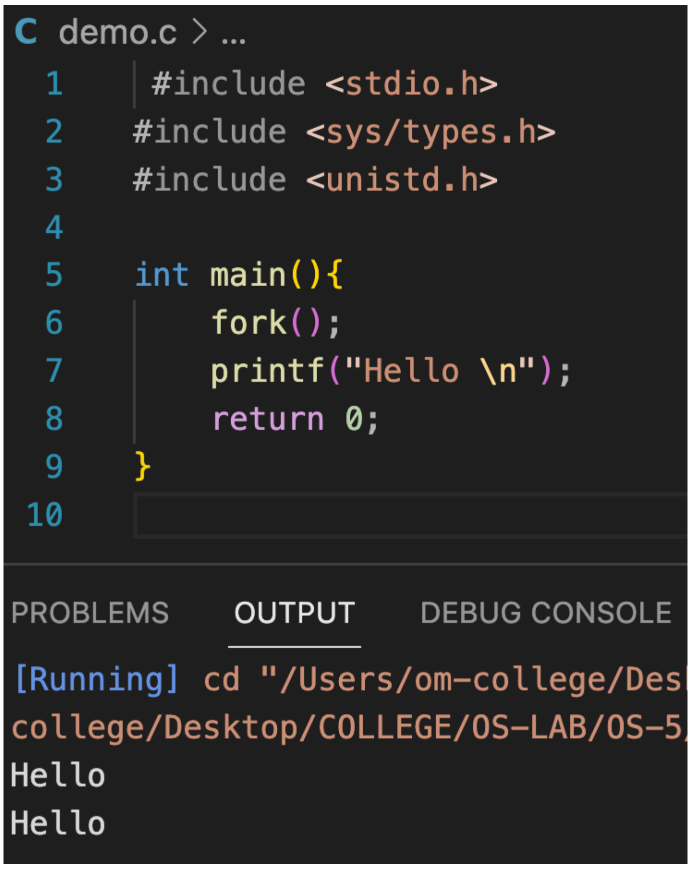
EXPERIMENT 5 : Processes

1) Write a program to create a simple. child process



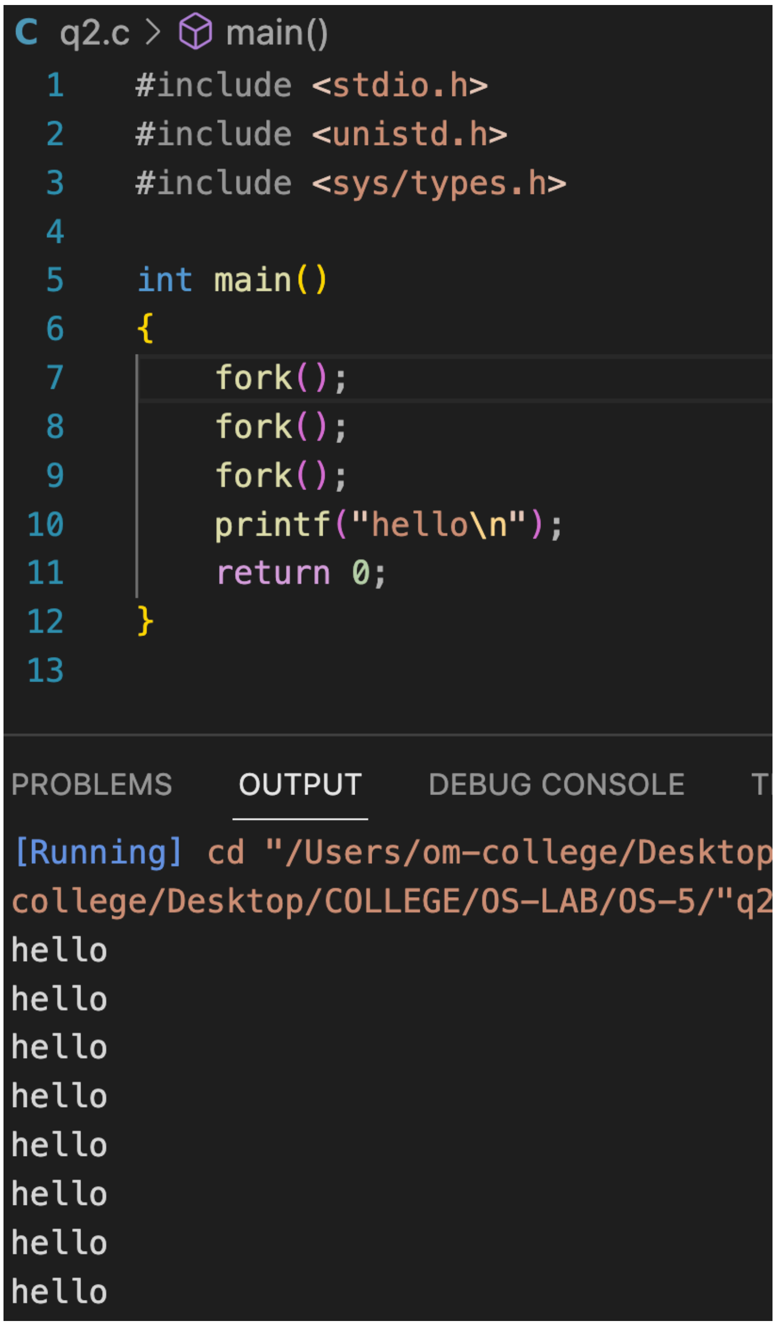
demo.c > ...

