Lab# 05

Command Line Argument, Process Environments, and Error Handlining

CS311L

Lab Engineer: Eisha ter raazia Mir

Learning Outcome

- To handle command line arguments in C
- To error handling in the C program
- To know the process Environments
- Structure in C

Command Line Arguments

- Command line arguments are nothing. Still, simply arguments that are specified after the name of the program in the system's command line, and these argument values are passed on to your program during program execution
- Given after the name of the program in command-line shell
- Reduce the length of the code
- Define main() with two arguments :
  - First argument - number of command line arguments
  - Second - list of command-line arguments

Command Line Argument

int main(int argc, char *argv[])

int argc: This variable counts the number of arguments passed to the program, including the program's name itself.

char *argv[]: This is an array of character pointers (strings), where each element represents an argument.

Argv: Argument Vector (Array of Strings)

It’s an array of character pointers (strings), where:

- argv[0] = program name
- argv[1] = first argument
- argv[2] = second argument, and so on.

./eisha testfile.txt

- Program Name (./**eisha**): This is always counted as the first argument.
- Argument (testfile.txt): This is the second argument.

Index	Value
argv[0]	./eisha
argv[1]	testfile.txt

Example : 01

```c
#include <stdio.h>
int main( int argc, char *argv [] )
{
printf(" \n Name of my Program %s \t", argv[0]);
if( argc == 2 )
{
printf("\n Value given by user is: %s \t", argv[1]);
}
else if( argc > 2 )
{
printf("\n Many values given by users.\n");
}
else
{
printf(" \n Single value expected.\n");
}
}
```

```
eisha@eisha-virtual-machine:~$ gcc lab2.c -o lab2
eisha@eisha-virtual-machine:~$ ./lab2 eisha

 Name of my Program ./lab2
 Value given by user is: eisha  eisha@eisha-virtual-machine:~$ S
```

Example: 02

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

int main(int argc, char *argv[])
{
    int i; // For iterating through the string
    printf("\n Name of my Program %s \t", argv[0]);

    if (argc == 2)
    {
        printf("\n Value given by user is: %s \t", argv[1]);
        printf("\n\n");

        // Loop through each character of argv[1] and print its ASCII value
        for (i = 0; i < strlen(argv[1]); i++)
        {
            printf("Character: %c, ASCII value: %d\n", argv[1][i], argv[1][i]);
        }
    }
    else if (argc > 2)
    {
        printf("\n Many values given by users.\n");
    }
}
```

```
eisha@eisha-virtual-machine:~$ touch lab22.c
eisha@eisha-virtual-machine:~$ gcc lab22.c -o lab22
eisha@eisha-virtual-machine:~$ ./lab22 hello

 Name of my Program ./lab22
 Value given by user is: hello

Character: h, ASCII value: 104
Character: e, ASCII value: 101
Character: l, ASCII value: 108
Character: l, ASCII value: 108
Character: o, ASCII value: 111
eisha@eisha-virtual-machine:~$
```

Process Environment

- Every program in C runs within a process environment, which contains important system information like the user's home directory, terminal type, and current locale.
- These details are stored as name–value pairs (e.g., PATH=/usr/bin, HOME=/home/eisha), passed to the program when it starts.
- The environment list can be accessed through a global variable environ, which holds pointers to these strings for use by the program if needed.

Error Handling

- **Global Variable errno:**
  - When a function is called in C, a variable named as errno is automatically assigned a value which can be used to identify the type of error that has been encountered
  - Its a global variable indicating the error occurred during any function call and defined in the header file errno.h
    Different values for errno mean different types of errors

Error Handling

- Below is a list of few different errno values and its corresponding meaning:

```
errno value        Error
1               /* Operation not permitted */
2               /* No such file or directory */
3               /* No such process */
4               /* Interrupted system call */
5               /* I/O error */
6               /* No such device or address */
7               /* Argument list too long */
8               /* Exec format error */
9               /* Bad file number */
10              /* No child processes */
11              /* Try again */
12              /* Out of memory */
13              /* Permission denied */
```

Error Handling

- **perror():**
  - displays the string **'s'**, followed by **':'**, and the error message associated with **errno**
- **errno:**
  - is an integer variable already defined for you in **errno.h**
  - It is set to the latest/last error condition generated by your program
  - If your program tries to do something that it isn't allowed to do, an error condition is reached in such a case
- **strerror():**
  - returns a pointer to the textual representation of the current errno value.

```c
#include <stdio.h>
#include <errno.h>
#include <string.h>

int main ()
{
        FILE *fp;

        // If a file is opened which does not exist,
        // then it will be an error and corresponding
        // errno value will be set
        fp = fopen(" new.txt ", "r");

        // opening a file which does
        // not exist.
        printf("Value of errno: %d\n ", errno);
        printf("The error message is : %s\n", strerror(errno));
        perror("Message from perror");

        return 0;
}
```

Output:

```
Value of errno: 2
The error message is : No such file or directory
Message from perror: No such file or directory
```

Structure in C

- Structures (also called **structs**) are a way to group several related variables into one place
- We use structures to create user define datatypes in C language.
- Each variable in the structure is known as a **member** of the structure
- Unlike an array, a structure can contain many different data types (**int, float, char, etc.**) under a single name

Structure in C

- Define Structure:
  - You can create a structure by using the **struct** keyword and declaring each of its members inside curly braces:

```
struct structureName {
  dataType member1;
  dataType member2;
  ...
};
```

```
struct Person {
  char name[50];
  int citNo;
  float salary;
};
```

Structure in C

- When a **struct** type is declared, no storage or memory is allocated
- We need to create variables to allocate the memory of a given structure type and work with it
- Here's how we create structure variables:

```
struct person
 {
char name[50]; // Name of the person
int citNo; // Citizenship number float
salary; // Salary of the person
};
```

OR

```
struct person person1;
```

Structure in C

- Suppose you want to access the **salary** of a **person2**

Output

```
Name: George Orwell
Citizenship No.: 1984
Salary: 2500.00
```

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

// Define the structure
struct person {
   char name[50];
   int citNo;
   float salary;
};

int main() {
   struct person person1; // Declare person1 within main

   // Assign values to the fields of person1
   strcpy(person1.name, "George Orwell");
   person1.citNo = 1984;
   person1.salary = 2500.00;

   // Print the values of the structure fields
   printf("Name: %s\n", person1.name);
   printf("Citizenship No: %d\n", person1.citNo);
   printf("Salary: %.2f\n", person1.salary);

   return 0;
}
```

Structure in C

- **Keyword typedef**
- We use the **typedef** keyword to create an alias name for data types. It is commonly used with structures to simplify the syntax of declaring variables
- This can make the code more readable and understandable, especially in cases where the purpose of the data type is not immediately clear.

Tasks:

- Write a C code which prints command line arguments only when these are more than one
- Write a code in C that declares a structure to store id, pages and price of a book. It inputs the records of five books from the user by asking every time to insert the data till the five books and display the record of most costly book. Clear the memory after your work
- Design a structure named **Question** to encapsulate a quiz question, including the question text, four possible answer options, and the correct answer. Implement an array of five Question instances. Prompt the user to respond to each question, and at the end of the quiz, display the user's score along with feedback indicating which answers were correct or incorrect.

Sample Output:

- Welcome to the Quiz!

Q1: What is the capital of France?

- A) Paris
- B) Rome
- C) Madrid
- D) Berlin

Enter your answer (A/B/C/D): B

Wrong! The correct answer was A.

Quiz finished! Your final score is: 2 out of 5