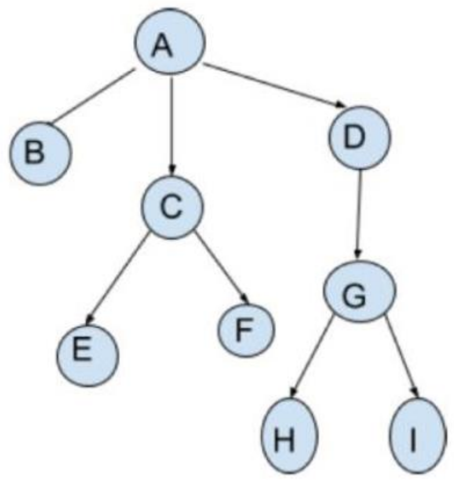
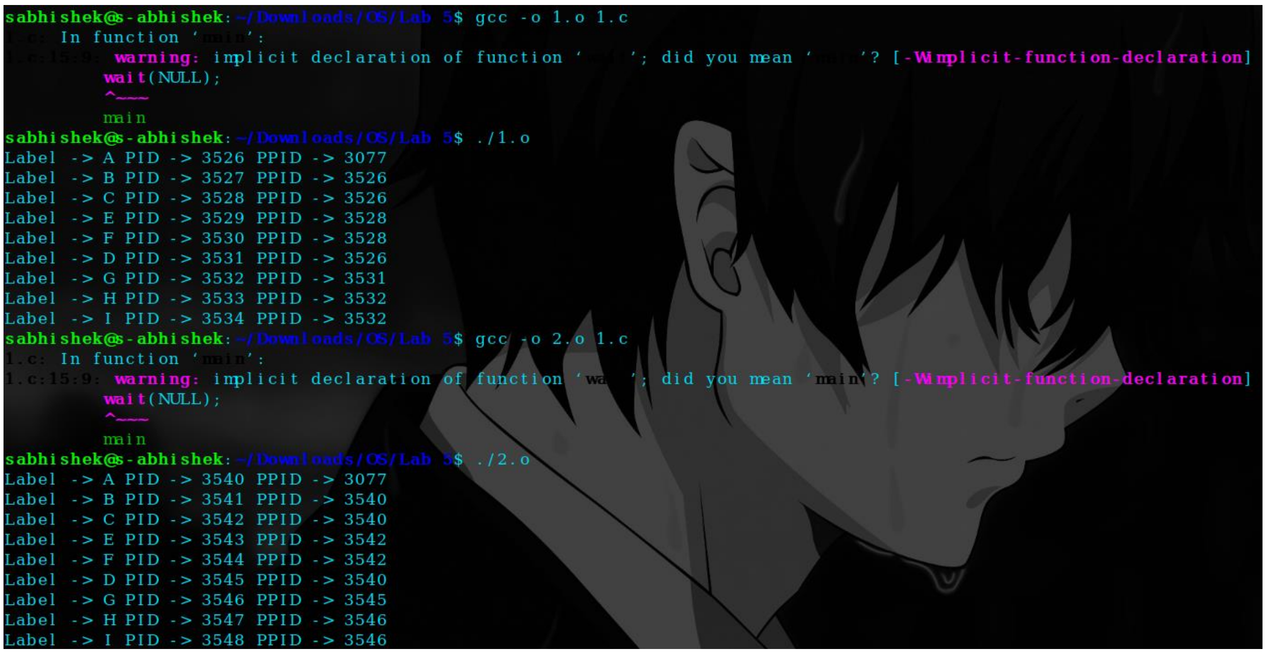
S ABHISHEK **AM.EN.U4CSE19147**

OPERATING SYSTEM LAB 5

A program to create processes according to the tree structure given below.