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

int main(){
    fork();
    printf("Hello \n");
    return 0;
}
```

PROBLEMS OUTPUT DEBUG CONSOLE

```
[Running] cd "/Users/om-college/Des
college/Desktop/COLLEGE/OS-LAB/OS-5/
Hello
Hello
```

2)Write a program to create multiple child. processes.



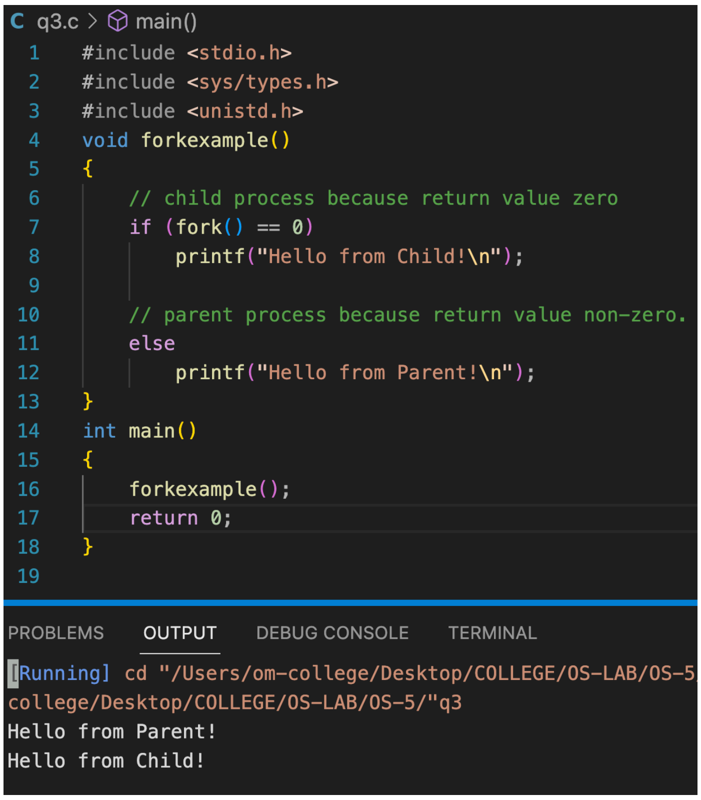
q2.c > main()

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

int main()
{
    fork();
    fork();
    fork();
    printf("hello\n");
    return 0;
}
```

PROBLEMS OUTPUT DEBUG CONSOLE T

```
[Running] cd "/Users/om-college/Desktop
college/Desktop/COLLEGE/OS-LAB/OS-5/"q2
hello
hello
hello
hello
hello
hello
hello
hello
```

