, then click OK.

Following code displays the command line arguments entered by the user:

```

2  #include <stdio.h>
3
4  int main(int argc, char** argv)
5  {
6      printf("You have entered %d arguments:\n", argc-1);
7
8      for (int i = 1; i < argc; ++i)
9          printf("%s\n", argv[i]);
10     return 0;
11 }
```

Fig 05. (Command Line Argument)

The iterator is started from 1 because the first argument is always the executable file.

Output of the above program is:



```
Microsoft Visual Studio Debug Console
You have entered 2 arguments:
FirstArgument
SecondArgument
```

Fig 06. (Command Line Argument)

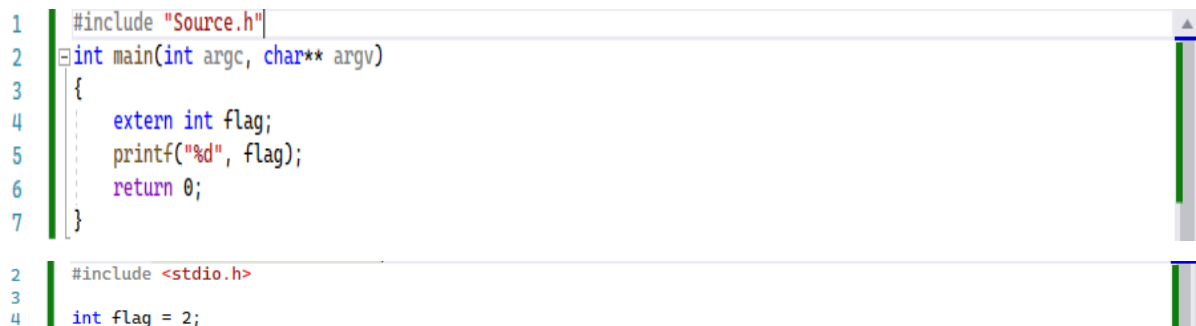
Compiling Multi Source File Programs:

Global Variables in Other Files:

Global variables can be accessible to functions in other files if they're declared in each file that uses them. For example, to refer to the global integer variable `flag` in another file, you can use the declaration

extern int flag

The storage-class specifier `extern` indicates that `flag` is defined either later in the same file or in a different file. The compiler informs the linker that unresolved references to the variable `flag` appear in the file. If the linker finds a proper global definition, the linker resolves the references to `flag`. If the linker cannot locate a definition of `flag`, it issues an error message and does not produce an executable file.



```
1 #include "Source.h"
2 int main(int argc, char** argv)
3 {
4     extern int flag;
5     printf("%d", flag);
6     return 0;
7 }

2 #include <stdio.h>
3
4 int flag = 2;
```

Fig 07. (Compiling Multiple Source Files)

The output of the above program is:



```
Microsoft Visual Studio Debug Console
2
```

Fig 08. (Compiling Multiple Source Files)

Restricting Scope with static:

It's possible to restrict a global variable or function's scope to the file in which it's defined. Applying the storage-class specifier `static` to a global variable or function prevents it from being used outside the file that defines it. This is known as internal linkage. Global variables and functions not preceded by `static` in their definitions have external linkage. They can be accessed in other files containing proper declarations.

The global variable definition

static const int flag = 2;

```

1  #include "Source.h"
2  int main(int argc, char** argv)
3  {
4      extern int flag;
5      printf("%d", flag);
6      return 0;
7  }

2  #include <stdio.h>
3
4  static const int flag = 2;

```

Fig 09. (Compiling Multiple Source Files)

The red line below flag indicates that it is inaccessible in another file.

Program Termination:

exit() Function:

The exit function terminates a program immediately. This function often is used to terminate a program when an error is detected. The function takes one argument— normally, EXIT_SUCCESS or EXIT_FAILURE.

atexit() Function:

The atexit function registers a function to call when the program terminates by reaching the end of main or when exit is invoked. This function takes as an argument another function's name.

The program below prompts the user to determine whether the program should be terminated with exit or by reaching the end of main.

