OPERATING SYSTEMS PRACTICE (ASSIGNMENT 3)

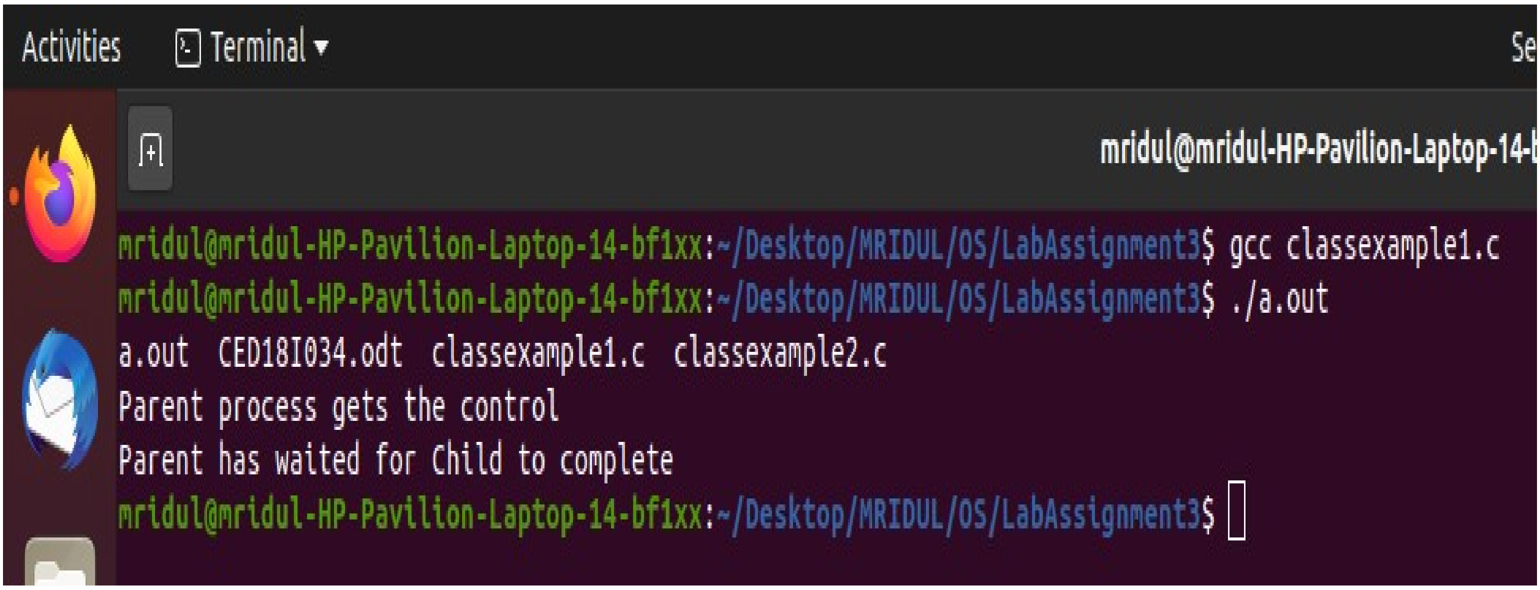
Name – Mridul Harish
Roll number – CED18I034

Question 1: Test Drive all the examples discussed so far in the class for the usageof wait, exec call variants.For the following questions you are free to decide the responsibility ofparent / child processes. Asmentioned in the class u r allowed to use vfork / file based approach toavoid the data sharing issueswhich we will later address using pipes in later classes to follow.

Code :-

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/wait.h>
#include<unistd.h>

int main()
{
	pid_t pid;
	pid = fork();
	if(pid == 0)
	{
		execl("/bin/ls", "ls", NULL);
	}
	else
	{
		wait(NULL);
		printf("Parent process gets the control\n");
		printf("Parent has waited for Child to complete\n");
	}
}
```

Output :-



```
mridul@mridul-HP-Pavilion-Laptop-14-bf1xx:~/Desktop/MRIDUL/OS/LabAssignment3$ gcc classexample1.c
mridul@mridul-HP-Pavilion-Laptop-14-bf1xx:~/Desktop/MRIDUL/OS/LabAssignment3$ ./a.out
a.out  CED18I034.odt  classexample1.c  classexample2.c
Parent process gets the control
Parent has waited for Child to complete
mridul@mridul-HP-Pavilion-Laptop-14-bf1xx:~/Desktop/MRIDUL/OS/LabAssignment3$
```

Explanation :-

Here the child function is executed so ls is executed.

int execl(const char*__path, const char*__arg, ...)
->path: Path of the executable file.
->arg: Arguments to be passed on to run the function.

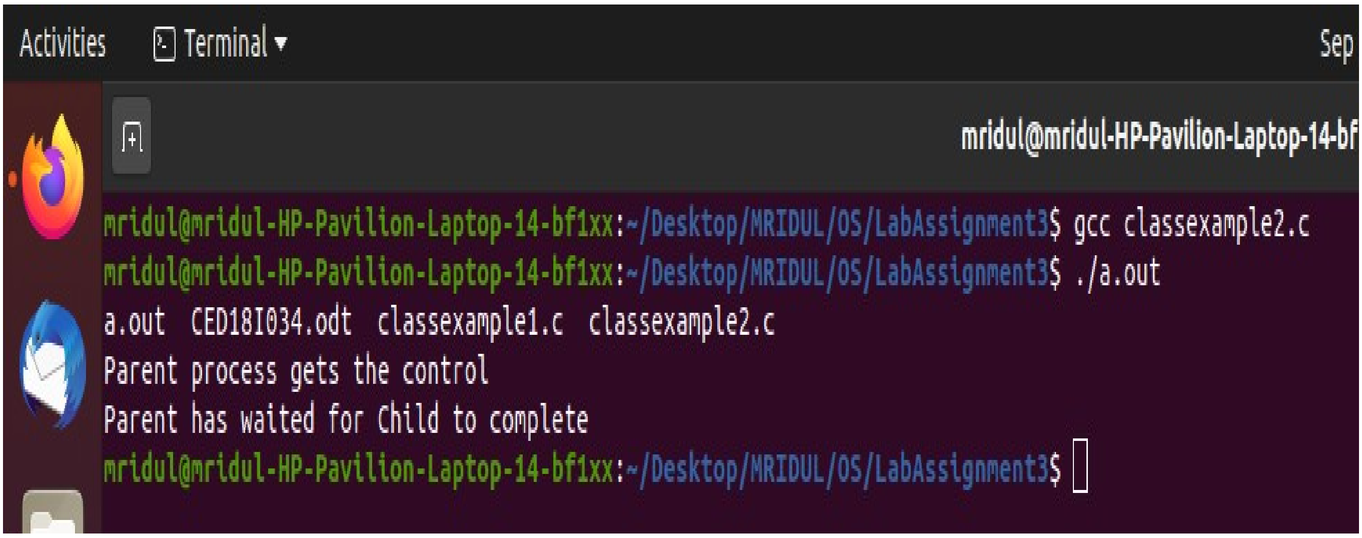
The parent process waits for the child to complete.

Code :-

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/wait.h>
#include<unistd.h>

int main()
{
        pid_t pid;
        pid = fork();
        if(pid == 0)
```

```
	{
		execlp("ls", "ls", NULL);
	}
	else
	{
		wait(NULL);
		printf("Parent process gets the control\n");
		printf("Parent has waited for Child to complete\n");
	}
}
```

