# Operating System
# COM301P

## Programming
## Assignment
## Lab - 3

By :

Sai Kaushik S

CED18I044

**Question:** Test drive a C program that creates Orphan and Zombie Processes

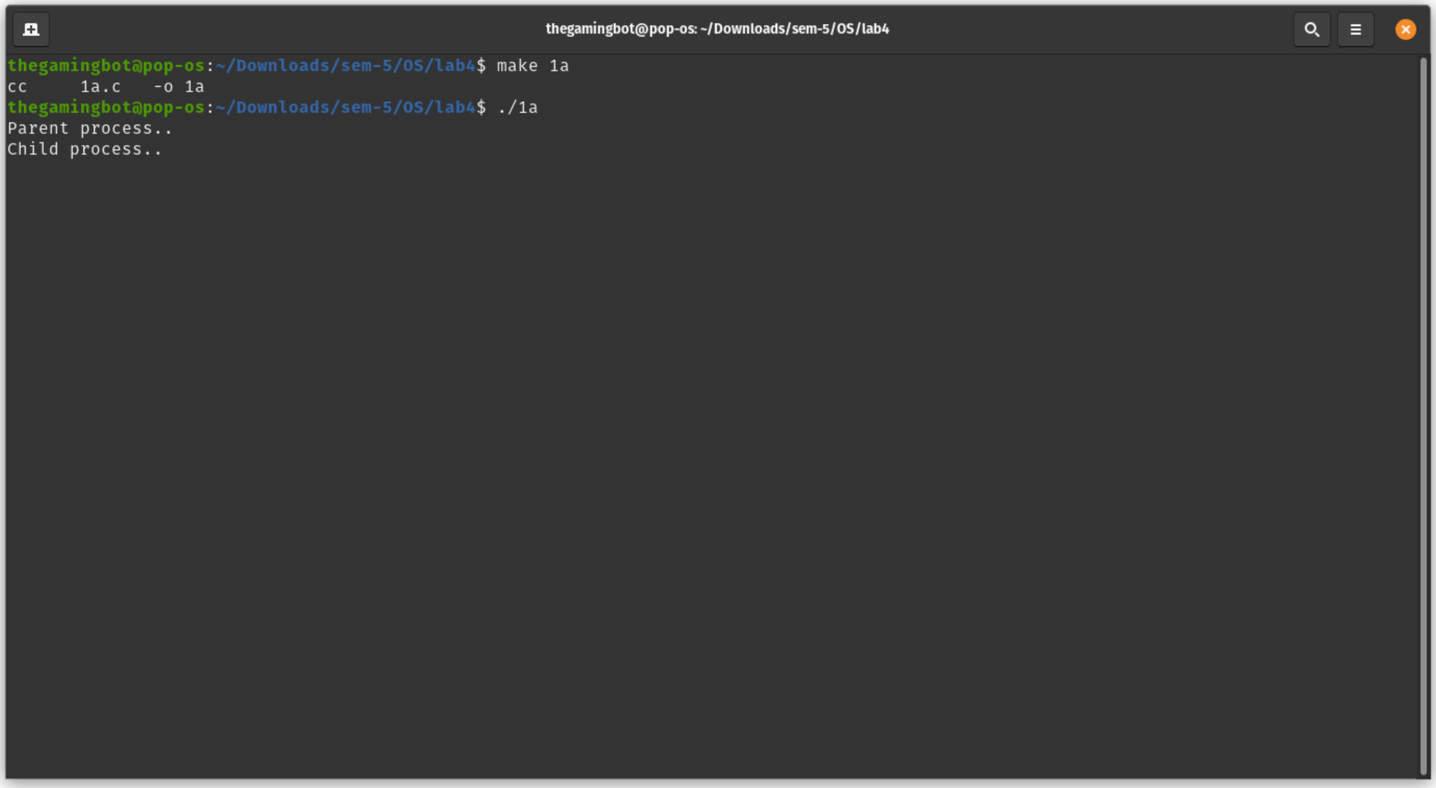
**C code:**

```c
//          __  __                               _             __          __
//         / /_/ /_  ___  ____ _____ _____ ___  (_)___  ____ _/ /_  ____  / /_
//        / __/ __ \/ _ \/ __ `/ __ `/ __ `__ \/ / __ \/ __ `/ __ \/ __ \/ __/
//       / /_/ / / /  __/ /_/ / /_/ / / / / / / / / / / /_/ / /_/ / /_/ / /_
//       \__/_/ /_/\___/\__, /\__,_/_/ /_/ /_/_/_/ /_/\__, /_.___/\____/\__/
//                     /____/                        /____/
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>

int main(){
   pid_t pid = fork();
   if (pid < 0) {
       printf("Fork failed.\n");
       exit(1);
   }
   else if (pid > 0){
       printf("Parent process..\n");
       sleep(50);
   }
   else{
       printf("Child process..\n");
       exit(0);
   }
   return 0;
}
```

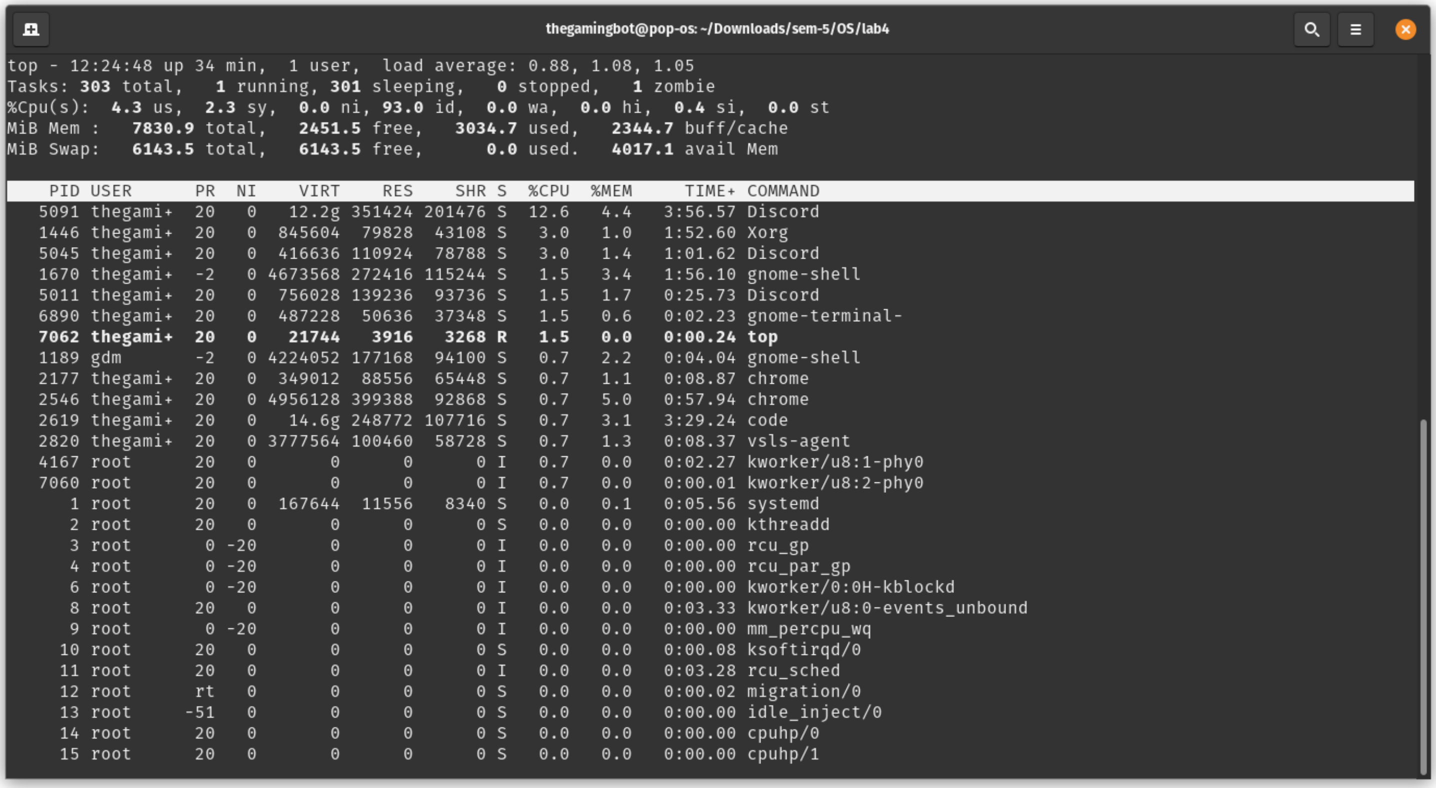
**Explanation:**

The parent process reads the exit status of the child process which reaps off the child process entry from the process table, making it a zombie.