3) Write a program to check the return status of parent and child processes



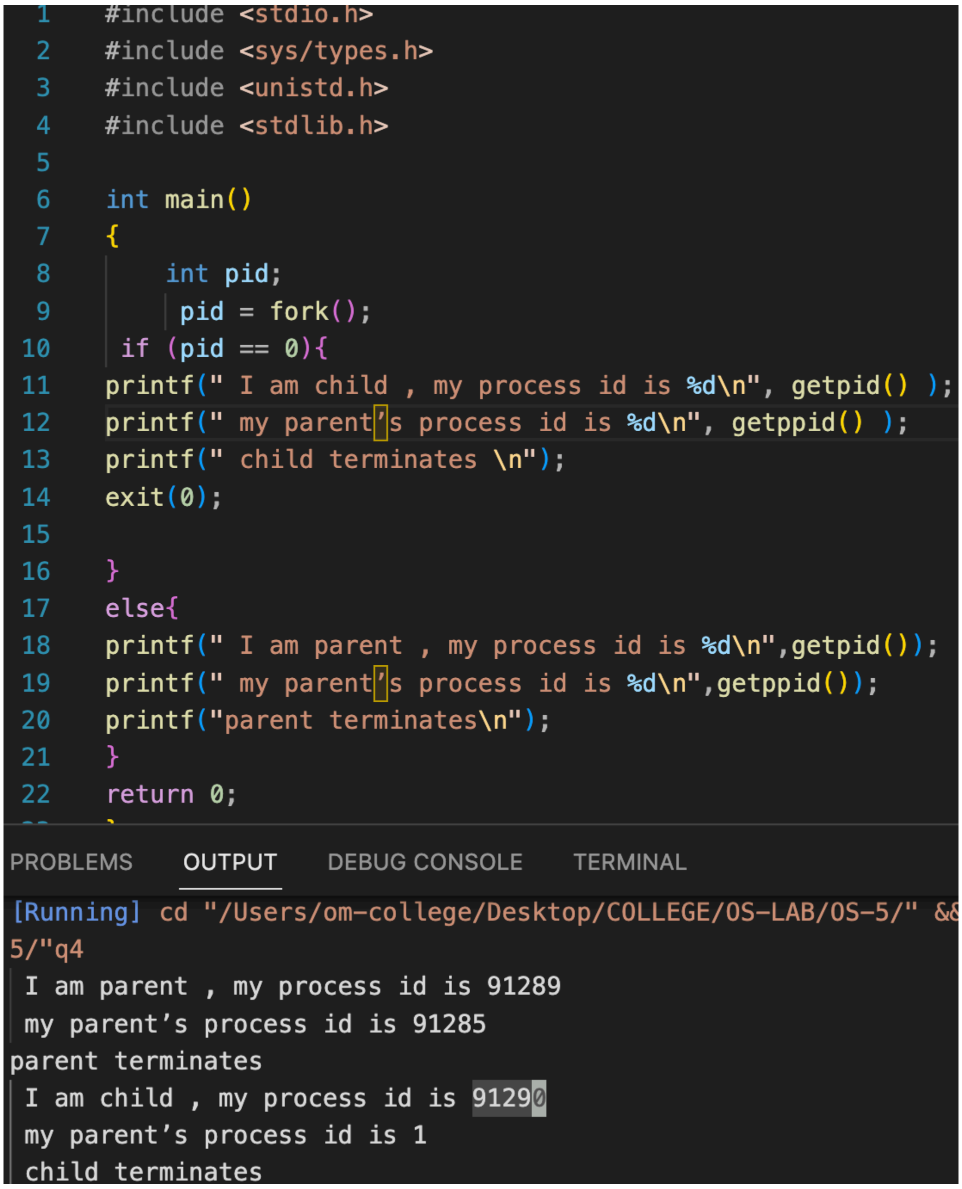
q3.c > main()

```c
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
void forkexample()
{
    // child process because return value zero
    if (fork() == 0)
        printf("Hello from Child!\n");

    // parent process because return value non-zero.
    else
        printf("Hello from Parent!\n");
}
int main()
{
    forkexample();
    return 0;
}
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
[Running] cd "/Users/om-college/Desktop/COLLEGE/OS-LAB/OS-5
college/Desktop/COLLEGE/OS-LAB/OS-5/"q3
Hello from Parent!
Hello from Child!
```

4) Write a program to get the process IDs of a child process and it’s parent process.