Output :-



```
mridul@mridul-HP-Pavilion-Laptop-14-bf1xx:~/Desktop/MRIDUL/OS/LabAssignment3$ gcc classexample2.c
mridul@mridul-HP-Pavilion-Laptop-14-bf1xx:~/Desktop/MRIDUL/OS/LabAssignment3$ ./a.out
a.out  CED18I034.odt  classexample1.c  classexample2.c
Parent process gets the control
Parent has waited for Child to complete
mridul@mridul-HP-Pavilion-Laptop-14-bf1xx:~/Desktop/MRIDUL/OS/LabAssignment3$ 
```

Explanation :-

Here the child function is executed so ls is executed.

int execlp(const char*__path, const char*__arg, ...)

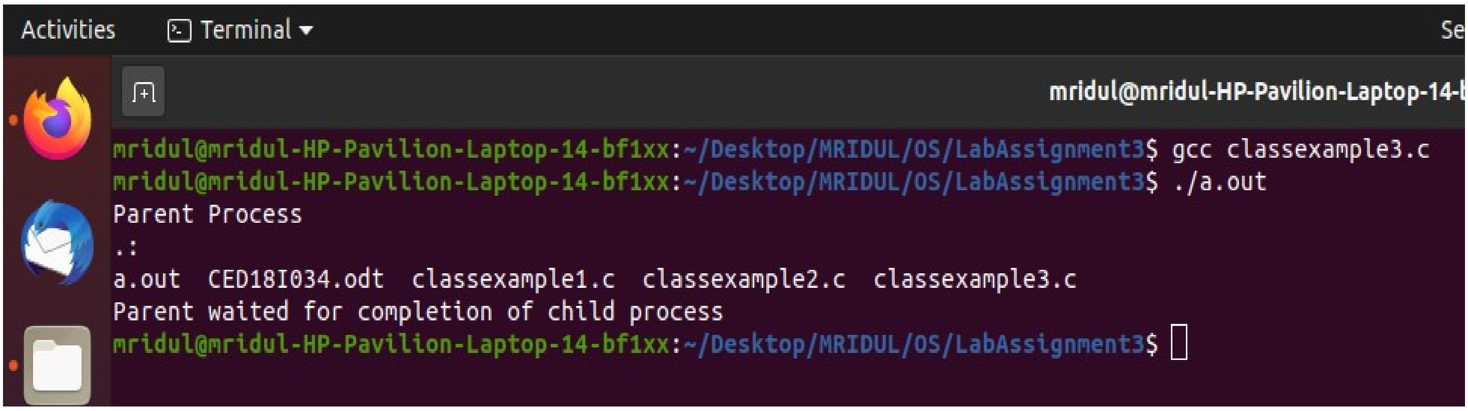
->file: Executable file.
->name: Name of the command to be executed.
->arg: Arguments to be passed on to run the function.

The parent process waits for the child to complete.

Code :-

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/types.h>
#include<sys/wait.h>
#include<unistd.h>

int main()
{
	char *args[] = {"/bin/ls", "-LR", NULL};

	pid_t pid = fork();

	if(pid < 0)
	{
		printf("Fork failed\n");
	}
	else if(pid == 0)
	{
		execv("/bin/ls", args);
	}
	else
	{
		printf("Parent Process\n");
		wait(NULL);
		printf("Parent waited for completion of child 
process\n");
	}