# Output:

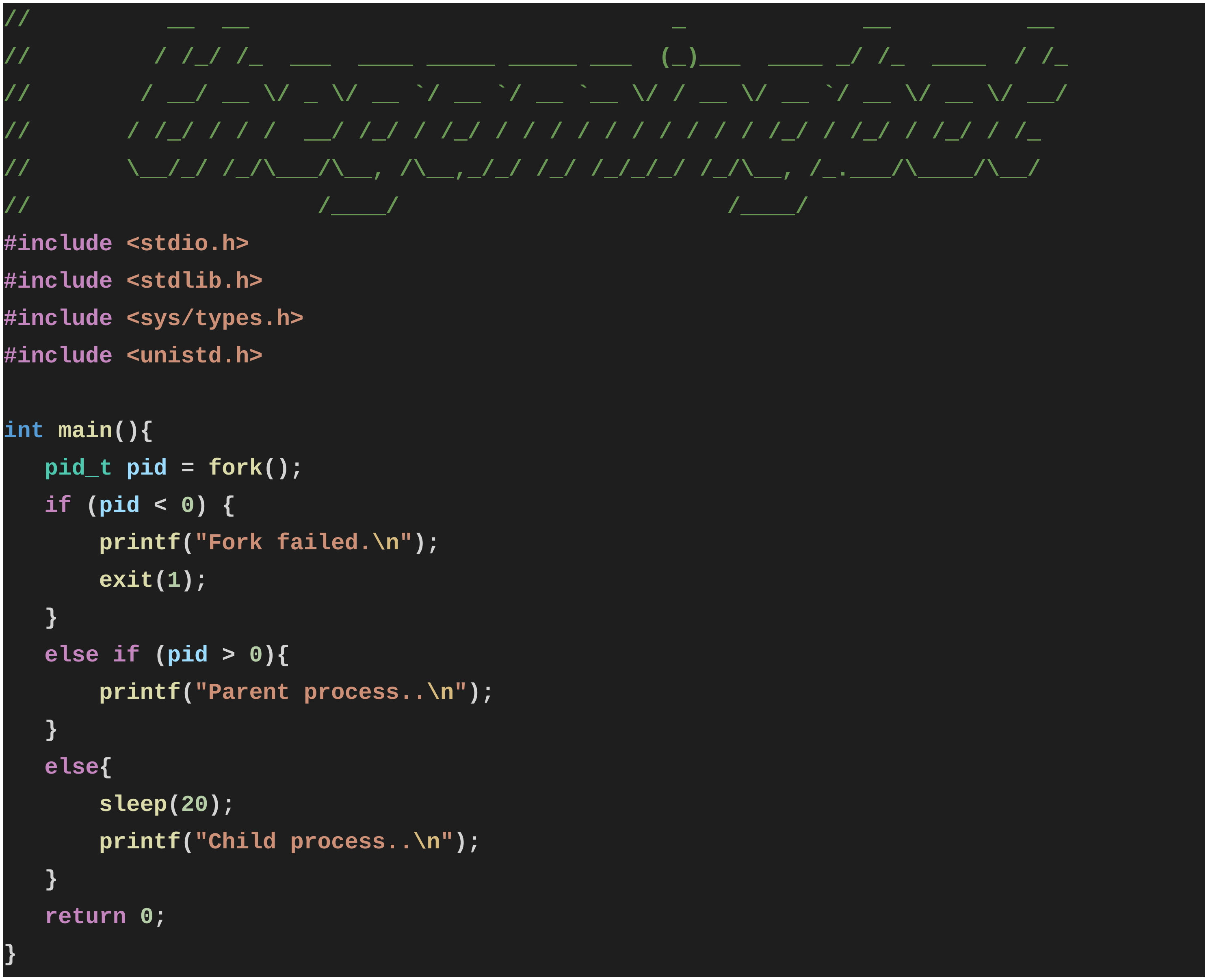


```
thegamingbot@pop-os:~/Downloads/sem-5/OS/lab4$ make 1a
cc     1a.c   -o 1a
thegamingbot@pop-os:~/Downloads/sem-5/OS/lab4$ ./1a
Parent process..
Child process..
```



```
top - 12:24:48 up 34 min,  1 user,  load average: 0.88, 1.08, 1.05
Tasks: 303 total,   1 running, 301 sleeping,   0 stopped,   1 zombie
%Cpu(s):  4.3 us,  2.3 sy,  0.0 ni, 93.0 id,  0.0 wa,  0.0 hi,  0.4 si,  0.0 st
MiB Mem :   7830.9 total,   2451.5 free,   3034.7 used,   2344.7 buff/cache
MiB Swap:   6143.5 total,   6143.5 free,      0.0 used.   4017.1 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
   5091 thegami+  20   0   12.2g 351424 201476 S  12.6   4.4   3:56.57 Discord
   1446 thegami+  20   0  845604  79828  43108 S   3.0   1.0   1:52.60 Xorg
   5045 thegami+  20   0  416636 110924  78788 S   3.0   1.4   1:01.62 Discord
   1670 thegami+  -2   0 4673568 272416 115244 S   1.5   3.4   1:56.10 gnome-shell
   5011 thegami+  20   0  756028 139236  93736 S   1.5   1.7   0:25.73 Discord
   6890 thegami+  20   0  487228  50636  37348 S   1.5   0.6   0:02.23 gnome-terminal-
   7062 thegami+  20   0   21744   3916   3268 R   1.5   0.0   0:00.24 top
   1189 gdm       -2   0 4224052 177168  94100 S   0.7   2.2   0:04.04 gnome-shell
   2177 thegami+  20   0  349012  88556  65448 S   0.7   1.1   0:08.87 chrome
   2546 thegami+  20   0 4956128 399388  92868 S   0.7   5.0   0:57.94 chrome
   2619 thegami+  20   0   14.6g 248772 107716 S   0.7   3.1   3:29.24 code
   2820 thegami+  20   0 3777564 100460  58728 S   0.7   1.3   0:08.37 vsls-agent
   4167 root      20   0       0      0      0 I   0.7   0.0   0:02.27 kworker/u8:1-phy0
   7060 root      20   0       0      0      0 I   0.7   0.0   0:00.01 kworker/u8:2-phy0
      1 root      20   0  167644  11556   8340 S   0.0   0.1   0:05.56 systemd
      2 root      20   0       0      0      0 S   0.0   0.0   0:00.00 kthreadd
      3 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 rcu_gp
      4 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 rcu_par_gp
      6 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 kworker/0:0H-kblockd
      8 root      20   0       0      0      0 I   0.0   0.0   0:03.33 kworker/u8:0-events_unbound
      9 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 mm_percpu_wq
     10 root      20   0       0      0      0 S   0.0   0.0   0:00.08 ksoftirqd/0
     11 root      20   0       0      0      0 I   0.0   0.0   0:03.28 rcu_sched
     12 root      rt   0       0      0      0 S   0.0   0.0   0:00.02 migration/0
     13 root     -51   0       0      0      0 S   0.0   0.0   0:00.00 idle_inject/0
     14 root      20   0       0      0      0 S   0.0   0.0   0:00.00 cpuhp/0
     15 root      20   0       0      0      0 S   0.0   0.0   0:00.00 cpuhp/1
```

