

CS 332/532 – 1G- Systems Programming

Lab 2

Objectives

The objective of this lab is to introduce you to C programming by developing the following programs and debugging them:

1. Go over the solution of Lab 01, and discuss differences & similarities between C and Java.
2. Introduction of Arrays in C.
3. Introduction of GDB - GNU Debugger.
4. Write a simple insertion sort program in C, compile and execute the program.

Exercise

Task 1:

In this task first we will go over the solution of Lab 01 and compare the C and Java code (you can download the C program (`prime.c`), and the Java program (`Prime.java`)).

`prime.c`:

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

int main(int argc, char** argv) {
    int given_number, i, flag = 1;
    printf("Enter a positive integer number: ");
    scanf("%d", &given_number);

    if (given_number <= 1) {
        printf("Number must be greater than 1\n");
        exit(-1);
    }

    for (i=2; i<=given_number/2 && flag != 0; i++) {
        if (given_number % i == 0) {
            flag = 0;
        }
    }
}
```

```

    }

    if (flag == 1) {
        printf("Given number %d is a prime number.\n", given_number);
    } else {
        printf("Given number %d is not a prime number.\n", given_number);
    }
}

```

Prime.java:

```

import java.util.Scanner;
public class Prime {

    public static void main(String[] args) {
        int i, flag = 1;

        System.out.print("Enter a positive integer number: ");
        Scanner scan = new Scanner(System.in);
        int given_number = scan.nextInt();

        if (given_number <=1) {
            System.out.println("Number must be greater than 1\n");
            System.exit(-1);
        }

        for (i=2; i<=given_number/2 && flag != 0; i++) {
            if (given_number % i == 0) {
                flag = 0;
            }
        }

        if (flag == 1) {
            System.out.println("Given number " + given_number + " is a prime number.");
        } else {
            System.out.println("Given number " + given_number + " is not a prime number.");
        }
    }
}

```

Let us compare the similarities and differences between programming in Java and C using this code.

Differences	
C Program	Java Program
No need to define class	Need to define class
no restriction	Filename should be same as class name, prefer to be CamelCase letter
File extensions are “*.c”, “*.h”	File extension is “*.java”
Library access by using keyword “include”	Library access by using keyword “import”
No need to specify main method as static	Need to specify main method as static
main method can return int or void	main method can return only void
Need to define garbage collection	Automatic garbage collection
Types of compilers: GCC C compiler, Turbo C compiler, Intel C compiler, Tiny C compiler	Type of compilers: Java Programming Language Compiler (javac), Eclipse Compiler for Java (ECJ), GNU Compiler for Java (GCJ)
Need to compile all dependent files together with main file	Any dependent file will automatically compile and re-compile if needed.

Similarities	
C Program	Java Program
<ul style="list-style-type: none"> Both need main method Syntax for comments /* comment */ & //line comment Escape Sequences like: \n, \t, \b, \r, \\, \' Logical operators like: &&, , ! Syntax for loops are same Statement always terminate with ; (semicolon) Conditional statement is same – <pre>if (expression) {statement} else {statement}</pre>	

The above is a partial list of differences and similarities, you can expand this list as you learn more about the C programming language.

Task 2:

Array syntax:

```
type array_name [array_size];
type array_name [] = {Element 1, ..., Element N};
```

Example:

```
int array[10];
int array[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
```

Now, write a program to find minimum number in an array by performing the following steps:

1. Create static array of type double and size 100
2. Initialize the array with random number (use the drand48)
3. Print the array
4. Find the minimum value in the array and print that value.

Code snippet:

1. Define constant size for array

```
#define size 100
```

This functionality allows us to initialize the constant value throughout the program.

2. Initialize array with random elements

```
#include <stdlib.h>
#include <time.h>

int main(int args, char** argv) {
    srand48((unsigned int)time(NULL));
    double array [size];

    for (i=0; i<size; i++) {
        array[i] = drand48();
    }
}
```

Here, first we need to define the seed point with `srand48()` in-order to call random generator function `drand48()`. Now, after every iteration we will assign the new random value to the array.

3. Find minimum from array.

```
min_value = array[0];

for (i=0; i<size; i++) {
    if (array[i] < min_value) {
        min_value = array[i];
    }
}
```

Here at first, we assume that the first element of array is minimum and then go over all the element and check whether next element is smaller than the previous or not.

Task 3:

In this task you will be using the GDB - GNU Debugger to debug your code. `gdb` is an extremely useful tool when you have to debug your program for runtime (e.g., segmentation faults, core dumps, array out-of-bound exceptions) and logical errors. To debug C programs using `gdb`, you have to compile your programs with the `-g` option to instruct the compiler to generate the executable with source-level debug information. Once your program is compiled with the `-g` option, then you can use `gdb` to debug your program as shown below:

```
gdb myexefile
gdb -tui myexefile
```

The `-tui` option displays the source code and gdb command prompt in a split screen (this is the recommended option for new users of gdb). The table below provides some of the commonly used gdb commands, the corresponding action performed by these commands, and an example.

gdb command	Description	Example
<code>file <i>executable</i></code>	specifies the program to execute (if you did not provide it as an argument to gdb)	file a.out
<code>break [<i>file:</i>]function</code>	set breakpoint at function (in file)	break main b myfunc
<code>break <i>line</i></code>	set breakpoint at line	break 15 b 15
<code>run [<i>args</i>]</code>	execute the program with optional command-line arguments	run run 10 20
<code>set <i>args</i></code>	set command-line arguments	set args 10 20
<code>bt</code>	display the program stack (backtrace)	bt
<code>print <i>expr</i></code>	print the value of the expression	print i p i
<code>continue</code>	continue program execution	continue c
<code>next</code>	execute next program line and step over any function calls in the line	next n
<code>step</code>	execute next program line and step into any function calls in the line	step

		s
list	display source lines where it is currently stopped (or lines after the last list command)	list l
list <i>[file]:function</i>	display source lines from the beginning of the function (in file)	list myfunc l myfunc
list <i>line</i>	display source lines around the line	list 15 l 15
where	display where error occurred	where
help <i>[command]</i>	display information on using gdb or display information on gdb command	help help break
quit	Exit gdb	quit

When you get a segmentation fault, if you compile your program with the `-g` option and run the program through gdb (using: `gdb -tui myexefile`), you will be able to immediately identify the line that is causing the segmentation fault.

Also check “**Resources**” section at the end of this file to learn more about C programming and GDB commands.

Lab Assignment #2

Write an insertion sort algorithm in C language by performing following steps:

1. Prompt the user to enter the number of array elements (say, N).
2. Read the number of elements (N).
3. Implement the insertion sort algorithm (note that insertion sort is an in-place sort that modifies the original array).
4. Print the sorted array.

Use the Java program (InsertionSort.java) provided as a reference to implement and test the C program.

Submission

Upload the C source file (.c file) to Canvas as part of this lab submission. Make sure to test the program on the CS Linux systems and include instructions to compile and run the program in the comments section of your program. Please do not upload executables.

Resources

1. Overview of C with
Tutorialspoint: https://www.tutorialspoint.com/cprogramming/c_overview.htm
2. A detailed tutorial on using gdb at: <http://beej.us/guide/bggdb/>
3. Quick tutorial of GDB Debugger: [Introduction to GDB a tutorial - Harvard CS50](#)
4. GDB Debugger cheat sheet: <https://darkdust.net/files/GDB%20Cheat%20Sheet.pdf>