exit(0);
}
```

Output :-



```
mridul@mridul-HP-Pavilion-Laptop-14-bf1xx:~/Desktop/MRIDUL/OS/LabAssignment3$ gcc classexample3.c
mridul@mridul-HP-Pavilion-Laptop-14-bf1xx:~/Desktop/MRIDUL/OS/LabAssignment3$ ./a.out
Parent Process
.:
a.out  CED18I034.odt  classexample1.c  classexample2.c  classexample3.c
Parent waited for completion of child process
mridul@mridul-HP-Pavilion-Laptop-14-bf1xx:~/Desktop/MRIDUL/OS/LabAssignment3$ 
```

Explanation :-

Here the child function is executed so ls is executed.

int execv(const char*__path, char*const*__argv)
->path: Path of the executable file.
->argv: Arguments to be passed on to run the function.

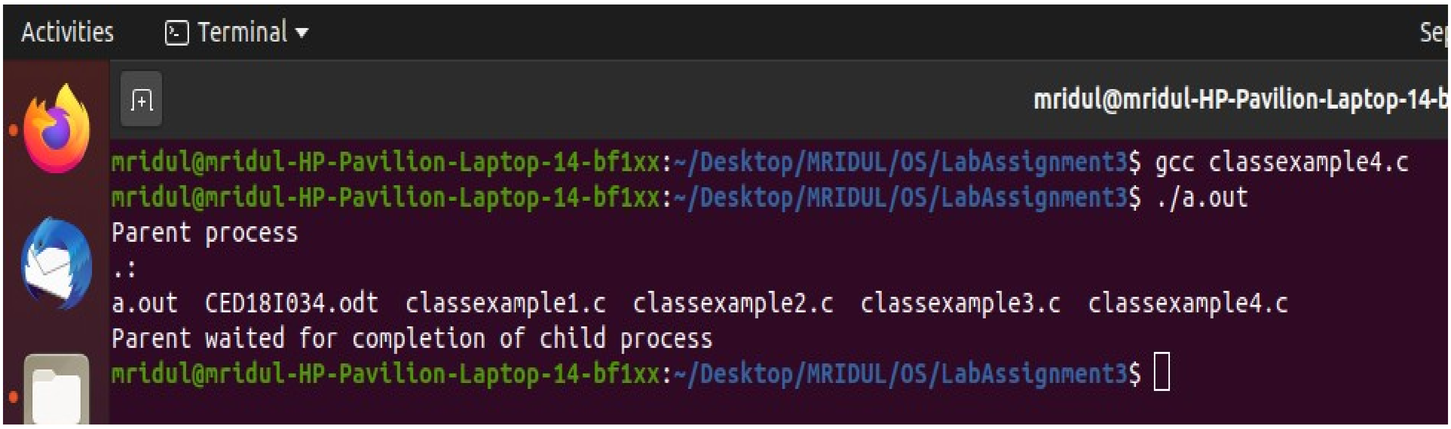
The parent process waits for the child to complete.

Code :-

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/types.h>
#include<sys/wait.h>
#include<unistd.h>

int main()
{
	char *args[] = {"/bin/ls", "-LR", NULL};

	pid_t pid = fork();
```

```
        if(pid < 0)
        {
                printf("Fork Failed\n");
        }
        else if(pid == 0)
        {
                execvp("ls", args);
        }
        else
        {
                printf("Parent process\n");
                wait(NULL);
                printf("Parent waited for completion of child
process\n");
        }

exit(0);
}
```

Output :-



```
mridul@mridul-HP-Pavilion-Laptop-14-bf1xx:~/Desktop/MRIDUL/OS/LabAssignment3$ gcc classexample4.c
mridul@mridul-HP-Pavilion-Laptop-14-bf1xx:~/Desktop/MRIDUL/OS/LabAssignment3$ ./a.out
Parent process
.:
a.out  CED18I034.odt  classexample1.c  classexample2.c  classexample3.c  classexample4.c
Parent waited for completion of child process
mridul@mridul-HP-Pavilion-Laptop-14-bf1xx:~/Desktop/MRIDUL/OS/LabAssignment3$ 
```

Explanation :-

Here the child function is executed so ls is executed.

int execvp(const char*__file, char *const*__argv)

->path: Path of the executable file.
->argv: Arguments to be passed on to run the function.

The parent process waits for the child to complete.

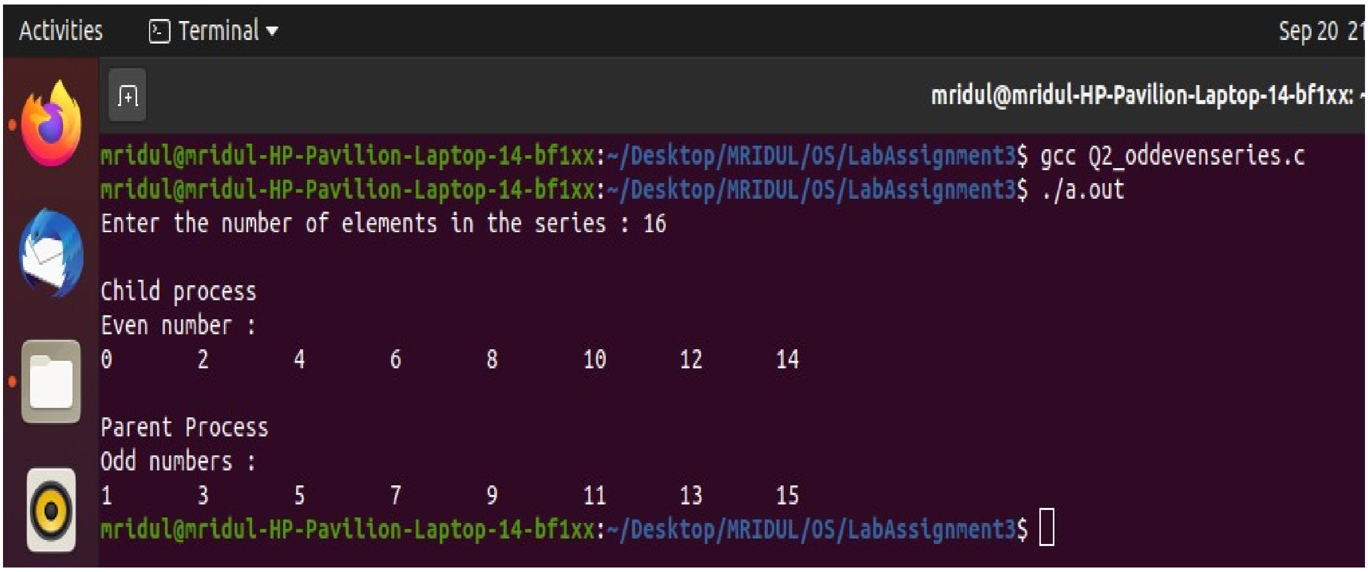
Question 2(a): Odd and Even series generation for n terms using Parent Childrelationship (say odd is the duty ofthe parent and even series as that of child).