## C code:

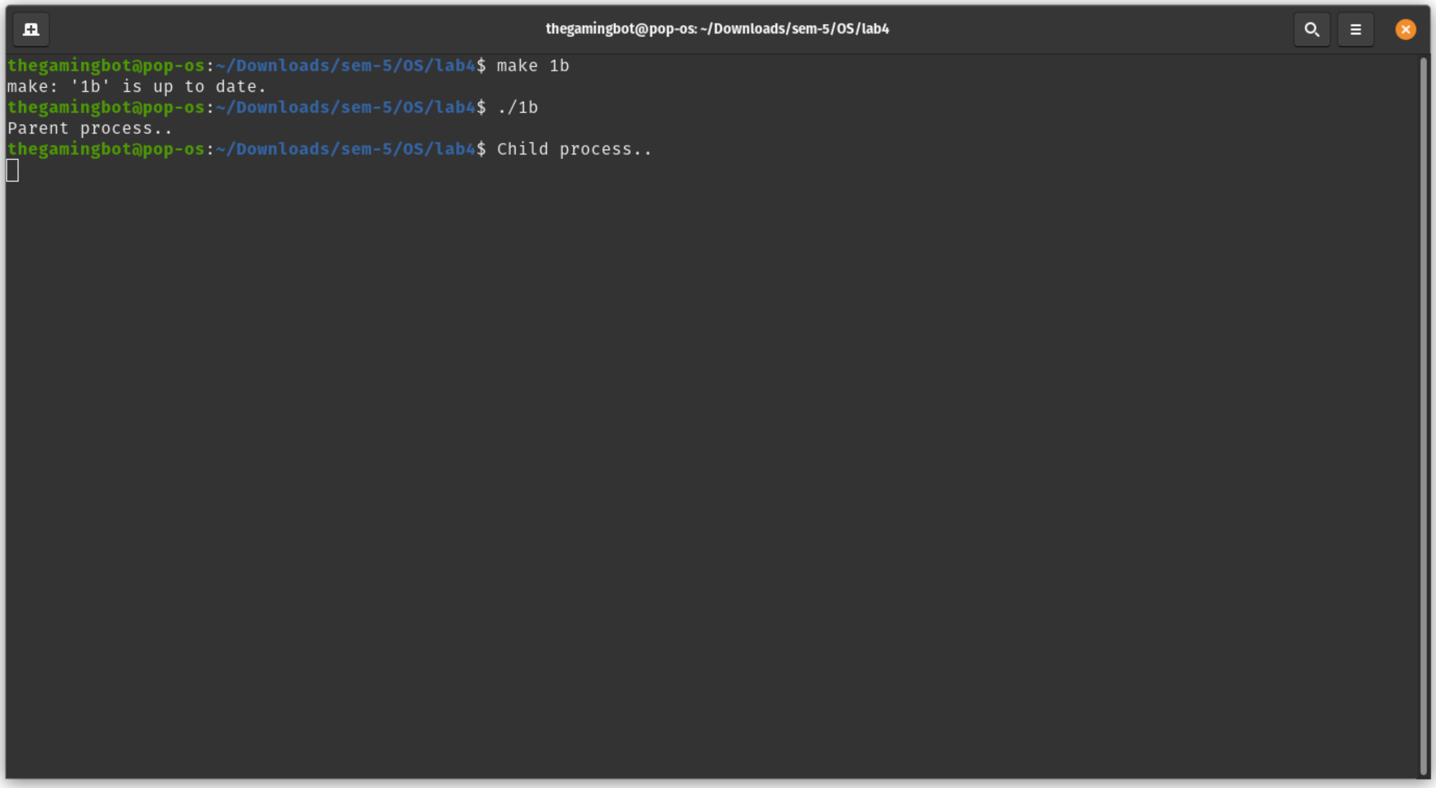


```c
//          __  __                               _             __          __
//         / /_/ /_  ___  ____ _____ _____ ___  (_)___  ____ _/ /_  ____  / /_
//        / __/ __ \/ _ \/ __ `/ __ `/ __ `__ \/ / __ \/ __ `/ __ \/ __ \/ __/
//       / /_/ / / /  __/ /_/ / /_/ / / / / / / / / / / /_/ / /_/ / /_/ / /_
//       \__/_/ /_/\___/\__, /\__,_/_/ /_/ /_/_/_/ /_/\__, /_.___/\____/\__/
//                     /____/                        /____/
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>

int main(){
   pid_t pid = fork();
   if (pid < 0) {
       printf("Fork failed.\n");
       exit(1);
   }
   else if (pid > 0){
       printf("Parent process..\n");
   }
   else{
       sleep(20);
       printf("Child process..\n");
   }
   return 0;
}
```

## Explanation:

The parent process exits before the child, leaving the child without any parent, i.e. orphan.

## Output:



```
thegamingbot@pop-os:~/Downloads/sem-5/OS/lab4$ make 1b
make: '1b' is up to date.
thegamingbot@pop-os:~/Downloads/sem-5/OS/lab4$ ./1b
Parent process..
thegamingbot@pop-os:~/Downloads/sem-5/OS/lab4$ Child process..
```

**Question:** Develop a multiprocessing version of Merge or Quick Sort. Extra credits would be given for those who implement both in a multiprocessing fashion [increased no of processes to enhance the effect of parallelization]

**C code:**

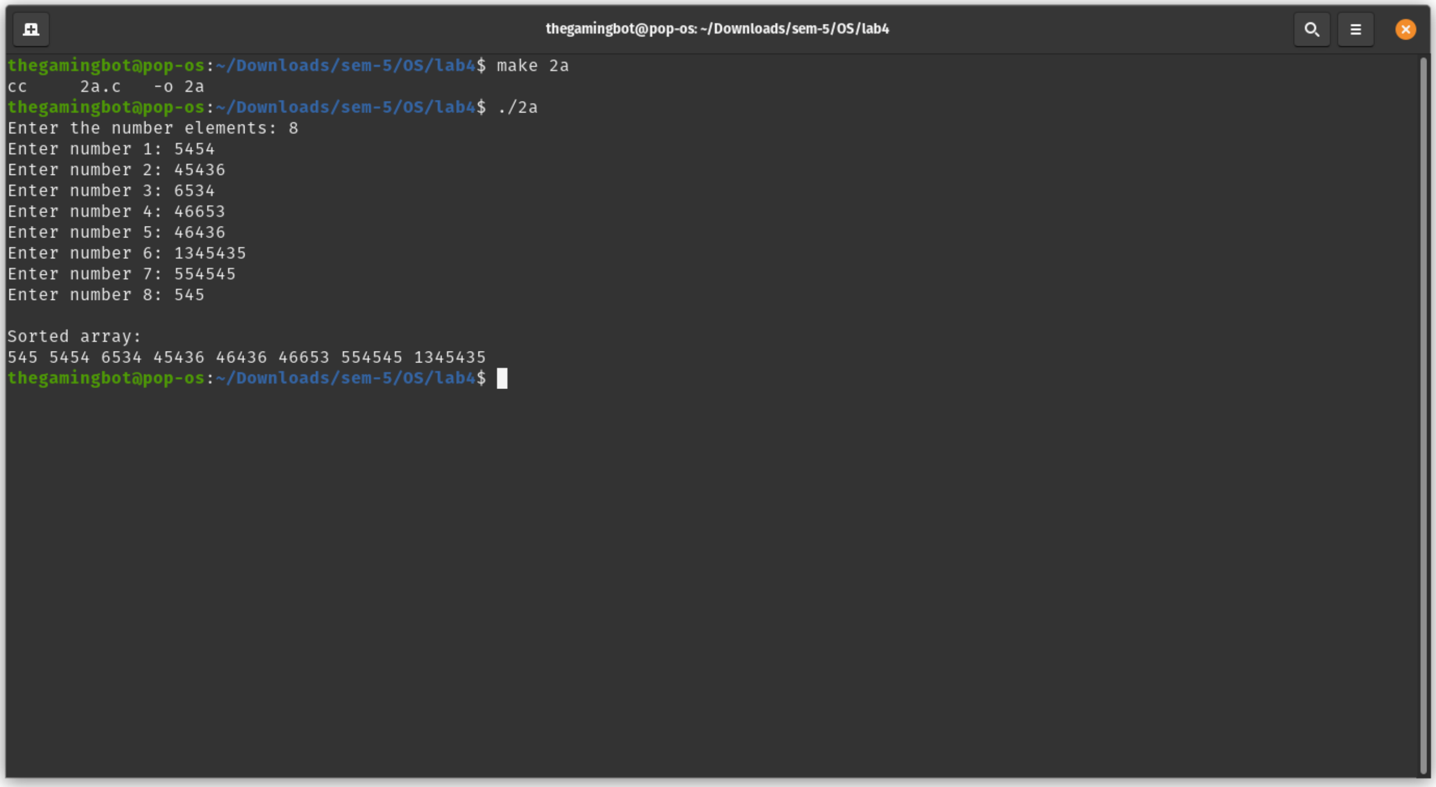
```c
//          __  __                               _             __          __
//         / /_/ /_  ___  ____ _____ _____ ___  (_)___  ____ _/ /_  ____  / /_
//        / __/ __ \/ _ \/ __ `/ __ `/ __ `__ \/ / __ \/ __ `/ __ \/ __ \/ __/
//       / /_/ / / /  __/ /_/ / /_/ / / / / / / / / / / /_/ / /_/ / /_/ / /_
//       \__/_/ /_/\___/\__, /\__,_/_/ /_/ /_/_/_/ /_/\__, /_.___/\____/\__/
//                     /____/                        /____/
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

void merge(int* arr, int l, int m, int r);
void mergeSort(int* arr, int l, int r);
void print(int* arr, int n);

int main(){
   int n;
   printf("Enter the number elements: ");
   fflush(stdin);
   scanf("%d", &n);
   int arr[n];
   for(int i = 0; i < n; i++){
       printf("Enter number %d: ", i + 1);
       fflush(stdin);
       scanf("%d", &arr[i]);
   }

   mergeSort(arr, 0, n - 1);

   printf("\nSorted array: \n");
   print(arr, n);
   return 0;
}

void merge(int* arr, int l, int m, int r){
   int i, j, k;
   int n1 = m - l + 1;
   int n2 = r - m;

   int L[n1], R[n2];

   for (i = 0;i < n1;i++)
       L[i] = arr[l + i];
   for (j = 0;j < n2;j++)
       R[j] = arr[m + 1 + j];

   i = 0;
   j = 0;
   k = l;
   while (i < n1 && j < n2) {
       if (L[i] <= R[j]) {
           arr[k] = L[i];
           i++;
       }
       else {
           arr[k] = R[j];
           j++;
       }
```

```
       k++;
   }

   while (i < n1) {
       arr[k] = L[i];
       i++;
       k++;
   }

   while (j < n2) {
       arr[k] = R[j];
       j++;
       k++;
   }
}

void mergeSort(int* arr, int l, int r){
   if (l < r) {
       int m = l + (r - l) / 2;
       pid_t pid = vfork();

       if(pid < 0){
           printf("Fork failed..\n");
           exit(1);
       }
       else if (pid == 0){
           mergeSort(arr, l, m);
           exit(0);
       }
       else{
           mergeSort(arr, m + 1, r);
           wait(NULL);
       }
       merge(arr, l, m, r);
   }
}

void print(int* arr, int n){
   for (int i = 0;i < n;i++)
       printf("%d ", arr[i]);
   printf("\n");
}
```