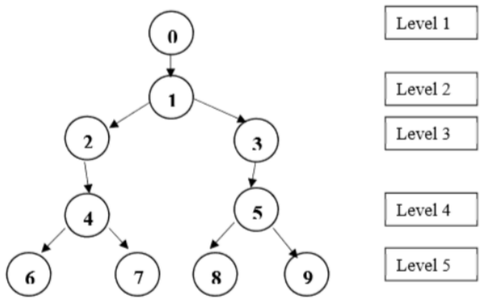
```
#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>


int main()

{


    pid_t pid = getpid();

    pid_t pptd = getppid();


    printf("Label -> A PID -> %d PPID -> %d\n", getpid(),getppid());
    if(fork())
    {
        wait(NULL);
        if(fork())
        {
            wait(NULL);
            if(!fork())
            {
                printf("Label -> D PID -> %d PPID -> %d\n", getpid(),getppid());
                if(!fork())
                {
                    printf("Label -> G PID -> %d PPID -> %d\n", getpid(),getppid());
                    if(fork())
                    {
                        wait(NULL);
                        if(!fork())
                        {
                            printf("Label -> I PID -> %d PPID -> %d\n",
getpid(),getppid());
                        }
                        else
                        {
                            wait(NULL);
                        }
                    }
                    else
                    {
                        printf("Label -> H PID -> %d PPID -> %d\n",
getpid(),getppid());
                    }
                }
                else
                {
                    wait(NULL);
                }
            }
            else
            {
                wait(NULL);
            }
        }
        else
        {
            printf("Label -> C PID -> %d PPID -> %d\n", getpid(),getppid());
            if(fork())
            {
                wait(NULL);
                if(!fork())
                {
                    printf("Label -> F PID -> %d PPID -> %d\n", getpid(),getppid());
                }
                else{
                    wait(NULL);
                  }
            }
            else{
                printf("Label -> E PID -> %d PPID -> %d\n", getpid(),getppid());
            }
        }
    }
    else {
        printf("Label -> B PID -> %d PPID -> %d\n", getpid(),getppid());
    }
    return 0;
}
```



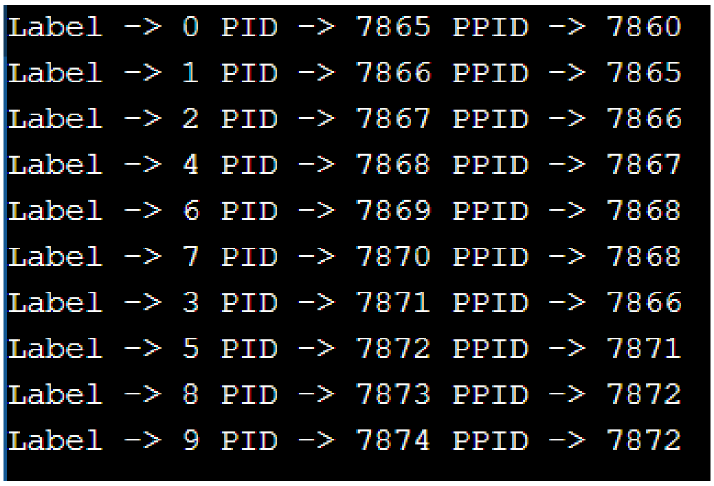
```
sabhishek@s-abhishek:~/Downloads/OS/Lab 5$ gcc -o 1.o 1.c
1.c: In function 'main':
1.c:15:9: warning: implicit declaration of function 'wait'; did you mean 'main'? [-Wimplicit-function-declaration]
         wait(NULL);
         ^~~~
         main
sabhishek@s-abhishek:~/Downloads/OS/Lab 5$ ./1.o
Label -> A PID -> 3526 PPID -> 3077
Label -> B PID -> 3527 PPID -> 3526
Label -> C PID -> 3528 PPID -> 3526
Label -> E PID -> 3529 PPID -> 3528
Label -> F PID -> 3530 PPID -> 3528
Label -> D PID -> 3531 PPID -> 3526
Label -> G PID -> 3532 PPID -> 3531
Label -> H PID -> 3533 PPID -> 3532
Label -> I PID -> 3534 PPID -> 3532
sabhishek@s-abhishek:~/Downloads/OS/Lab 5$ gcc -o 2.o 1.c
1.c: In function 'main':
1.c:15:9: warning: implicit declaration of function 'wait'; did you mean 'main'? [-Wimplicit-function-declaration]
         wait(NULL);
         ^~~~
         main
sabhishek@s-abhishek:~/Downloads/OS/Lab 5$ ./2.o
Label -> A PID -> 3540 PPID -> 3077
Label -> B PID -> 3541 PPID -> 3540
Label -> C PID -> 3542 PPID -> 3540
Label -> E PID -> 3543 PPID -> 3542
Label -> F PID -> 3544 PPID -> 3542
Label -> D PID -> 3545 PPID -> 3540
Label -> G PID -> 3546 PPID -> 3545
Label -> H PID -> 3547 PPID -> 3546
Label -> I PID -> 3548 PPID -> 3546
```