Code :-

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/types.h>
#include<sys/wait.h>
#include<unistd.h>

int main()
{
	int n;
	printf("Enter the number of elements in the series : ");
	fflush(stdin);
	scanf("%d", &n);

	pid_t pid = fork();

	if(pid < 0)
	{
		printf("Fork Failed\n");
	}
	else if(pid == 0)
	{
		printf("\nChild process\n");
		printf("Even number : \n");
```

```
            for(int i = 0; i < n; i = i+2)
            {
                printf("%d\t", i);
            }
            printf("\n");
        }
        else
        {
            wait(NULL);
            printf("\nParent Process\n");
            printf("Odd numbers : \n");

            for(int i = 1; i < n; i = i+2)
            {
                printf("%d\t", i);
            }
            printf("\n");
        }
exit(0);
}
```

Output :-



```
mridul@mridul-HP-Pavilion-Laptop-14-bf1xx:~/Desktop/MRIDUL/OS/LabAssignment3$ gcc Q2_oddevenseries.c
mridul@mridul-HP-Pavilion-Laptop-14-bf1xx:~/Desktop/MRIDUL/OS/LabAssignment3$ ./a.out
Enter the number of elements in the series : 16

Child process
Even number :
0       2       4       6       8       10      12      14

Parent Process
Odd numbers :
1       3       5       7       9       11      13      15
mridul@mridul-HP-Pavilion-Laptop-14-bf1xx:~/Desktop/MRIDUL/OS/LabAssignment3$ 
```

Explanation :-

The child process prints the even numbers till then the parent process waits.After the child process is completed, the parent process prints the oddnumbers.

Question 2(b): Given a series of n numbers (u can assume natural numbers till n)generate the sum of odd terms inthe parent and the sum of even terms in the child process

Code :-

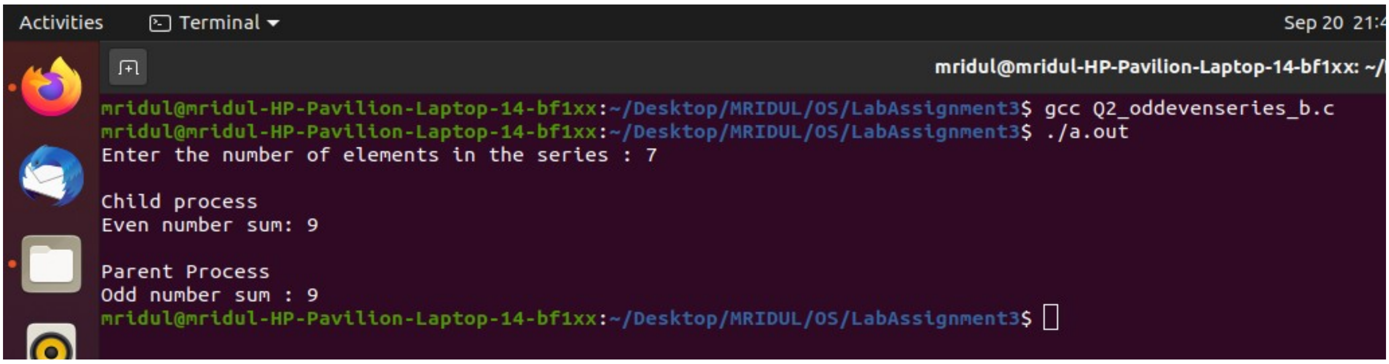
```
#include<stdio.h>
#include<stdlib.h>
#include<sys/types.h>
#include<sys/wait.h>
#include<unistd.h>

int main()
{
	int n; int x;
	printf("Enter the number of elements in the series : ");
	fflush(stdin);
	scanf("%d", &n);

	pid_t pid = fork();

	if(pid < 0)
	{
		printf("Fork Failed\n");
	}
	else if(pid == 0)
	{
		x = 0;
		printf("\nChild process\n");
```

```
        printf("Even number sum: ");

        for(int i = 1; i < n; i = i+2)
        {
            x = x+i;
        }

        printf("%d\t", x);
        printf("\n");
    }
    else
    {
        wait(NULL);
        x = 0;
        printf("\nParent Process\n");
        printf("Odd number sum : ");

        for(int i = 1; i < n; i = i+2)
        {
            x = x+i;
        }

        printf("%d\t", x);
        printf("\n");
    }
exit(0);
}
```

Explanation :-

The child process prints the sum of even indices till then the parent processwaits. After the child process is completed, the parent process prints the sumof odd indices.

Output :-



```
mridul@mridul-HP-Pavilion-Laptop-14-bf1xx:~/Desktop/MRIDUL/OS/LabAssignment3$ gcc Q2_oddevenseries_b.c
mridul@mridul-HP-Pavilion-Laptop-14-bf1xx:~/Desktop/MRIDUL/OS/LabAssignment3$ ./a.out
Enter the number of elements in the series : 7

Child process
Even number sum: 9

Parent Process
Odd number sum : 9
mridul@mridul-HP-Pavilion-Laptop-14-bf1xx:~/Desktop/MRIDUL/OS/LabAssignment3$ 
```

Question 3: Armstrong number generation within a range. The digit extraction,cubing can be the responsibility of the child while the checking for sum== no can happen in the child and the output list in the child.

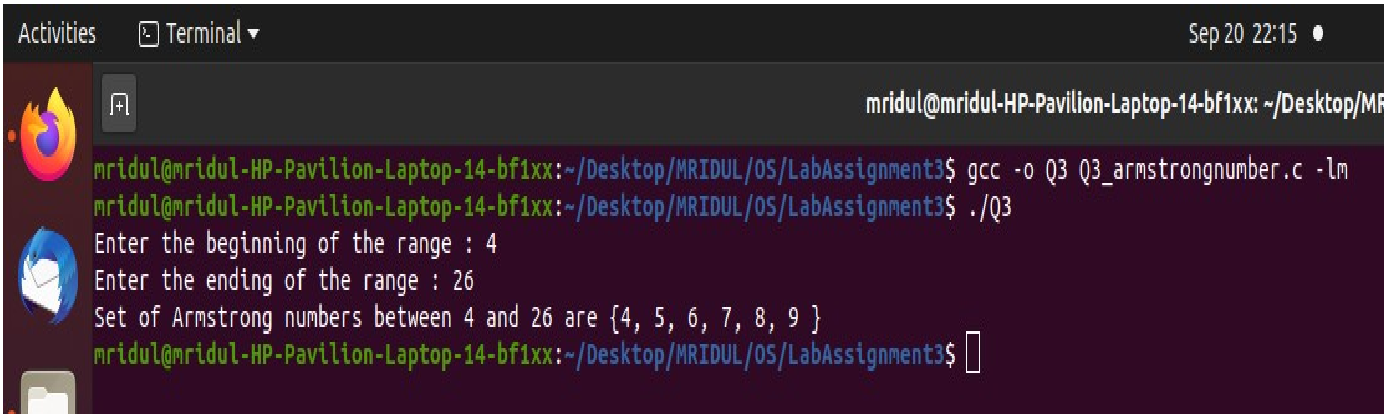
Code :-

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/types.h>
#include<sys/wait.h>
#include<unistd.h>
#include<math.h>

int main()
{
        int begin; int end;
        printf("Enter the beginning of the range : ");
        fflush(stdin);
        scanf("%d", &begin);
        printf("Enter the ending of the range : ");
        fflush(stdin);
        scanf("%d", &end);
```

```
int n = end - begin + 1;
int temp; int count = 0; int digit;
int a[n];

for(int i = 0; i < n; i = i+1)
{
	a[i] = 0;
}

pid_t pid = vfork();

if(pid < 0)
{
	printf("Fork failed\n");
}
else if(pid == 0)
{
	for(int i = begin; i < end + 1; i = i+1)
	{
		temp = i;
		while(temp != 0)
		{
			temp /= 10;
			count = count + 1;
		}

		temp = i;
		while(temp != 0)
		{
			digit = temp % 10;
			temp /= 10;
			a[i - begin] += pow(digit, count);
		}
		count = 0;
	}
```

```
        }
        else
        {
                wait(NULL);
                printf("Set of Armstrong numbers between %d and %d are {", begin, end);
                for(int i = begin; i < end + 1; i = i+1)
                {
                        if(a[i - begin] == i)
                        {
                                printf("%d, ", i);
                        }
                }
        printf("\b\b }\n");
        }
exit(0);
}
```