## Explanation:

The right half of the array is sorted by the parent, whereas the left by the child.

## Output:



```
thegamingbot@pop-os:~/Downloads/sem-5/OS/lab4$ make 2a
cc     2a.c   -o 2a
thegamingbot@pop-os:~/Downloads/sem-5/OS/lab4$ ./2a
Enter the number elements: 8
Enter number 1: 5454
Enter number 2: 45436
Enter number 3: 6534
Enter number 4: 46653
Enter number 5: 46436
Enter number 6: 1345435
Enter number 7: 554545
Enter number 8: 545

Sorted array:
545 5454 6534 45436 46436 46653 554545 1345435
thegamingbot@pop-os:~/Downloads/sem-5/OS/lab4$ 
```

**Question:** Develop a C program to count the maximum number of processes that can be created using fork call.

## C code:

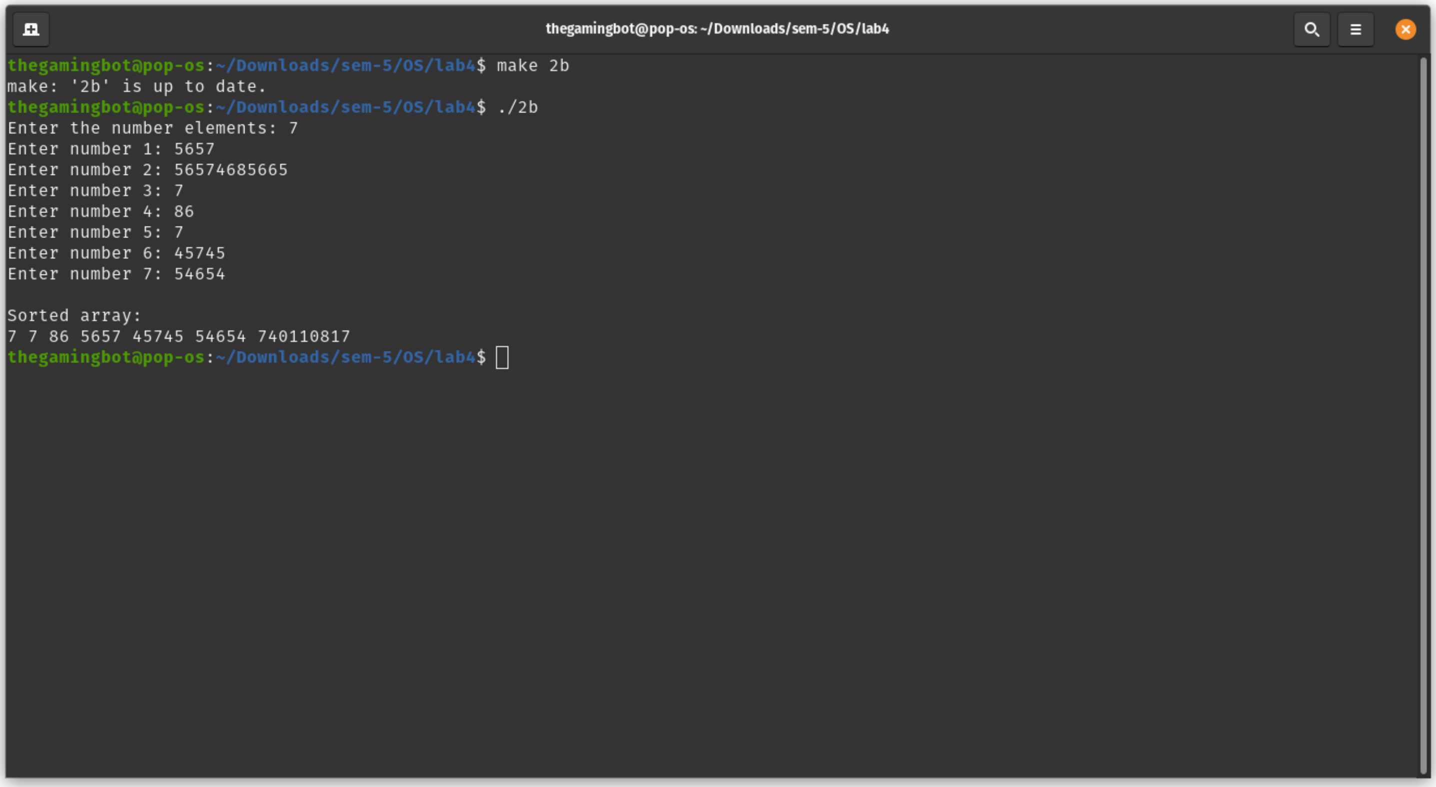
```c
//          __  __                               _              __          __
//         / /_/ /_  ___  ____ _____ _____ ___  (_)___  ____ _/ /_  ____  / /_
//        / __/ __ \/ _ \/ __ `/ __ `/ __ `__ \/ / __ \/ __ `/ __ \/ __ \/ __/
//       / /_/ / / /  __/ /_/ / /_/ / / / / / / / / / / /_/ / /_/ / /_/ / /_
//       \__/_/ /_/\___/\__, /\__,_/_/ /_/ /_/_/_/ /_/\__, /_.___/\____/\__/
//                     /____/                        /____/
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
```

```
void swap(int* a, int* b);
int partition (int* arr, int l, int r);
void quickSort(int* arr, int l, int r);
void print(int* arr, int n);

int main(){
   int n;
   printf("Enter the number elements: ");
   fflush(stdin);
   scanf("%d", &n);
   int arr[n];
   for(int i = 0; i < n; i++){
       printf("Enter number %d: ", i + 1);
       fflush(stdin);
       scanf("%d", &arr[i]);
   }

   quickSort(arr, 0, n-1);

   printf("\nSorted array: \n");
   print(arr, n);
   return 0;
}

void swap(int* a, int* b){
   int c = *a;
   *a = *b;
   *b = c;
}

int partition (int* arr, int l, int r){
   int pivot = arr[r];
   int i = (l - 1);

   for (int j = l;j <= r- 1;j++){
       if (arr[j] < pivot){
           i++;
           swap(&arr[i], &arr[j]);
       }
   }
   swap(&arr[i + 1], &arr[r]);
   return (i + 1);
}

void quickSort(int* arr, int l, int r){
   if (l < r){
       int pi = partition(arr, l, r);
       pid_t pid = vfork();

       if(pid < 0){
           printf("Fork failed..\n");
           exit(1);
       }
       else if (pid == 0){
           quickSort(arr, l, pi - 1);
           exit(0);
       }
       else{
           quickSort(arr, pi + 1, r);
           wait(NULL);
       }
   }
}

void print(int* arr, int n){
   for (int i = 0; i < n; i++)
       printf("%d ", arr[i]);
   printf("\n");
}
```