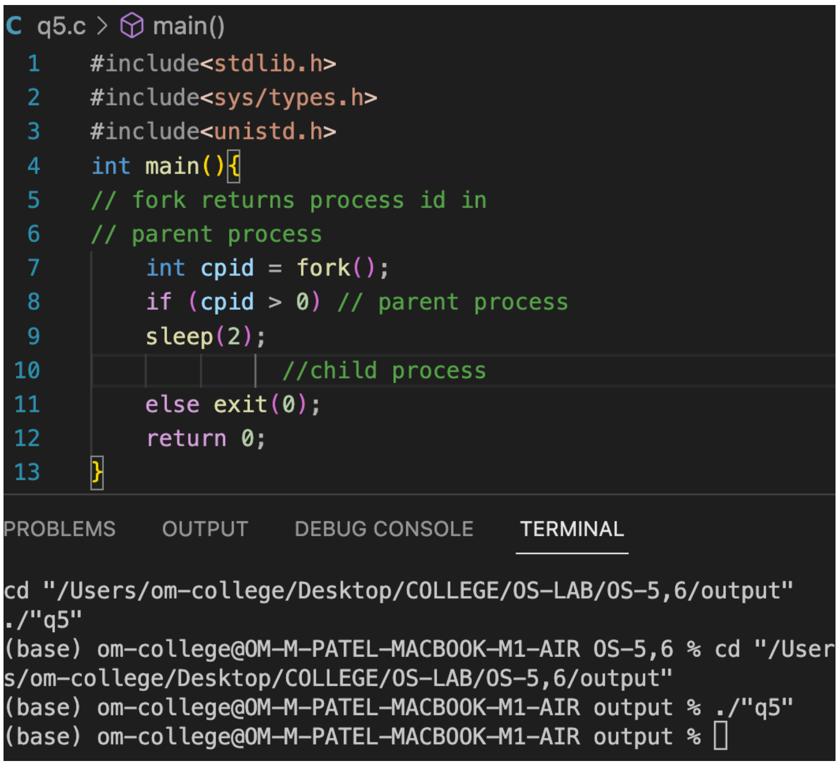


```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>

int main()
{
    int pid;
     pid = fork();
 if (pid == 0){
printf(" I am child , my process id is %d\n", getpid() );
printf(" my parent’s process id is %d\n", getppid() );
printf(" child terminates \n");
exit(0);

}
else{
printf(" I am parent , my process id is %d\n",getpid());
printf(" my parent’s process id is %d\n",getppid());
printf("parent terminates\n");
}
return 0;
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
[Running] cd "/Users/om-college/Desktop/COLLEGE/OS-LAB/OS-5/" &&
5/"q4
 I am parent , my process id is 91289
 my parent’s process id is 91285
parent terminates
 I am child , my process id is 91290
 my parent’s process id is 1
 child terminates
```

5) Write a program to create a zombie process.



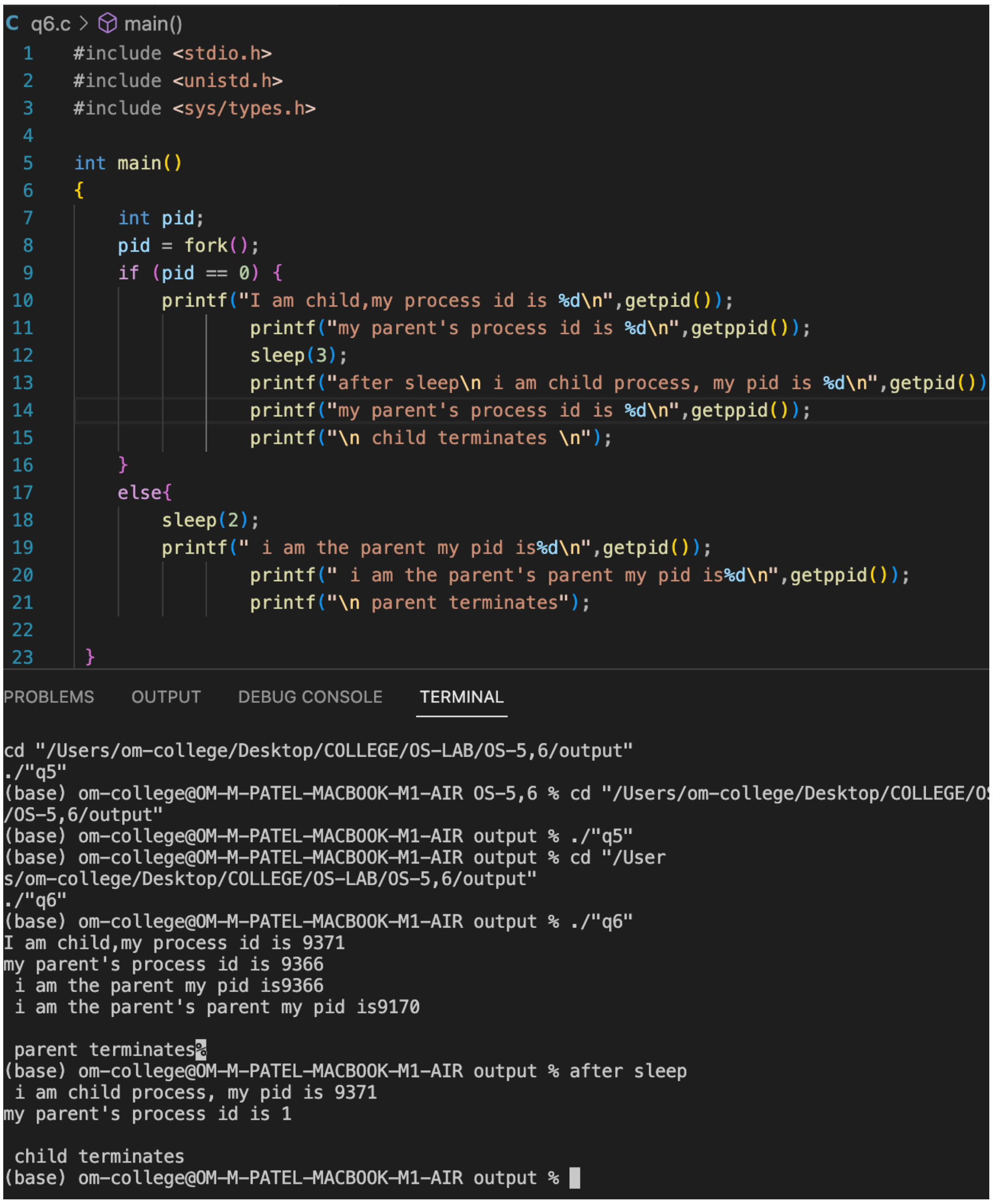
C q5.c > main()

```
#include<stdlib.h>
#include<sys/types.h>
#include<unistd.h>
int main(){
// fork returns process id in
// parent process
    int cpid = fork();
    if (cpid > 0) // parent process
    sleep(2);
            //child process
    else exit(0);
    return 0;
}
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
cd "/Users/om-college/Desktop/COLLEGE/OS-LAB/OS-5,6/output"
./"q5"
(base) om-college@OM-M-PATEL-MACBOOK-M1-AIR OS-5,6 % cd "/User
s/om-college/Desktop/COLLEGE/OS-LAB/OS-5,6/output"
(base) om-college@OM-M-PATEL-MACBOOK-M1-AIR output % ./"q5"
(base) om-college@OM-M-PATEL-MACBOOK-M1-AIR output % 
```