Output :-



```
mridul@mridul-HP-Pavilion-Laptop-14-bf1xx:~/Desktop/MRIDUL/OS/LabAssignment3$ gcc -o Q3 Q3_armstrongnumber.c -lm
mridul@mridul-HP-Pavilion-Laptop-14-bf1xx:~/Desktop/MRIDUL/OS/LabAssignment3$ ./Q3
Enter the beginning of the range : 4
Enter the ending of the range : 26
Set of Armstrong numbers between 4 and 26 are {4, 5, 6, 7, 8, 9 }
mridul@mridul-HP-Pavilion-Laptop-14-bf1xx:~/Desktop/MRIDUL/OS/LabAssignment3$ 
```

Explanation :-

The child process computes the sum of the numbers raised to the power ofthe length of the number. After the child process

completes, the parentprocess prints the armstrong numbers from the given range.

Question 4: Fibonacci Series AND Prime parent child relationship (say parent doesfib Number generation usingseries and child does prime series)

Code :-

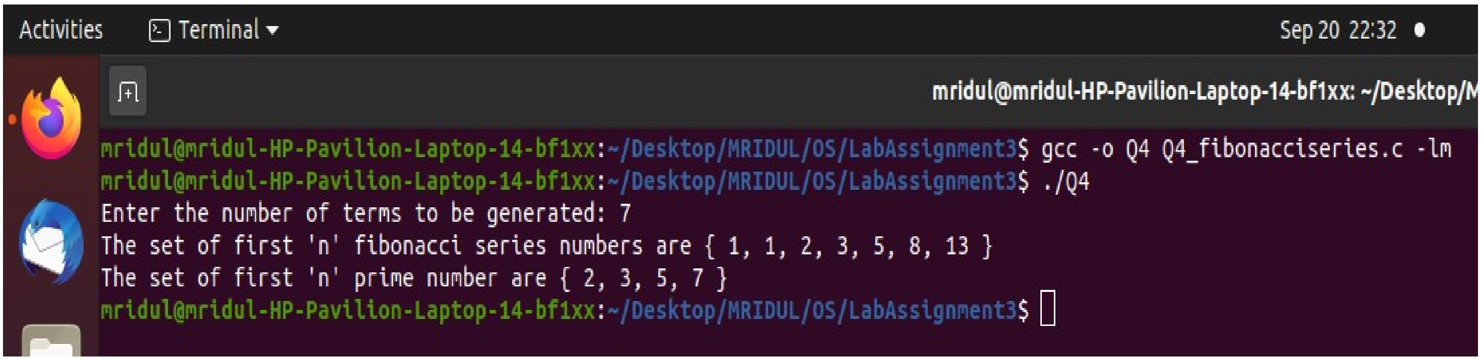
```
#include<stdio.h>
#include<stdlib.h>
#include<sys/types.h>
#include<sys/wait.h>
#include<unistd.h>
#include<math.h>

void prime(int n);
int fib(int n);
void fibgen(int n);

int main()
{
    int n;
    printf("Enter the number of terms to be generated: ");
    fflush(stdin);
    scanf("%d", &n);

    pid_t pid = fork();

    if(pid < 0)
    {
        printf("Fork Failed\n");
    }
    else if(pid == 0)
    {
```

```
                fibgen(n);
        }
        else
        {
                wait(NULL);
                prime(n);
        }

exit(0);
}

void prime(int n)
{
        int flag;

        printf("The set of first 'n' prime number are { ");

        for(int i = 1; i <= n; i = i+1)
        {
                if(i == 1 || i == 0)
                {
                        continue;
                }

                flag = 1;

                for(int j = 2; j <= sqrt(i); ++j)
                {
                        if(i % j == 0)
                        {
                                flag = 0;
                                break;
                        }
                }
        if(flag == 1)
        {
            printf("%d, ", i);
        }
    }

    printf("\b\b }\n");
}

int fib(int n)
{
    if(n <= 1)
        return n;

    return fib(n-1) + fib(n-2);
}

void fibgen(int n)
{
    printf("The set of first 'n' fibonacci series numbers are { ");

    for(int i = 0; i < n; i = i+1)
    {
        printf("%d, ", fib(i + 1));
    }
    printf("\b\b }\n");
}
```

Explanation :-

The child process prints the first ‘n’ fibonacci numbers. After the childprocess is completed, the parent process prints the prime numbers whichare less than ‘n’.