**Explanation:**

Here, the right of the partition is sorted by the parent, while the left is done by the child.

## Output:



thegamingbot@pop-os: ~/Downloads/sem-5/OS/lab4

```
thegamingbot@pop-os:~/Downloads/sem-5/OS/lab4$ make 2b
make: '2b' is up to date.
thegamingbot@pop-os:~/Downloads/sem-5/OS/lab4$ ./2b
Enter the number elements: 7
Enter number 1: 5657
Enter number 2: 56574685665
Enter number 3: 7
Enter number 4: 86
Enter number 5: 7
Enter number 6: 45745
Enter number 7: 54654

Sorted array:
7 7 86 5657 45745 54654 740110817
thegamingbot@pop-os:~/Downloads/sem-5/OS/lab4$ 
```

**Question:** Develop your own command shell [say mark it with @] that accepts user commands (System or User Binaries), executes the commands and returns the prompt for further user interaction. Also extend this to support a history feature (if the user types !6 at the command prompt; it should display the most recent execute 6 commands). You may provide validation features such as !10 when there are only 9 files to display the entire history contents and other validations required for the history feature;

## C code:

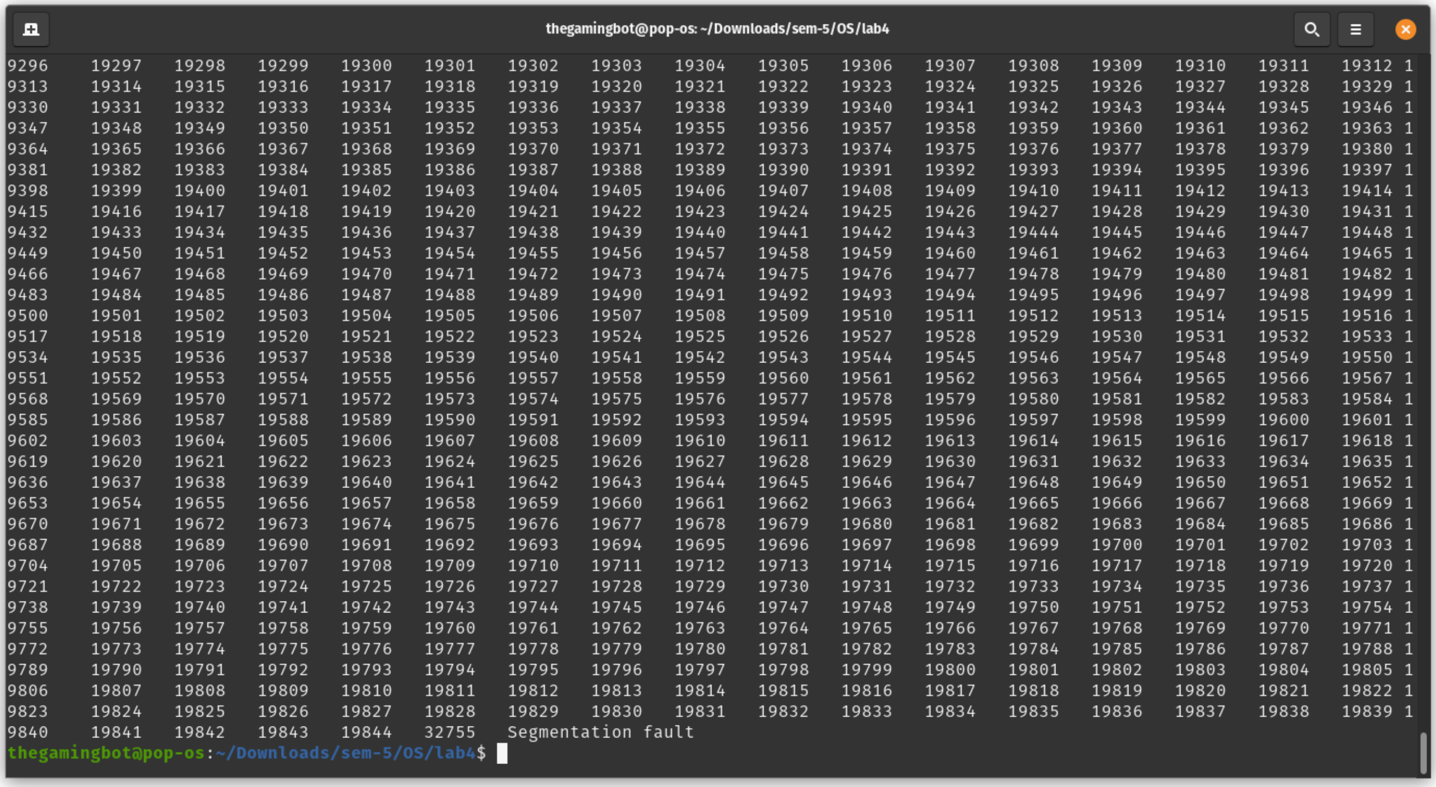
```c
//          __  __                               _             __          __
//         / /_/ /_  ___  ____ _____ _____ ___  (_)___  ____ _/ /_  ____  / /_
//        / __/ __ \/ _ \/ __ `/ __ `/ __ `__ \/ / __ \/ __ `/ __ \/ __ \/ __/
//       / /_/ / / /  __/ /_/ / /_/ / / / / / / / / / / /_/ / /_/ / /_/ / /_
//       \__/_/ /_/\___/\__, /\__,_/_/ /_/ /_/_/_/ /_/\__, /_.___/\____/\__/
//                     /____/                        /____/
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>

int main(){
   int count = 0;
```

```c
    while(vfork() == 0){
        count++;
        printf("%d\t", count);
    }
}
```

## Explanation:

Count is being incremented every time a child process is being created.

