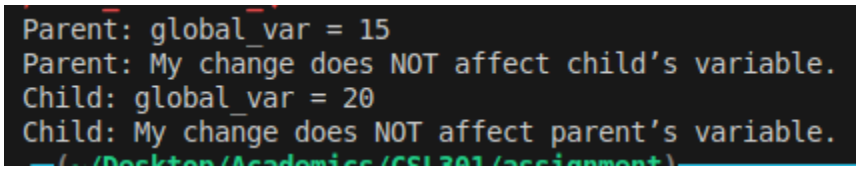
Question 1

File Name: `HA1_12340220_Q1.c`

Code:

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

int global_var = 10;

int main() {
   pid_t pid = fork();

   if (pid < 0) {
       perror("fork failed");
       return 1;
   }
   else if (pid == 0) {
       global_var += 10;
       printf("Child Process: global_var = %d\n", global_var);
       printf("Child: My changes do not affect the parent.\n");
   }
   else {
       global_var += 5;
       printf("Parent Process: global_var = %d\n", global_var);
       printf("Parent: My changes do not affect the child.\n");
   }
   return 0;
}
```

Output Screenshot:



Explanation:

When `fork()` is called, the entire process memory is **copied** to the child process.

- The global variable `global_var` starts at 10 in both processes.

- The **child** modifies its copy (+10), the **parent** modifies its copy (+5).
- Changes are not shared due to **copy-on-write** memory behavior.

Question 2

File Name: `HA1_12340220_Q2.c`

Code:

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

int main() {
   pid_t pid1 = fork();
   pid_t pid2 = fork();

   printf("PID: %d, Parent PID: %d\n", getpid(), getppid());

   return 0;
}
```

Output Screenshot:

```
PID: 23454, Parent PID: 23144
PID: 23455, Parent PID: 3000
PID: 23456, Parent PID: 3000
PID: 23457, Parent PID: 3000
```

Explanation:

Two calls to `fork()` create **4 processes** in total:

- 1st `fork()` → 2 processes.
- 2nd `fork()` → each of those 2 processes creates a new one → total = 4.
  The output shows each process printing its PID and its parent's PID, forming a **process tree**.

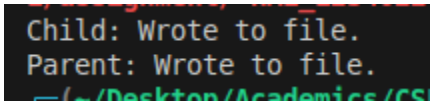
Question 3

File Name: `HA1_12340220_Q3.c`

Code:

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

int main() {
    int fd = open("output.txt", O_WRONLY | O_CREAT | O_TRUNC, 0644);
    if (fd < 0) {
        perror("open");
        return 1;
    }

    pid_t pid = fork();

    if (pid == 0) {
        write(fd, "12340220 - Child wrote this.\n", 29);
        printf("Child: Wrote to file.\n");
    }
    else {
        write(fd, "Amay Dixit - Parent wrote this.\n", 32);
        printf("Parent: Wrote to file.\n");
    }

    close(fd);
    return 0;
}
```

Output Screenshot:



Explanation:

After `fork()`, the **file descriptor** is inherited by the child with the same file offset.

- Both processes write to the file independently, but the writes may appear in mixed order depending on scheduling.
- This happens because they share the same underlying **open file table entry**.

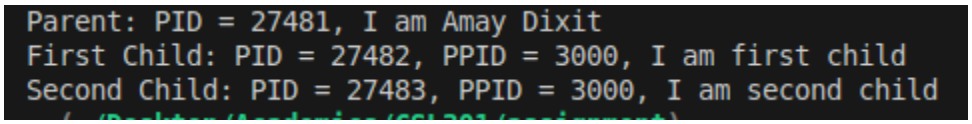
Question 4

File Name: HA1_12340220_Q4.c

Code:

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

int main() {
    pid_t pid1, pid2;

    pid1 = fork();
    if (pid1 == 0) {
        printf("First Child: PID = %d, PPID = %d, I am first child\n",
getpid(), getppid());
        return 0;
    }

    pid2 = fork();
    if (pid2 == 0) {
        printf("Second Child: PID = %d, PPID = %d, I am second child\n",
getpid(), getppid());
        return 0;
    }

    printf("Parent: PID = %d, I am Amay Dixit\n", getpid());
    return 0;
}
```

Output Screenshot:



Explanation:

- The parent process first forks **child 1**, then forks **child 2**.
- All processes run independently and print their details.
- The children's PPID matches the parent's PID, confirming the hierarchy.
- This results in **3 total processes**.