6) Write a program to create an orphan process.



```
q6.c > main()
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>

int main()
{
    int pid;
    pid = fork();
    if (pid == 0) {
        printf("I am child,my process id is %d\n",getpid());
                printf("my parent's process id is %d\n",getppid());
                sleep(3);
                printf("after sleep\n i am child process, my pid is %d\n",getpid())
                printf("my parent's process id is %d\n",getppid());
                printf("\n child terminates \n");
    }
    else{
        sleep(2);
        printf(" i am the parent my pid is%d\n",getpid());
                printf(" i am the parent's parent my pid is%d\n",getppid());
                printf("\n parent terminates");

}
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
cd "/Users/om-college/Desktop/COLLEGE/OS-LAB/OS-5,6/output"
./"q5"
(base) om-college@OM-M-PATEL-MACBOOK-M1-AIR OS-5,6 % cd "/Users/om-college/Desktop/COLLEGE/O
/OS-5,6/output"
(base) om-college@OM-M-PATEL-MACBOOK-M1-AIR output % ./"q5"
(base) om-college@OM-M-PATEL-MACBOOK-M1-AIR output % cd "/User
s/om-college/Desktop/COLLEGE/OS-LAB/OS-5,6/output"
./"q6"
(base) om-college@OM-M-PATEL-MACBOOK-M1-AIR output % ./"q6"
I am child,my process id is 9371
my parent's process id is 9366
 i am the parent my pid is9366
 i am the parent's parent my pid is9170

 parent terminates%
(base) om-college@OM-M-PATEL-MACBOOK-M1-AIR output % after sleep
 i am child process, my pid is 9371
my parent's process id is 1

 child terminates
(base) om-college@OM-M-PATEL-MACBOOK-M1-AIR output %
```

- **Zombie process - known as dead process , ideally when press execution completes, its entry from process table should be removed but this doesn't happen in this case**

Int main(){
// fork returns process id in parent process
Int child_pid = fork();
//parent process
If (child_pid>0)
sleep(60)
// child process
Else

```
exit(0);   // here parent is unaware about child process end , so it hasn't
erased entry.
return(0);
}
```

- Here child process completes its executions by using exit() system call.
- Generally when child process ends, systems asks parent to erase its entry from cpu,but here as parent in sleep this case doesn't occurs.

- **orphan process - a process which is executing (live) , but its parent process has terminted(dead)**

In real world as orphans are adopted by guardians who look after them. Similarly in linux orphan process is adopted by new process which is mostly init(pid =1). This is called re parenting.
Reparenting is done by kernel , when kernel detects an orphan process in os land assigns a new parent process.

```
Int main{

}
```

A daemon process is a. Background process that is not under the direct control
Of the user. , this proves usually start when the system is bootstrapped and it terminated with system, shut down.
Usually parent process of daemon process is init process. Its because init process usually adopts daemon after parent process forks the daemon process a=nd terminates.

Generally all daemon process end int - "d"
Eg:
- crowd
- syslogd
- httpd
- dhcpd