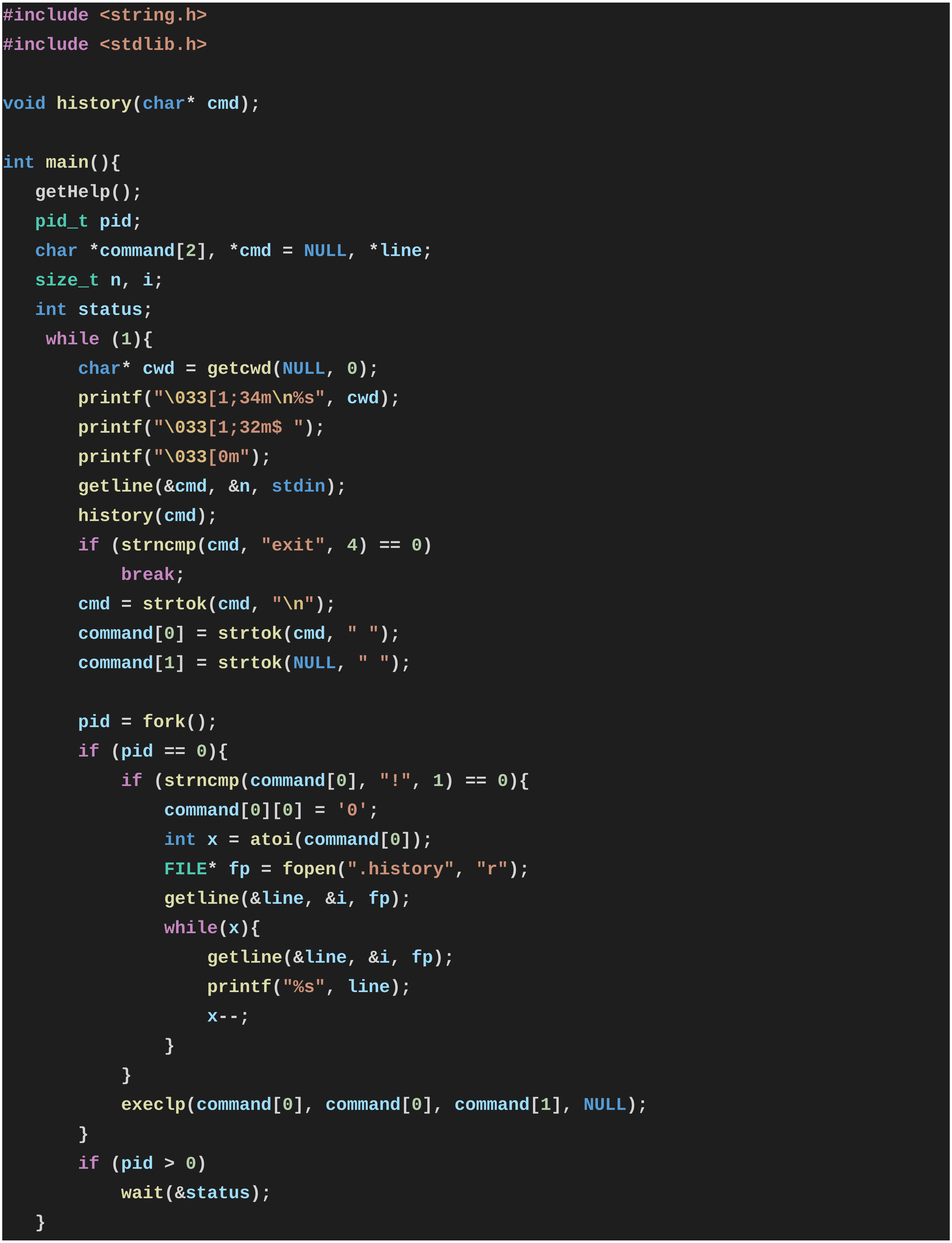
## Output:



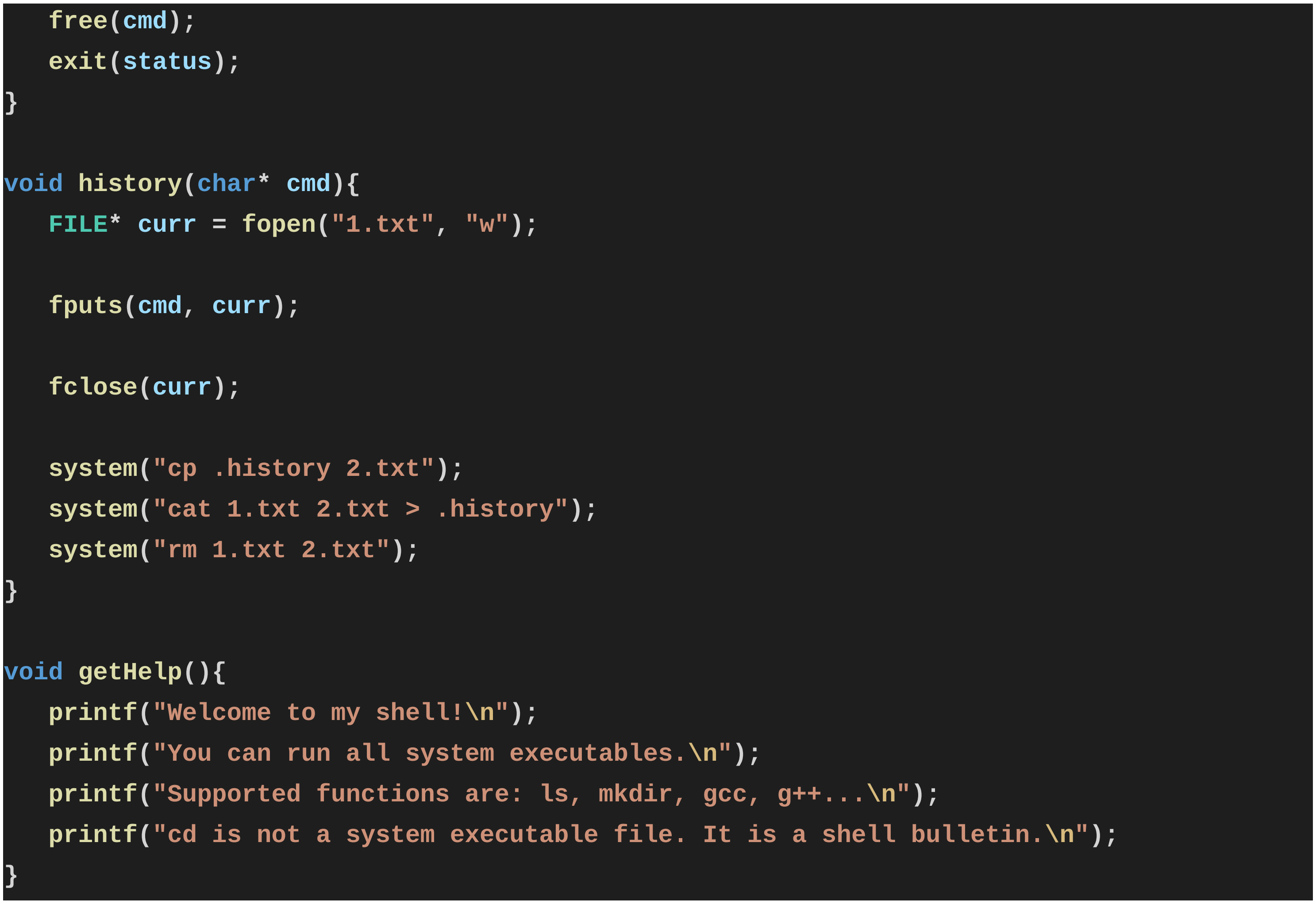
```
9296    19297   19298   19299   19300   19301   19302   19303   19304   19305   19306   19307   19308   19309   19310   19311   19312 1
9313    19314   19315   19316   19317   19318   19319   19320   19321   19322   19323   19324   19325   19326   19327   19328   19329 1
9330    19331   19332   19333   19334   19335   19336   19337   19338   19339   19340   19341   19342   19343   19344   19345   19346 1
9347    19348   19349   19350   19351   19352   19353   19354   19355   19356   19357   19358   19359   19360   19361   19362   19363 1
9364    19365   19366   19367   19368   19369   19370   19371   19372   19373   19374   19375   19376   19377   19378   19379   19380 1
9381    19382   19383   19384   19385   19386   19387   19388   19389   19390   19391   19392   19393   19394   19395   19396   19397 1
9398    19399   19400   19401   19402   19403   19404   19405   19406   19407   19408   19409   19410   19411   19412   19413   19414 1
9415    19416   19417   19418   19419   19420   19421   19422   19423   19424   19425   19426   19427   19428   19429   19430   19431 1
9432    19433   19434   19435   19436   19437   19438   19439   19440   19441   19442   19443   19444   19445   19446   19447   19448 1
9449    19450   19451   19452   19453   19454   19455   19456   19457   19458   19459   19460   19461   19462   19463   19464   19465 1
9466    19467   19468   19469   19470   19471   19472   19473   19474   19475   19476   19477   19478   19479   19480   19481   19482 1
9483    19484   19485   19486   19487   19488   19489   19490   19491   19492   19493   19494   19495   19496   19497   19498   19499 1
9500    19501   19502   19503   19504   19505   19506   19507   19508   19509   19510   19511   19512   19513   19514   19515   19516 1
9517    19518   19519   19520   19521   19522   19523   19524   19525   19526   19527   19528   19529   19530   19531   19532   19533 1
9534    19535   19536   19537   19538   19539   19540   19541   19542   19543   19544   19545   19546   19547   19548   19549   19550 1
9551    19552   19553   19554   19555   19556   19557   19558   19559   19560   19561   19562   19563   19564   19565   19566   19567 1
9568    19569   19570   19571   19572   19573   19574   19575   19576   19577   19578   19579   19580   19581   19582   19583   19584 1
9585    19586   19587   19588   19589   19590   19591   19592   19593   19594   19595   19596   19597   19598   19599   19600   19601 1
9602    19603   19604   19605   19606   19607   19608   19609   19610   19611   19612   19613   19614   19615   19616   19617   19618 1
9619    19620   19621   19622   19623   19624   19625   19626   19627   19628   19629   19630   19631   19632   19633   19634   19635 1
9636    19637   19638   19639   19640   19641   19642   19643   19644   19645   19646   19647   19648   19649   19650   19651   19652 1
9653    19654   19655   19656   19657   19658   19659   19660   19661   19662   19663   19664   19665   19666   19667   19668   19669 1
9670    19671   19672   19673   19674   19675   19676   19677   19678   19679   19680   19681   19682   19683   19684   19685   19686 1
9687    19688   19689   19690   19691   19692   19693   19694   19695   19696   19697   19698   19699   19700   19701   19702   19703 1
9704    19705   19706   19707   19708   19709   19710   19711   19712   19713   19714   19715   19716   19717   19718   19719   19720 1
9721    19722   19723   19724   19725   19726   19727   19728   19729   19730   19731   19732   19733   19734   19735   19736   19737 1
9738    19739   19740   19741   19742   19743   19744   19745   19746   19747   19748   19749   19750   19751   19752   19753   19754 1
9755    19756   19757   19758   19759   19760   19761   19762   19763   19764   19765   19766   19767   19768   19769   19770   19771 1
9772    19773   19774   19775   19776   19777   19778   19779   19780   19781   19782   19783   19784   19785   19786   19787   19788 1
9789    19790   19791   19792   19793   19794   19795   19796   19797   19798   19799   19800   19801   19802   19803   19804   19805 1
9806    19807   19808   19809   19810   19811   19812   19813   19814   19815   19816   19817   19818   19819   19820   19821   19822 1
9823    19824   19825   19826   19827   19828   19829   19830   19831   19832   19833   19834   19835   19836   19837   19838   19839 1
9840    19841   19842   19843   19844   32755   Segmentation fault
thegamingbot@pop-os:~/Downloads/sem-5/OS/lab4$ 
```