- Even though the program is compiled several times, the order of the execution of the process is same.
- The order of execution is A-B-C-E-F-D-G-H-I.
- Only the process ID (Parent ID and Child ID) changes.
- Parent Process calls the fork() which creates the child process and using wait() system call the parent process waits until the child terminates.

2. Write a program to create processes according to the tree structure given below. All processes should print their Process id and Parent Process id and the label given in the diagram.



```
#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>


int main()

{

    pid_t pid = getpid();

    pid_t pptd = getppid();

    printf("Label -> 0 PID -> %d PPID -> %d\n", getpid(),getppid());


    if(!fork())
    {
        printf("Label -> 1 PID -> %d PPID -> %d\n", getpid(),getppid());
        if(fork())
        {
            wait(NULL);
            if(!fork())
            {
                printf("Label -> 3 PID -> %d PPID -> %d\n", getpid(),getppid());
                if(!fork())
                {
                    printf("Label -> 5 PID -> %d PPID -> %d\n", getpid(),getppid());
                    if(fork())
                    {
                        wait(NULL);
                        if(!fork())
                        {
                            printf("Label -> 9 PID -> %d PPID -> %d\n",
getpid(),getppid());
                        }
                        else
                        {
                            wait(NULL);
                        }
                    }
                    else
                    {
                        printf("Label -> 8 PID -> %d PPID -> %d\n",
getpid(),getppid());
                    }
                }
                else
                {
                    wait(NULL);
                }
            }
            else
            {
                wait(NULL);
            }
        }
        else
        {
            printf("Label -> 2 PID -> %d PPID -> %d\n", getpid(),getppid());
            if(!fork())
            {
                printf("Label -> 4 PID -> %d PPID -> %d\n", getpid(),getppid());
                if(fork())
                {
                    wait(NULL);
                    if(!fork())
                    {
                        printf("Label -> 7 PID -> %d PPID -> %d\n",
getpid(),getppid());
                    }
                    else
                    {
                        wait(NULL);
                    }
                }
                else
                {
                    printf("Label -> 6 PID -> %d PPID -> %d\n", getpid(),getppid());
                }
            }
            else
            {
                wait(NULL);
            }
        }
    }
    else
    {
        wait(NULL);
    }
}
```

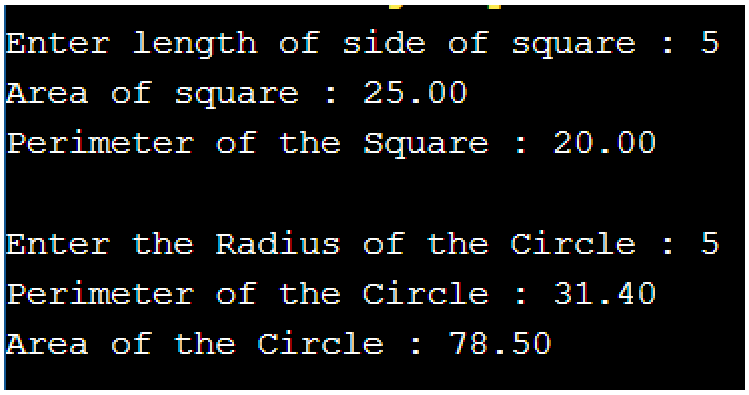


```
Label -> 0 PID -> 7865 PPID -> 7860
Label -> 1 PID -> 7866 PPID -> 7865
Label -> 2 PID -> 7867 PPID -> 7866
Label -> 4 PID -> 7868 PPID -> 7867
Label -> 6 PID -> 7869 PPID -> 7868
Label -> 7 PID -> 7870 PPID -> 7868
Label -> 3 PID -> 7871 PPID -> 7866
Label -> 5 PID -> 7872 PPID -> 7871
Label -> 8 PID -> 7873 PPID -> 7872
Label -> 9 PID -> 7874 PPID -> 7872
```

