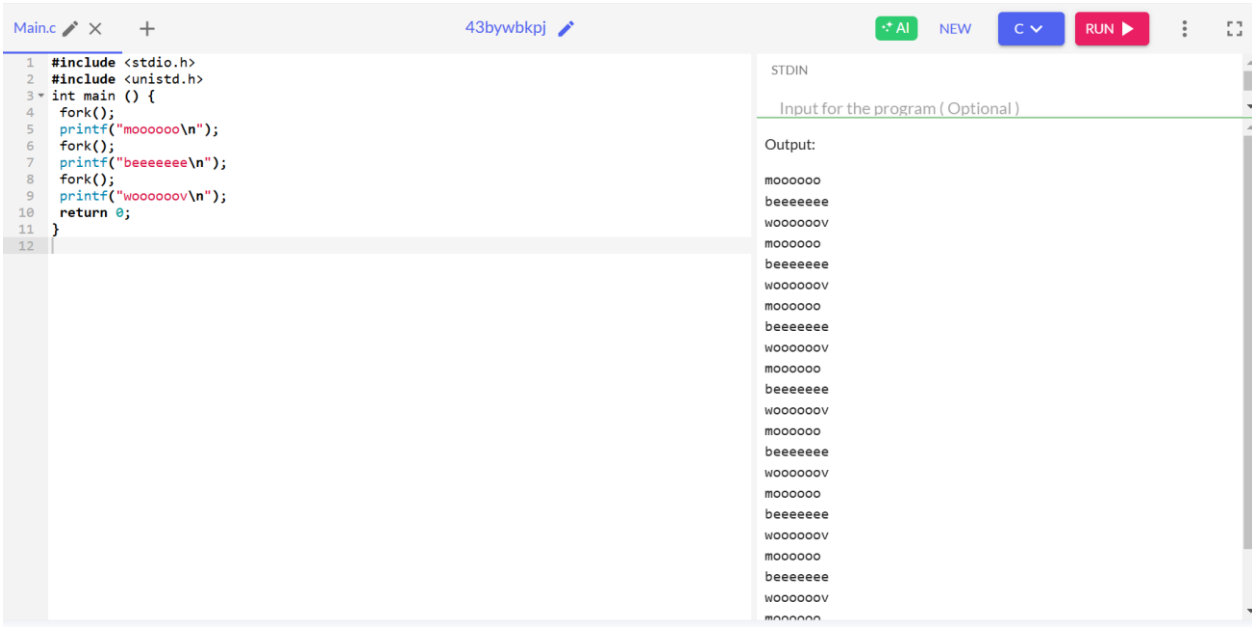
Practice #4

1.

2.



3.

```c
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
int main()
{
pid_t pid, pid1;
/* fork a child process */
pid = fork();
if (pid < 0) { /* error occurred */
fprintf(stderr, "Fork Failed");
return 1;
}
else if (pid == 0) { /* child process */
pid1 = getpid();
printf("child: pid = %d",pid); /* A */
printf("child: pid1 = %d",pid1); /* B */
}
else { /* parent process */
pid1 = getpid();
printf("parent: pid = %d",pid); /* C */
printf("parent: pid1 = %d",pid1); /* D */
wait(NULL);
}
return 0;
}
```

```
STDIN
Input for the program ( Optional )
Output:
child: pid = 0child: pid1 = 6413parent: pid = 6413parent: pid1
```

4.

```c
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
int value = 5;
int main()
{
pid_t pid;
pid = fork();
if (pid == 0) { /* child process */
value += 15;
return 0;
}
else if (pid > 0) { /* parent process */
wait(NULL);
printf("PARENT: value = %d",value); /* LINE A */
return 0;
}
}
```

```
STDIN
Input for the program ( Optional )
Output:
PARENT: value = 5
```

5.

```c
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
#define SIZE 5
int nums[SIZE] = {0,1,2,3,4};
int main()
{ int i;
pid t pid;
pid = fork();
if (pid == 0) {
for (i = 0; i < SIZE; i++) { nums[i] *= -i;
printf("CHILD: %d ",nums[i]); /* LINE X */ }}
else if (pid > 0) { wait(NULL);
for (i = 0; i < SIZE; i++)
printf("PARENT: %d ",nums[i]); /* LINE Y */
}
return 0;
}
```

```
STDIN
Input for the program ( Optional )
Output:
Main.c: In function ‘main’:
Main.c:8:1: error: unknown type name ‘pid’; did you mean ‘pid_
    8 | pid t pid;
      | ^~~
      | pid_t
Main.c:8:7: error: expected ‘=’, ‘,’, ‘;’, ‘asm’ or ‘__attribu
    8 | pid t pid;
      |       ^~~
Main.c:9:1: error: ‘pid’ undeclared (first use in this functic
    9 | pid = fork();
      | ^~~
Main.c:9:1: note: each undeclared identifier is reported only
Main.c:13:21: warning: implicit declaration of function ‘wait’
   13 | else if (pid > 0) { wait(NULL);
      |                     ^~~~
```


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

#define SIZE 5
int nums[SIZE] = {0, 1, 2, 3, 4};

int main() {
    int i;
    pid_t pid;

    pid = fork();
    if (pid == 0) {
        for (i = 0; i < SIZE; i++) {
            nums[i] *= -i;
            printf("CHILD: %d ", nums[i]);  // LINE X
        }
    } else if (pid > 0) {
        wait(NULL);
        for (i = 0; i < SIZE; i++) {
            printf("PARENT: %d ", nums[i]);  // LINE Y
        }
    }
    return 0;
}
```

Explanation of the code

This code demonstrates creating a child process using fork() and working with a shared array nums.

Code breakdown:

1. **Including libraries**
    - sys/types.h and unistd.h — for system calls (fork()).
    - sys/wait.h — for using wait(), so the parent process waits for the child.
    - stdio.h — for input-output functions (printf()).
2. **Declaring variables**
    - nums[SIZE] = {0, 1, 2, 3, 4}; — an array used by both the parent and child process.
    - pid_t pid; — a variable to store the process ID.
3. **Creating a new process**
    - pid = fork();
        - If pid == 0, it means the child process is running.
        - If pid > 0, it means the parent process is running.
4. **Processing in the child process**
    - Loops through nums, multiplies each element by -i.
    - Prints values to the console (CHILD: ...).
5. **Processing in the parent process**
    - The parent waits for the child process to finish (wait(NULL);).
    - After the child process finishes, it prints the original values (PARENT: ...).

Changes that were made:

1. **Fixed the pid declaration error**
    - Before: pid t pid;
    - Now: pid_t pid;
2. **Added #include <sys/wait.h>**
    - Needed for wait().