## C code:

```c
//            __  __                                _             __          __
//           / /_/ /_  ___  ____ _____ _____ ___  (_)___  ____ _/ /_  ____  / /_
//          / __/ __ \/ _ \/ __ `/ __ `/ __ `__ \/ / __ \/ __ `/ __ \/ __ \/ __/
//         / /_/ / / /  __/ /_/ / /_/ / / / / / / / / / / /_/ / /_/ / /_/ / /_
//         \__/_/ /_/\___/\__, /\__,_/_/ /_/ /_/_/_/ /_/\__, /_.___/\____/\__/
//                       /____/                        /____/
#include <unistd.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/wait.h>
```



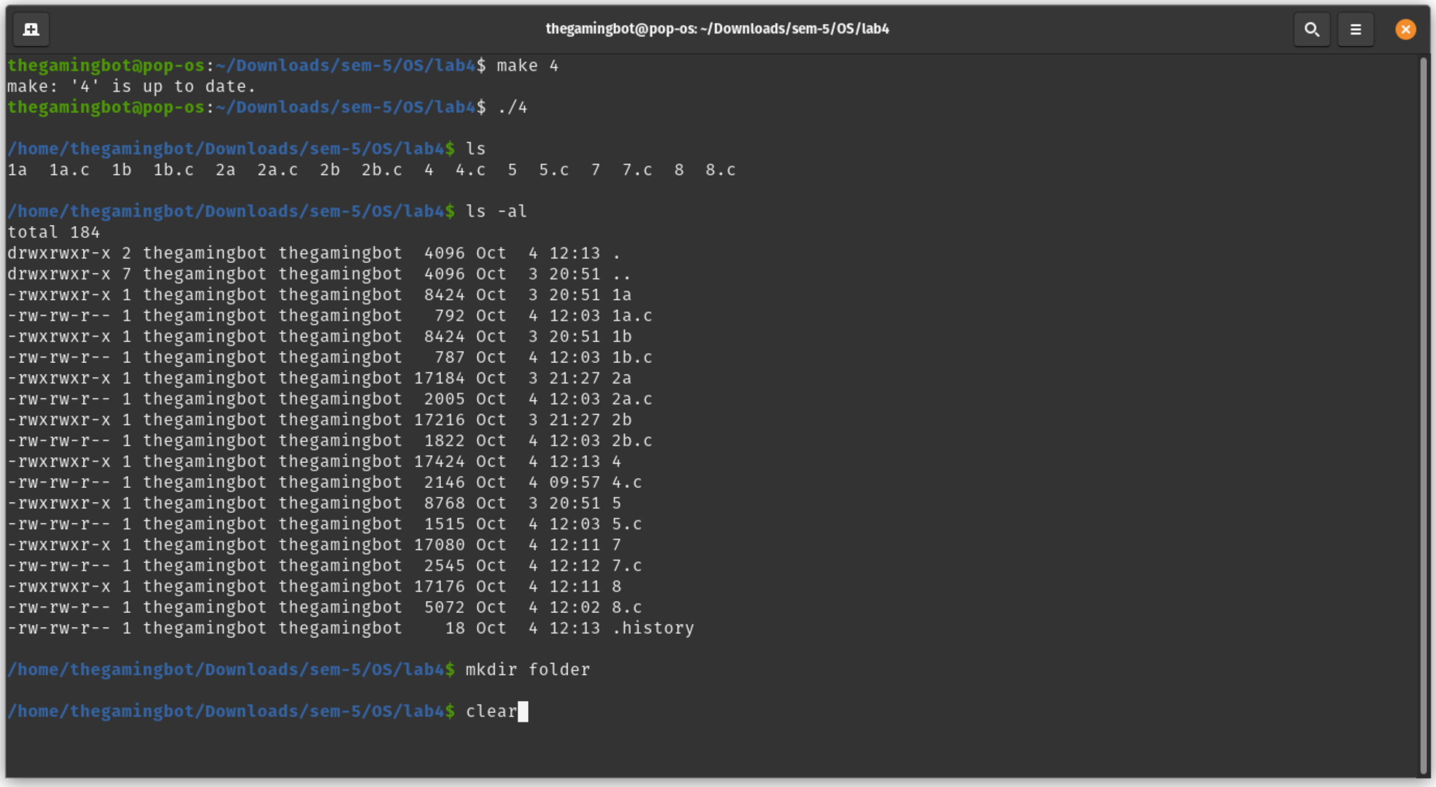
```c
#include <string.h>
#include <stdlib.h>

void history(char* cmd);

int main(){
   getHelp();
   pid_t pid;
   char *command[2], *cmd = NULL, *line;
   size_t n, i;
   int status;
    while (1){
       char* cwd = getcwd(NULL, 0);
       printf("\033[1;34m\n%s", cwd);
       printf("\033[1;32m$ ");
       printf("\033[0m");
       getline(&cmd, &n, stdin);
       history(cmd);
       if (strncmp(cmd, "exit", 4) == 0)
           break;
       cmd = strtok(cmd, "\n");
       command[0] = strtok(cmd, " ");
       command[1] = strtok(NULL, " ");

       pid = fork();
       if (pid == 0){
           if (strncmp(command[0], "!", 1) == 0){
               command[0][0] = '0';
               int x = atoi(command[0]);
               FILE* fp = fopen(".history", "r");
               getline(&line, &i, fp);
               while(x){
                   getline(&line, &i, fp);
                   printf("%s", line);
                   x--;
               }
           }
           execlp(command[0], command[0], command[1], NULL);
       }
       if (pid > 0)
           wait(&status);
   }
```