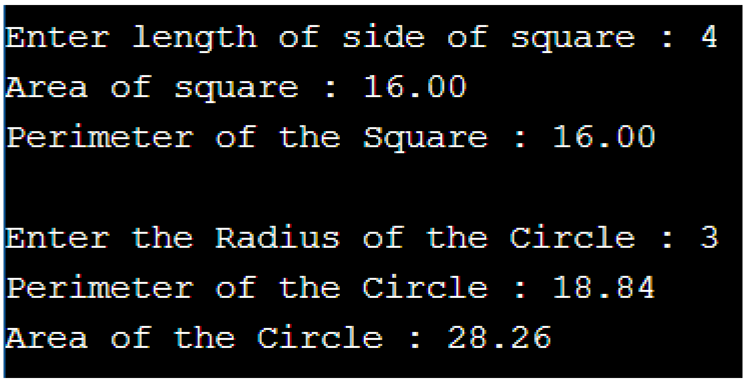
3. Write a program to find the area and perimeter of circle and square.

Create separate processes to perform the calculation of circle and square.

```
#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>


int main()

{

    if(fork())

    {

        wait(NULL);

        int radius;

        printf("\n\nEnter the Radius of the Circle : ");

        scanf("%d",&radius);
        printf("Perimeter of the Circle : %0.2f",2*3.14*radius);
         printf("\nArea of the Circle : %0.2f",3.14*radius*radius);
    }
    else
    {
        float side;
        printf("Enter length of side of square : ");
        scanf("%f", &side);
        printf("Area of square : %0.2f",side*side);
        printf("\nPerimeter of the Square : %0.2f",4*side);
    }
}
```



```
Enter length of side of square : 5
Area of square : 25.00
Perimeter of the Square : 20.00

Enter the Radius of the Circle : 5
Perimeter of the Circle : 31.40
Area of the Circle : 78.50
```



```
Enter length of side of square : 4
Area of square : 16.00
Perimeter of the Square : 16.00

Enter the Radius of the Circle : 3
Perimeter of the Circle : 18.84
Area of the Circle : 28.26
```

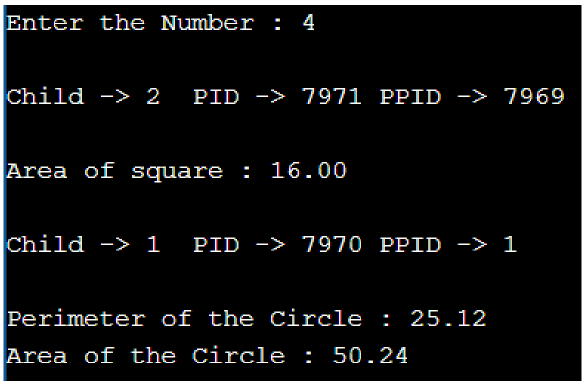
4. Modify the above program as follows: The parent process should create two children.

[User enters Value of variable ‘a’ only once].

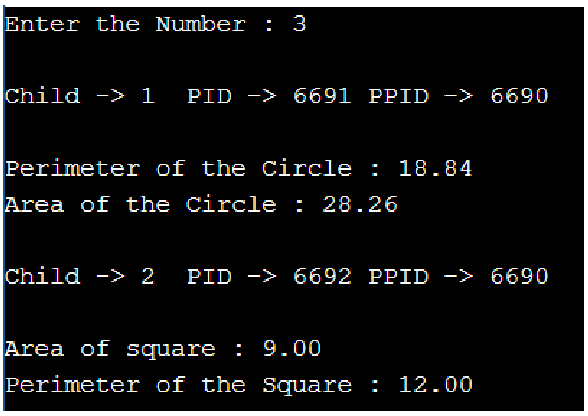
The first child finds the area and perimeter of a circle with radius ‘a’.