Output :-



```
mridul@mridul-HP-Pavilion-Laptop-14-bf1xx:~/Desktop/MRIDUL/OS/LabAssignment3$ gcc -o Q4 Q4_fibonacciseries.c -lm
mridul@mridul-HP-Pavilion-Laptop-14-bf1xx:~/Desktop/MRIDUL/OS/LabAssignment3$ ./Q4
Enter the number of terms to be generated: 7
The set of first 'n' fibonacci series numbers are { 1, 1, 2, 3, 5, 8, 13 }
The set of first 'n' prime number are { 2, 3, 5, 7 }
mridul@mridul-HP-Pavilion-Laptop-14-bf1xx:~/Desktop/MRIDUL/OS/LabAssignment3$ 
```

Question 5: Ascending Order sort within Parent and Descending order sort (or viceversa) within the childprocess of an input array. (u can view as two different outputs –firstentire array is asc order sorted in opand then the second part desc order output)

Code :-

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/types.h>
#include<sys/wait.h>
#include<unistd.h>
#include<stdbool.h>

void bubsort(int n, int *array, bool comp(const void* , const void*));
bool asc(const void* a, const void* b);
bool desc(const void* a, const void* b);
void print(int n, int* array);
void swap(int* x, int *y);

int main()
{
	int n;
	printf("Enter the number of elements: ");
```

```
fflush(stdin);
scanf("%d", &n);
int a[n];

for(int i = 0; i < n; i = i+1)
{
    printf("Enter number %d: ", i+1);
    fflush(stdin);
    scanf("%d", &a[i]);
}

pid_t pid = fork();

if(pid < 0)
{
    printf("Fork failed\n");
}
else if(pid == 0)
{
    printf("\nAscending sort\n");
    pid_t child = vfork();

    if(child < 0)
    {
        printf("Fork Failed\n");
    }
    else if(child == 0)
    {
        bubsort(n, a, asc);
    }
    else
    {
        wait(NULL);
        print(n, a);
    }
```

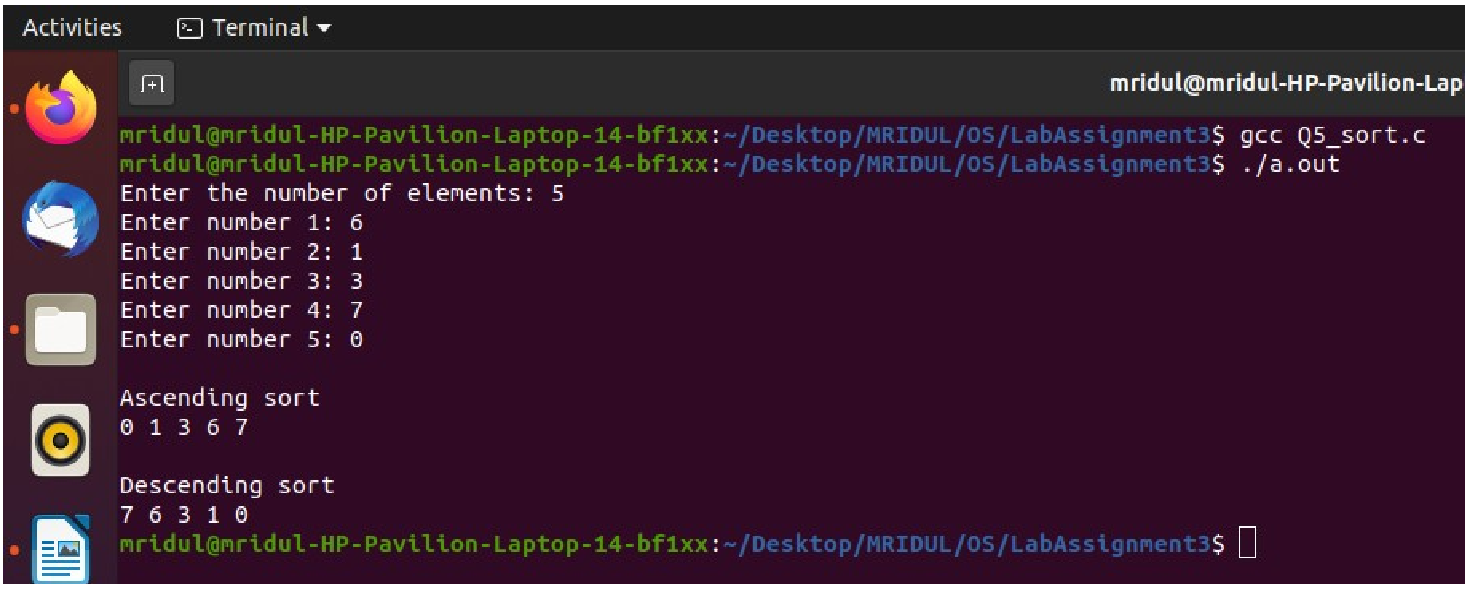
```
	}
	else
	{
		wait(NULL);
		printf("\nDescending sort\n");
		pid_t parent = vfork();

		if(parent < 0)
		{
			printf("Fork Failed\n");
		}
		else if(parent == 0)
		{
			bubsort(n, a, desc);
		}
		else
		{
			wait(NULL);
			print(n, a);
		}
	}

exit(0);
}

void bubsort(int n, int *array, bool comp(const void* , const void*))
{
	for(int i = 0; i < n-1; i = i+1)
	{
		for(int j = 0; j < n - i - 1; j = j+1)
		{
			if(comp(&array[j], &array[j+1]))
			{
				swap(&array[j], &array[j+1]);
```

```
                    }
            }
    }
}

bool asc(const void* a, const void* b)
{
    return *(int*)a > *(int*)b;
}

bool desc(const void* a, const void* b)
{
    return *(int*)a < *(int*)b;
}

void print(int n, int *array)
{
    for(int i = 0; i < n; i = i+1)
    {
        printf("%d ", array[i]);
    }
    printf("\n");
}

void swap(int *x, int *y)
{
    int c = *x;
    *x = *y;
    *y = c;
}
```

Explanation :-

The child process prints the input array into ascending order. After the childprocess completes , the parent process prints the same array in thedescending order.

Output :-



