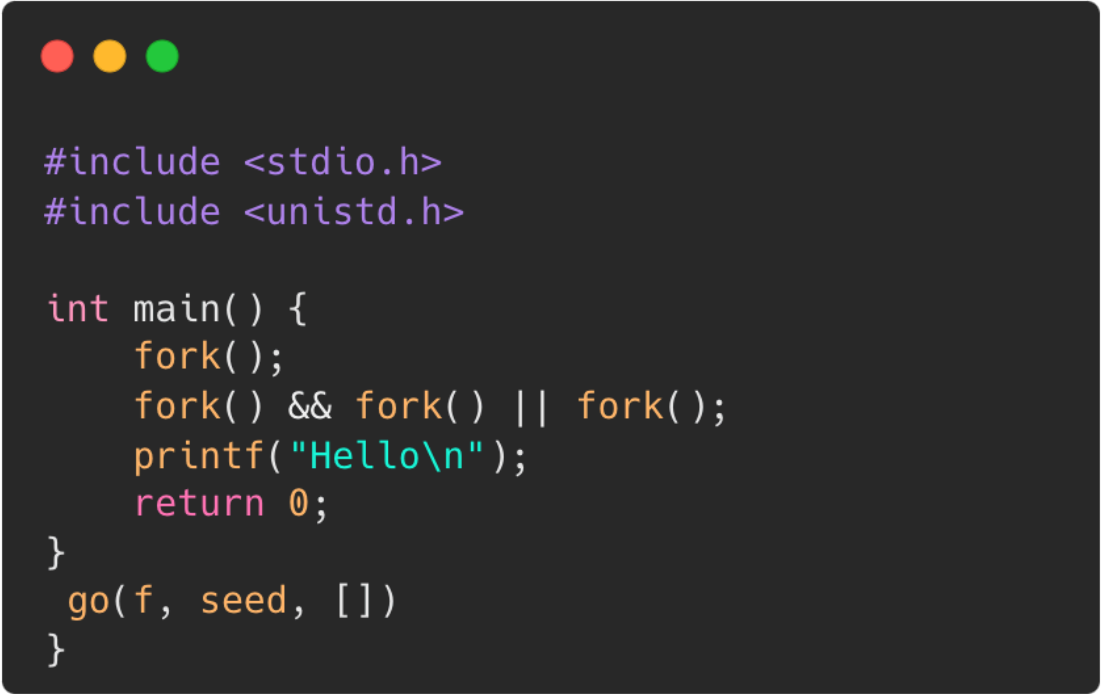


**UNIVERSITY OF NORTH CAROLINA AT GREENSBORO**  
DEPARTMENT OF COMPUTER SCIENCE  
**CSC362: System Programming**  
Assignment 2

---

Question 1: Given the following C code snippet:

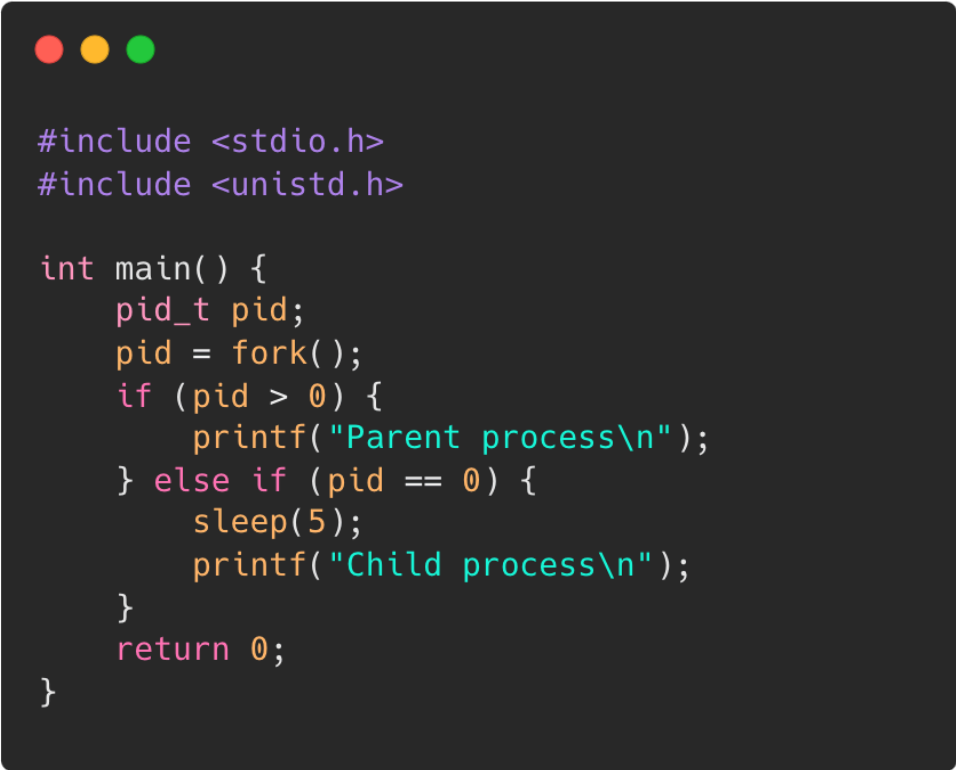


```
#include <stdio.h>
#include <unistd.h>

int main() {
    fork();
    fork() && fork() || fork();
    printf("Hello\n");
    return 0;
}
go(f, seed, [])
}
```

Analyze the code and explain the output produced by the code when executed. Identify any potential issues or bugs in the code and propose a corrected version if necessary.