The Second child finds the area and perimeter of square with side ‘a’.

```
#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>


int main()

{

    float num;

    printf("Enter the Number : ");

    scanf("%f",&num);


    pid_t pid = getpid();

    pid_t pptd = getppid();


    if(fork())

    {

        if(!fork())

        {
            printf("\nChild -> 2  PID -> %d PPID -> %d\n", getpid(),getppid());
            printf("\nArea of square : %0.2f",num*num);
            printf("\nPerimeter of the Square : %0.2f",4*num);
         }
      }
      else
      {
         printf("\nChild -> 1  PID -> %d PPID -> %d\n", getpid(),getppid());
         printf("\nPerimeter of the Circle : %0.2f",2*3.14*num);
          printf("\nArea of the Circle : %0.2f\n",3.14*num*num);
      }
}
```



```
Enter the Number : 4

Child -> 2  PID -> 7971 PPID -> 7969

Area of square : 16.00

Child -> 1  PID -> 7970 PPID -> 1

Perimeter of the Circle : 25.12
Area of the Circle : 50.24
```

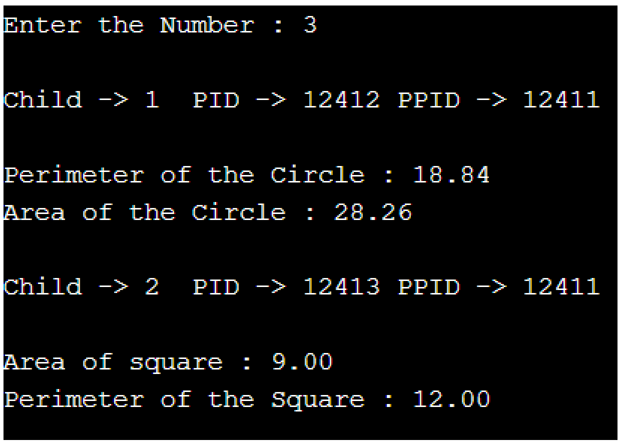


```
Enter the Number : 3

Child -> 1  PID -> 6691 PPID -> 6690

Perimeter of the Circle : 18.84
Area of the Circle : 28.26

Child -> 2  PID -> 6692 PPID -> 6690

Area of square : 9.00
Perimeter of the Square : 12.00
```

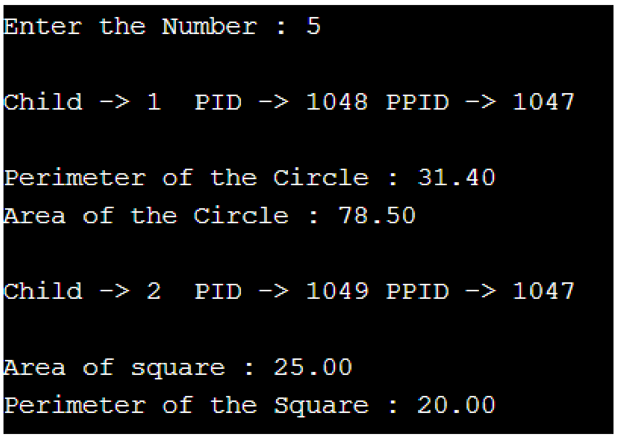
5. Modify the previous program to make the parent process wait until the completion of its children. [Hint. Use wait() system call]

```
#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>


int main()

{

    float num;

    printf("Enter the Number : ");

    scanf("%f",&num);


    pid_t pid = getpid();

    pid_t pptd = getppid();


    if(fork())

    {

        wait(NULL);

        if(!fork())

        {

            printf("\nChild -> 2  PID -> %d PPID -> %d\n", getpid(),getppid());

            printf("\nArea of square : %0.2f",num*num);

            printf("\nPerimeter of the Square : %0.2f",4*num);

        }
```

```
          else
          {
              wait(NULL);
          }
      }
      else
      {
          printf("\nChild -> 1  PID -> %d PPID -> %d\n", getpid(),getppid());
          printf("\nPerimeter of the Circle : %0.2f",2*3.14*num);
           printf("\nArea of the Circle : %0.2f\n",3.14*num*num);
      }
}
```



```
Enter the Number : 3

Child -> 1  PID -> 12412 PPID -> 12411

Perimeter of the Circle : 18.84
Area of the Circle : 28.26

Child -> 2  PID -> 12413 PPID -> 12411

Area of square : 9.00
Perimeter of the Square : 12.00
```



```
Enter the Number : 5

Child -> 1  PID -> 1048 PPID -> 1047

Perimeter of the Circle : 31.40
Area of the Circle : 78.50

Child -> 2  PID -> 1049 PPID -> 1047

Area of square : 25.00
Perimeter of the Square : 20.00
```