```
mridul@mridul-HP-Pavilion-Laptop-14-bf1xx:~/Desktop/MRIDUL/OS/LabAssignment3$ gcc Q5_sort.c
mridul@mridul-HP-Pavilion-Laptop-14-bf1xx:~/Desktop/MRIDUL/OS/LabAssignment3$ ./a.out
Enter the number of elements: 5
Enter number 1: 6
Enter number 2: 1
Enter number 3: 3
Enter number 4: 7
Enter number 5: 0

Ascending sort
0 1 3 6 7

Descending sort
7 6 3 1 0
mridul@mridul-HP-Pavilion-Laptop-14-bf1xx:~/Desktop/MRIDUL/OS/LabAssignment3$ 
```

Question 6: Given an input array use parent child relationship to sort the first halfof array in ascending order andthe trailing half in descending order (parent / child is ur choice)

Code :-

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/types.h>
#include<sys/wait.h>
#include<unistd.h>
#include<stdbool.h>

void bubsort(int n, int *array, bool comp(const void* , const
void*));
bool asc(const void* a, const void* b);
bool desc(const void* a, const void* b);
void print(int n, int* array);
void swap(int* x, int *y);
```

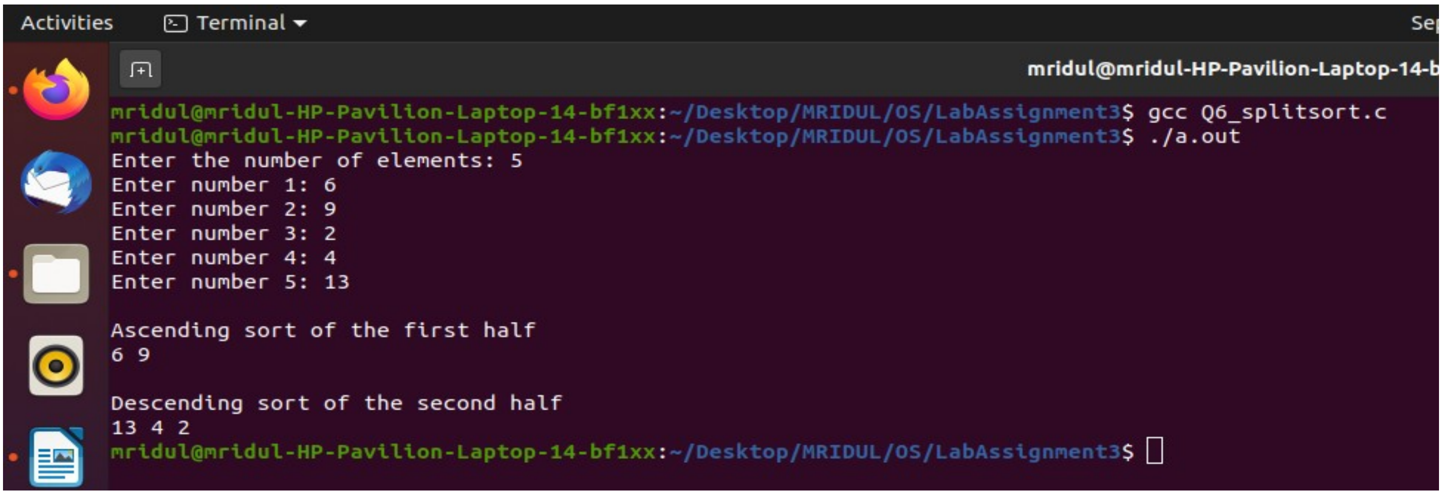
```
int main()
{
        int n; int mid; int x = 0;
        printf("Enter the number of elements: ");
        fflush(stdin);
        scanf("%d", &n);
        mid = n/2;
        int a1[mid]; int a2[n - mid];

        for(int i = 0; i < mid; i = i+1)
        {
                printf("Enter number %d: ", ++x);
                fflush(stdin);
                scanf("%d", &a1[i]);
        }

        for(int i = 0; i < n - mid; i = i+1)
        {
                printf("Enter number %d: ", ++x);
                fflush(stdin);
                scanf("%d", &a2[i]);
        }

        pid_t pid = fork();

        if(pid < 0)
        {
                printf("Fork failed\n");
        }
        else if(pid == 0)
        {
                printf("\nAscending sort of the first half\n");
                pid_t child = vfork();
```

```
		if(child < 0)
		{
			printf("Fork Failed\n");
		}
		else if(child == 0)
		{
			bubsort(mid, a1, asc);
		}
		else
		{
			wait(NULL);
			print(mid, a1);
		}
	}
	else
	{
		wait(NULL);
		printf("\nDescending sort of the second half\n");
		pid_t parent = vfork();

		if(parent < 0)
		{
			printf("Fork Failed\n");
		}
		else if(parent == 0)
		{
			bubsort(n - mid, a2, desc);
		}
		else
		{
			wait(NULL);
			print(n - mid, a2);
		}
	}
```

```
exit(0);
}

void bubsort(int n, int *array, bool comp(const void* , const void*))
{
    for(int i = 0; i < n-1; i = i+1)
    {
        for(int j = 0; j < n - i - 1; j = j+1)
        {
            if(comp(&array[j], &array[j+1]))
            {
                swap(&array[j], &array[j+1]);
            }
        }
    }
}

bool asc(const void* a, const void* b)
{
    return *(int*)a > *(int*)b;
}

bool desc(const void* a, const void* b)
{
    return *(int*)a < *(int*)b;
}

void print(int n, int *array)
{
    for(int i = 0; i < n; i = i+1)
    {
        printf("%d ", array[i]);
    }
    printf("\n");
```

```
}

void swap(int *x, int *y)
{
	int c = *x;
	*x = *y;
	*y = c;
}
```

Output :-



```
mridul@mridul-HP-Pavilion-Laptop-14-bf1xx:~/Desktop/MRIDUL/OS/LabAssignment3$ gcc Q6_splitsort.c
mridul@mridul-HP-Pavilion-Laptop-14-bf1xx:~/Desktop/MRIDUL/OS/LabAssignment3$ ./a.out
Enter the number of elements: 5
Enter number 1: 6
Enter number 2: 9
Enter number 3: 2
Enter number 4: 4
Enter number 5: 13

Ascending sort of the first half
6 9

Descending sort of the second half
13 4 2
mridul@mridul-HP-Pavilion-Laptop-14-bf1xx:~/Desktop/MRIDUL/OS/LabAssignment3$ 
```

Explanation :-

The child process prints the first half of the array int the ascending order.After the child process completes, the parent process prints the later half inthe descending order.