Question 2: Consider the following C code snippet:

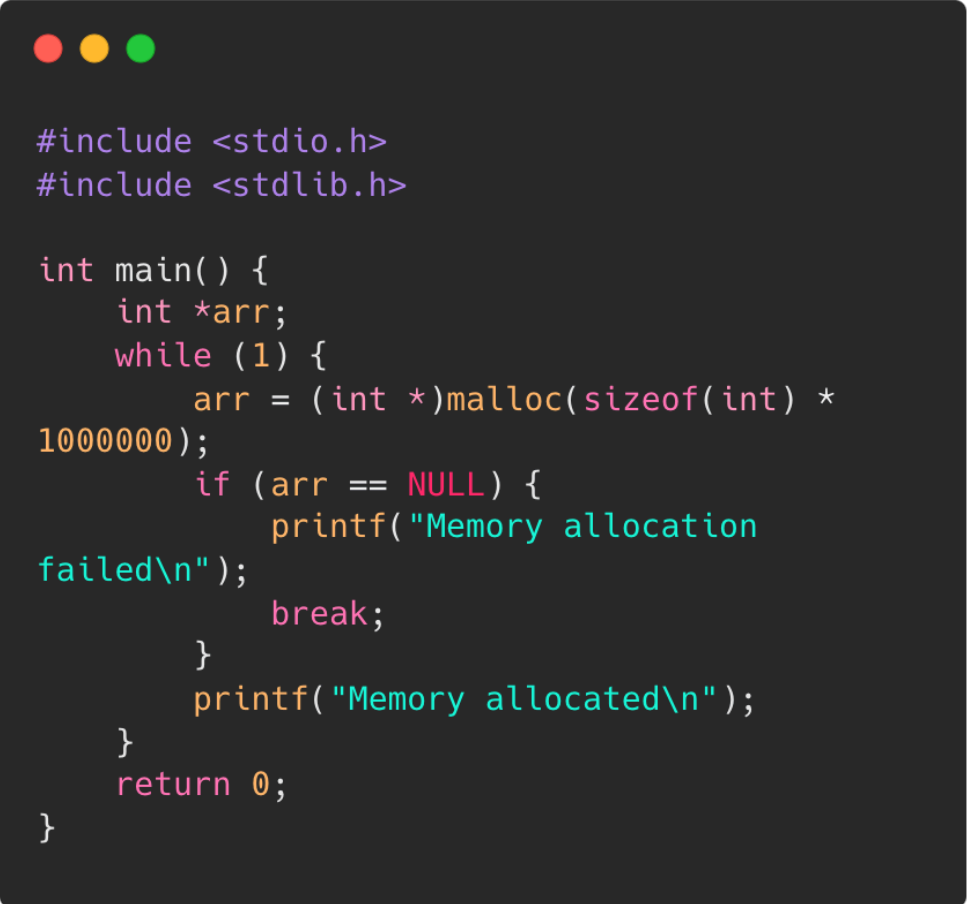


```
#include <stdio.h>
#include <unistd.h>

int main() {
    pid_t pid;
    pid = fork();
    if (pid > 0) {
        printf("Parent process\n");
    } else if (pid == 0) {
        sleep(5);
        printf("Child process\n");
    }
    return 0;
}
```

Analyze the code and explain the concept of process creation and termination, as demonstrated in the code. Discuss the differences between the parent and child processes in terms of their execution flow and output.

Question 3: Consider a scenario where a system has limited memory available for process creation. Analyze the following C code snippet:



```
#include <stdio.h>
#include <stdlib.h>

int main() {
    int *arr;
    while (1) {
        arr = (int *)malloc(sizeof(int) *
1000000);
        if (arr == NULL) {
            printf("Memory allocation
failed\n");
            break;
        }
        printf("Memory allocated\n");
    }
    return 0;
}
```

Analyze the code and propose a strategy for managing memory allocation to prevent issues like memory exhaustion or fragmentation. Discuss the potential implications of the code on system resources and management.

Question 4: Consider the following C program that demonstrates the use of the **wait()** system call to handle the termination of child processes:

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

int main() {
    pid_t pid;
    int status;
    for (int i = 0; i < 3; ++i) {
        pid = fork();
        if (pid == 0) {
            printf("Child process %d\n", getpid());
            exit(0);
        }
    }
    while ((pid = wait(&status)) > 0) {
        printf("Child process %d terminated\n", pid);
    }
    printf("Parent process\n");
    return 0;
}
```

Analyze the code and explain how the `wait()` system call facilitates process synchronization. Describe the behavior of the program in terms of creating and terminating child processes.