6. Create a parent process having two children. The first child should overwrite its address space with a process that prints “Happy new year” (happynewyear.c). The second child should overwrite its address space with another process that prints the sum of digits of a number entered by the user(sum.c).

[Hint: use exec family of system calls]

Sample output: The output should come in the following order

```
Happy new year

Enter the number: 123
Sum of Digits: 6
Parent exiting ...good bye.
```

Start.c

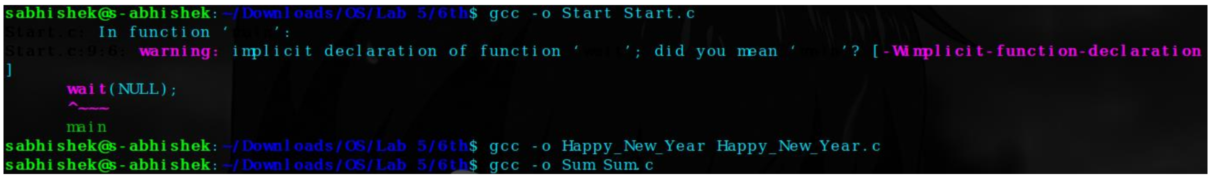
```
#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

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

{

   printf("ID of the Process ( Parent ) = %d\n", getpid());

   if(fork())

   {

        wait(NULL);

        if(!fork())

        {
                char *args[]={"./Happy_New_Year",NULL};
                execvp(args[0],args);
        }
        else
        {
                wait(NULL);
                printf("\nParent Process Terminating.....\n");
        }
    }
    else
    {
        char *args[]={"./Sum",NULL};
        execvp(args[0],args);
    }
}
```

Happy_New_Year.c

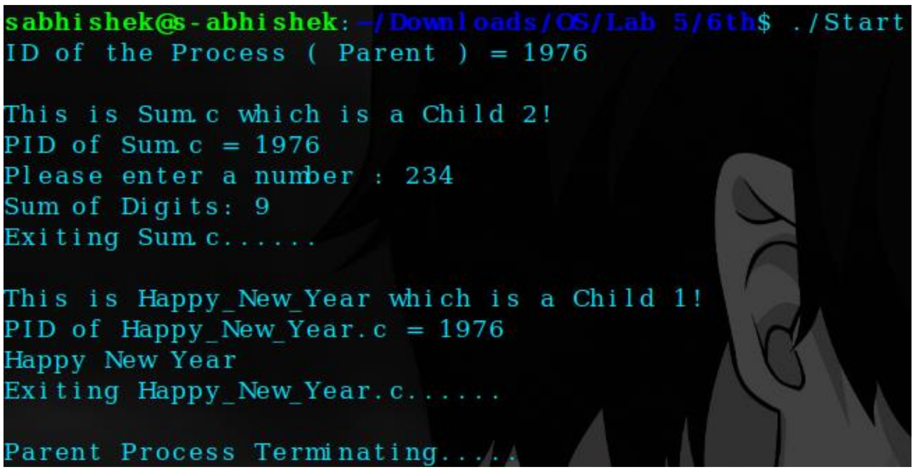
```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
int main()
{
        printf("\nThis is Happy_New_Year which is a Child 1!");
        printf("\nPID of Happy_New_Year.c = %d", getppid());
        printf("\nHappy New Year");
        printf("\nExiting Happy_New_Year.c......\n");
        return 0;
}
```