Question 7: Ascending Order sort within Parent and Descending order sort (or viceversa) within the childprocess of an input array. (u can view as two different outputs –firstentire array is asc order sorted in opand then the second part desc order output)

Code :-

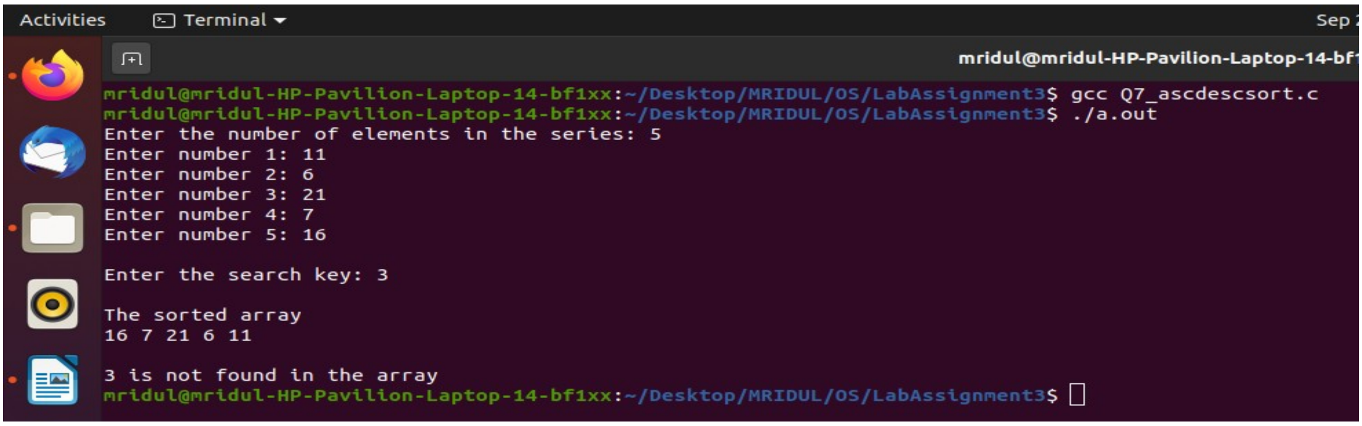
```
#include<stdio.h>
#include<stdlib.h>
#include<sys/types.h>
#include<sys/wait.h>
#include<unistd.h>
#include<stdbool.h>

int binarysearch(int *a, int n, int x);
void bubsort(int n, int *array);
void print(int n, int *array);
void swap(int *x, int *y);

int main()
{
	int n; int mid; int x; int count = 0;
	printf("Enter the number of elements in the series: ");
	fflush(stdin);
	scanf("%d", &n);
	int a[n];

	for(int i = 0; i < n; i = i+1)
	{
		printf("Enter number %d: ", i+1);
		fflush(stdin);
		scanf("%d", &a[i]);
	}

	printf("\nEnter the search key: ");
	fflush(stdin);
	scanf("%d", &x);
	bubsort(n, a);
	printf("\nThe sorted array\n");
	fflush(stdin);
```

```
print(n, a);

pid_t pid = vfork();

if(pid < 0)
{
        printf("Fork failed\n");
}
else if(pid == 0)
{
        if((mid = binarysearch(a, n, x)) == -1)
        {
                printf("\n%d is not found in the array\n", x);
                exit(1);
        }
        else
        {
                printf("\n%d found at index %d\n", x, mid);
        }
        exit(0);
}
else
{
        wait(NULL);
        int i = mid -1;
        while(i > -1 && a[i] == x)
        {
                printf("%d found at index %d\n", x, i);
                i = i-1;
        }
        i = mid + 1;

        while(i < n && a[i] == x)
        {
                printf("%d found at index %d\n", x, i);
```

```
            i = i+1;
        }
    }

exit(0);
}

int binarysearch(int *a, int n, int x)
{
    int l = 0; int r = n-1;
    while(l <= r)
    {
        int m = l + (r - l)/2;
        if(a[m] == x)
        {
            return m;
        }
        else if(a[m] < x)
        {
            l = m + 1;
        }
        else
        {
            r = m - 1;
        }
    }
    return -1;
}

void bubsort(int n, int *array)
{
    for(int i = 0; i < n-1; i = i+1)
    {
        for(int j = 0; j < n - i - 1; j = j+1)
        {
```

```
                        if((array[j], array[j+1]))
                        {
                                swap(&array[j], &array[j+1]);
                        }
                }
        }
}

void print(int n, int *array)
{
        for(int i = 0; i < n; i = i+1)
        {
                printf("%d ", array[i]);
        }
        printf("\n");
}

void swap(int *x, int *y)
{
        int c = *x;
        *x = *y;
        *y = c;
}
```

<u>Output</u> :-



```
mridul@mridul-HP-Pavilion-Laptop-14-bf1xx:~/Desktop/MRIDUL/OS/LabAssignment3$ gcc Q7_ascdescsort.c
mridul@mridul-HP-Pavilion-Laptop-14-bf1xx:~/Desktop/MRIDUL/OS/LabAssignment3$ ./a.out
Enter the number of elements in the series: 5
Enter number 1: 11
Enter number 2: 6
Enter number 3: 21
Enter number 4: 7
Enter number 5: 16

Enter the search key: 3

The sorted array
16 7 21 6 11

3 is not found in the array
mridul@mridul-HP-Pavilion-Laptop-14-bf1xx:~/Desktop/MRIDUL/OS/LabAssignment3$ 
```

Explanation :-

The input array must be sorted for binary search to work so if the input arrayis not sorted then the array is sorted first.Then the child process searches for the input search element. After the childprocess is completed then the parent process find the sameelement/duplicates (if any) in the array.