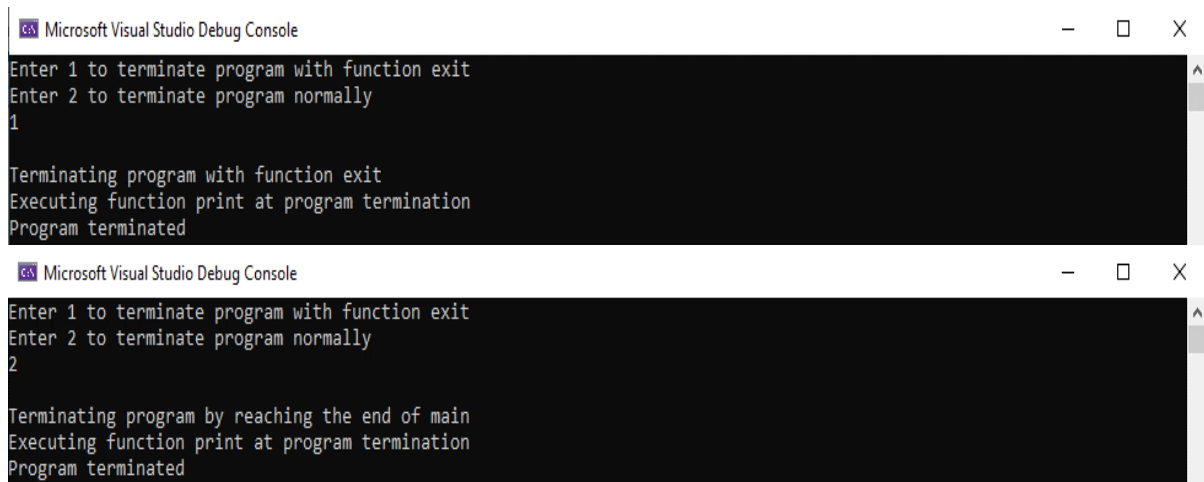
```

2  #include <stdio.h>
3  #include <stdlib.h>
4  void print(void);
5  int main(void) {
6      atexit(print); // register function print
7      puts("Enter 1 to terminate program with function exit\n"
8          "Enter 2 to terminate program normally");
9      int answer = 0; // user
10     scanf("%d", &answer);
11     // call exit if answer is 1
12     if (answer == 1) {
13         puts("\nTerminating program with function exit");
14         exit(EXIT_SUCCESS);
15     }
16     puts("\nTerminating program by reaching the end of main");
17 }
18 void print(void) {
19     puts("Executing function print at program termination\nProgram terminated");
20 }

```

Fig 10. (Program Termination)

The output of the above program is:



```
Microsoft Visual Studio Debug Console
Enter 1 to terminate program with function exit
Enter 2 to terminate program normally
1
Terminating program with function exit
Executing function print at program termination
Program terminated

Microsoft Visual Studio Debug Console
Enter 1 to terminate program with function exit
Enter 2 to terminate program normally
2
Terminating program by reaching the end of main
Executing function print at program termination
Program terminated
```

Fig 11. (Program Termination)

Suffixes for Integer and Floating- point Literals:

Integer and floating-point suffixes enable you to specify explicitly the data types of literal values.

The following literals have types unsigned int, long int, unsigned long int and unsigned long long int:

174u

8358L

28373ul

9876543210llu

The following are float and long double literals:

1.28f

3.14159L

Task 01: File Writing**[20 minutes / 20 mark]**

Write a C program that:

- Take two command line arguments
 - Filename
 - Content of File
- Open the file and append the contents in the file

Command Line	File Content
File.txt 1234	1234
File1.txt 1234	File doesn't exist

Task 02: Second Largest**[30 minutes / 30 marks]**

Create a C program that

- Reads an integer and variable argument list from command line
- Create a function with variable arguments
- Find the second largest among the numbers
- Display the Second Largest number on the Console

Input	Output
3 4 5 6	Second Largest Number: 5
2 6 16	Second Largest Number: 6

Task 03: Triplet Product**[40 minutes / 50 marks]**

Create a C Program that:

- Take an array of size 5 in source file
- Define elements of array in the header file
- Access the elements of array in the main program
- Find the maximum product of triplet (3 elements) in the array
- Display the result on the Console

Array Elements	Output
7 -6 0 1 -4	Maximum Product: 168
20 10 6 2 1	Maximum Product: 1200

Submissions:

- For In-Lab Activity:
 - Save the files on your PC.
 - TA's will evaluate the tasks offline.
- For Pre-Lab & Post-Lab Activity:
 - Submit the .c file on Google Classroom and name it to your roll no.

Evaluations Metric:

- All the lab tasks will be evaluated offline by TA's
- **Division of Pre-Lab marks:** [30 marks]
 - Task 01: Variable Length Arguments [30 marks]
- **Division of In-Lab marks:** [100 marks]
 - Task 01: Maximum Digit [20 marks]
 - Task 02: Second Largest Number [30 marks]
 - Task 03: Word Occurrence [50 marks]

References and Additional Material:

- Variable Arguments List
<https://www.geeksforgeeks.org/variable-length-argument-c/>
- Command Line Argument
<https://www.geeksforgeeks.org/command-line-arguments-in-c-cpp/>
- Literals in C
<https://www.geeksforgeeks.org/literals-in-c-cpp-with-examples>

Lab Time Activity Simulation Log:

- Slot – 01 – 00:00 – 00:15: Class Settlement
- Slot – 02 – 00:15 – 00:30: In-Lab Task
- Slot – 03 – 00:30 – 00:45: In-Lab Task
- Slot – 04 – 00:45 – 01:00: In-Lab Task
- Slot – 05 – 01:00 – 01:15: In-Lab Task
- Slot – 06 – 01:15 – 01:30: In-Lab Task
- Slot – 07 – 01:30 – 01:45: In-Lab Task
- Slot – 08 – 01:45 – 02:00: In-Lab Task
- Slot – 09 – 02:00 – 02:15: In-Lab Task
- Slot – 10 – 02:15 – 02:30: In-Lab Task
- Slot – 11 – 02:30 – 02:45: Evaluation of Lab Tasks
- Slot – 12 – 02:45 – 03:00: Evaluation of Lab Tasks