Sum.c

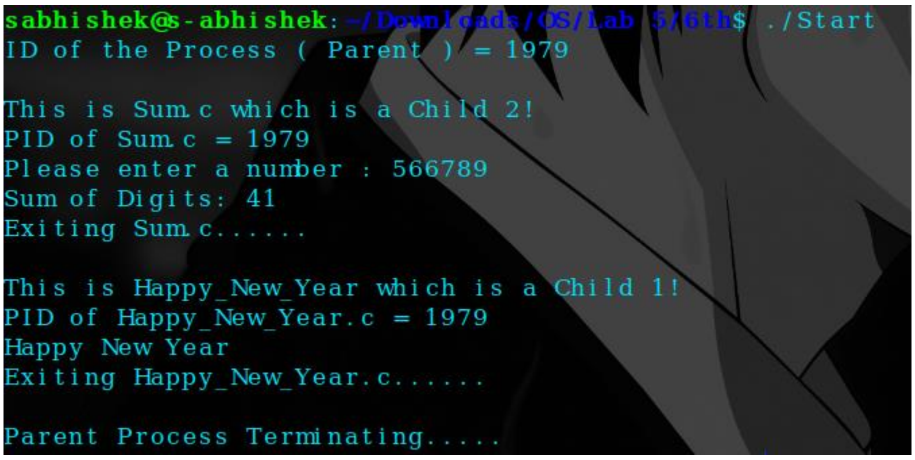
```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
int main()
{
        printf("\nThis is Sum.c which is a Child 2!");
        printf("\nPID of Sum.c = %d\n", getppid());
        printf("Please enter a number : ");
        int num;
        scanf("%d",&num);
        int sum=0;
        while(num>0)
        {
                int k=num%10;
                sum+=k;
                num=num/10;
        }
        printf("Sum of Digits: %d\n", sum);

        printf("Exiting Sum.c......\n");

        return 0;

}
```



```
sabhishek@s-abhishek:~/Downloads/OS/Lab 5/6th$ gcc -o Start Start.c
Start.c: In function ‘main’:
Start.c:9:6: warning: implicit declaration of function ‘wait’; did you mean ‘main’? [-Wimplicit-function-declaration
]
      wait(NULL);
      ^~~~
      main
sabhishek@s-abhishek:~/Downloads/OS/Lab 5/6th$ gcc -o Happy_New_Year Happy_New_Year.c
sabhishek@s-abhishek:~/Downloads/OS/Lab 5/6th$ gcc -o Sum Sum.c
```



```
sabhishek@s-abhishek:~/Downloads/OS/Lab 5/6th$ ./Start
ID of the Process ( Parent ) = 1976

This is Sum.c which is a Child 2!
PID of Sum.c = 1976
Please enter a number : 234
Sum of Digits: 9
Exiting Sum.c......

This is Happy_New_Year which is a Child 1!
PID of Happy_New_Year.c = 1976
Happy New Year
Exiting Happy_New_Year.c......

Parent Process Terminating.....
```



```
sabhishek@s-abhishek:~/Downloads/OS/Lab 5/6th$ ./Start
ID of the Process ( Parent ) = 1979

This is Sum.c which is a Child 2!
PID of Sum.c = 1979
Please enter a number : 566789
Sum of Digits: 41
Exiting Sum.c......

This is Happy_New_Year which is a Child 1!
PID of Happy_New_Year.c = 1979
Happy New Year
Exiting Happy_New_Year.c......

Parent Process Terminating.....
```

One Drive Link: Click Me!!

Thankyou!!