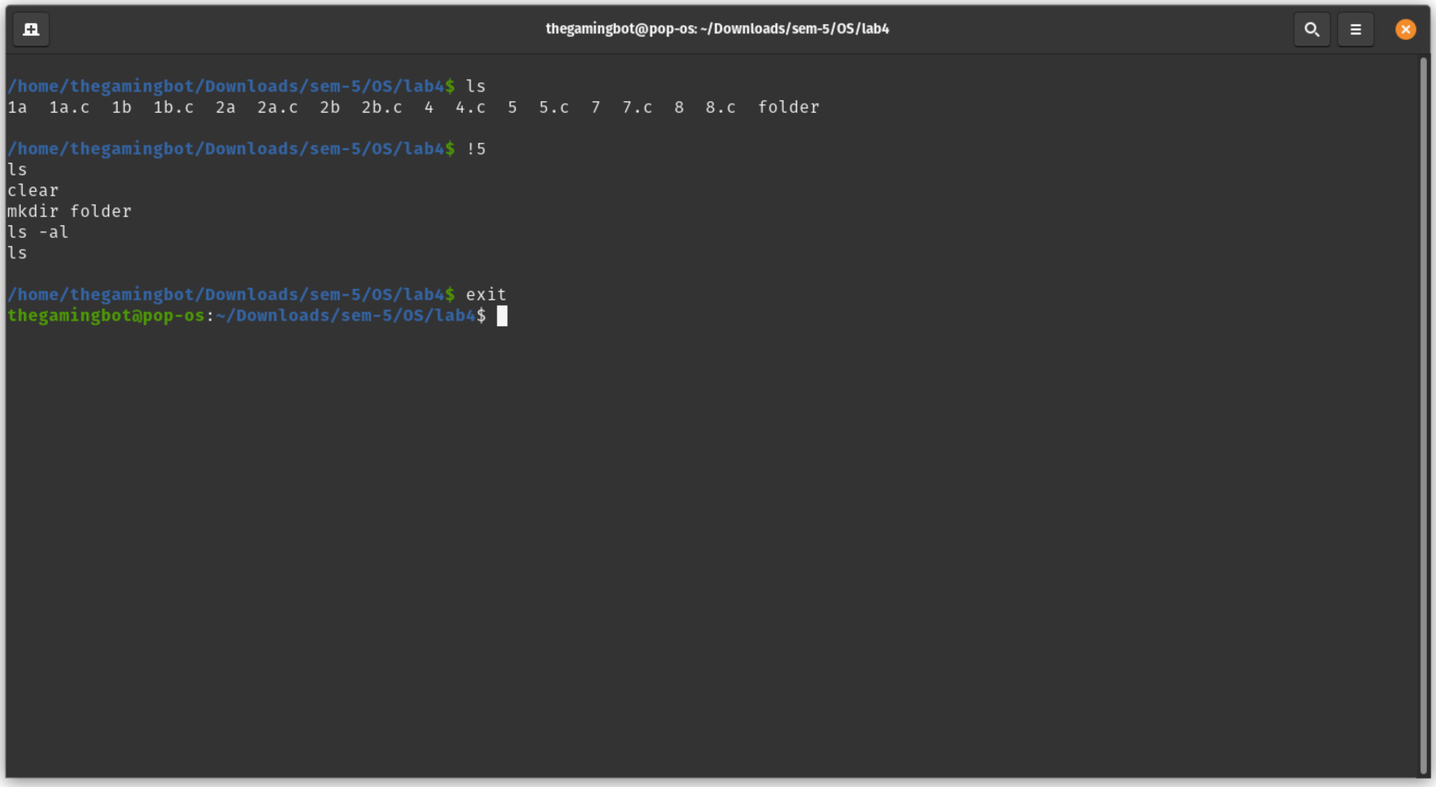
```
   free(cmd);
   exit(status);
}

void history(char* cmd){
   FILE* curr = fopen("1.txt", "w");

   fputs(cmd, curr);

   fclose(curr);

   system("cp .history 2.txt");
   system("cat 1.txt 2.txt > .history");
   system("rm 1.txt 2.txt");
}

void getHelp(){
   printf("Welcome to my shell!\n");
   printf("You can run all system executables.\n");
   printf("Supported functions are: ls, mkdir, gcc, g++...\n");
   printf("cd is not a system executable file. It is a shell bulletin.\n");
}
```

**Explanation:** Every child terminates after the exec call. For history, the latest command is appended at the start of a ".history" file.

## Output:



```
thegamingbot@pop-os:~/Downloads/sem-5/OS/lab4$ make 4
make: '4' is up to date.
thegamingbot@pop-os:~/Downloads/sem-5/OS/lab4$ ./4

/home/thegamingbot/Downloads/sem-5/OS/lab4$ ls
1a  1a.c  1b  1b.c  2a  2a.c  2b  2b.c  4  4.c  5  5.c  7  7.c  8  8.c

/home/thegamingbot/Downloads/sem-5/OS/lab4$ ls -al
total 184
drwxrwxr-x 2 thegamingbot thegamingbot  4096 Oct  4 12:13 .
drwxrwxr-x 7 thegamingbot thegamingbot  4096 Oct  3 20:51 ..
-rwxrwxr-x 1 thegamingbot thegamingbot  8424 Oct  3 20:51 1a
-rw-rw-r-- 1 thegamingbot thegamingbot   792 Oct  4 12:03 1a.c
-rwxrwxr-x 1 thegamingbot thegamingbot  8424 Oct  3 20:51 1b
-rw-rw-r-- 1 thegamingbot thegamingbot   787 Oct  4 12:03 1b.c
-rwxrwxr-x 1 thegamingbot thegamingbot 17184 Oct  3 21:27 2a
-rw-rw-r-- 1 thegamingbot thegamingbot  2005 Oct  4 12:03 2a.c
-rwxrwxr-x 1 thegamingbot thegamingbot 17216 Oct  3 21:27 2b
-rw-rw-r-- 1 thegamingbot thegamingbot  1822 Oct  4 12:03 2b.c
-rwxrwxr-x 1 thegamingbot thegamingbot 17424 Oct  4 12:13 4
-rw-rw-r-- 1 thegamingbot thegamingbot  2146 Oct  4 09:57 4.c
-rwxrwxr-x 1 thegamingbot thegamingbot  8768 Oct  3 20:51 5
-rw-rw-r-- 1 thegamingbot thegamingbot  1515 Oct  4 12:03 5.c
-rwxrwxr-x 1 thegamingbot thegamingbot 17080 Oct  4 12:11 7
-rw-rw-r-- 1 thegamingbot thegamingbot  2545 Oct  4 12:12 7.c
-rwxrwxr-x 1 thegamingbot thegamingbot 17176 Oct  4 12:11 8
-rw-rw-r-- 1 thegamingbot thegamingbot  5072 Oct  4 12:02 8.c
-rw-rw-r-- 1 thegamingbot thegamingbot    18 Oct  4 12:13 .history

/home/thegamingbot/Downloads/sem-5/OS/lab4$ mkdir folder

/home/thegamingbot/Downloads/sem-5/OS/lab4$ clear
```



```

/home/thegamingbot/Downloads/sem-5/OS/lab4$ ls
1a  1a.c  1b  1b.c  2a  2a.c  2b  2b.c  4  4.c  5  5.c  7  7.c  8  8.c  folder

/home/thegamingbot/Downloads/sem-5/OS/lab4$ !5
ls
clear
mkdir folder
ls -al
ls

/home/thegamingbot/Downloads/sem-5/OS/lab4$ exit
thegamingbot@pop-os:~/Downloads/sem-5/OS/lab4$ 
```

**Question:** Develop a multiprocessing version of Histogram generator to count the occurrence of various characters in a given text.

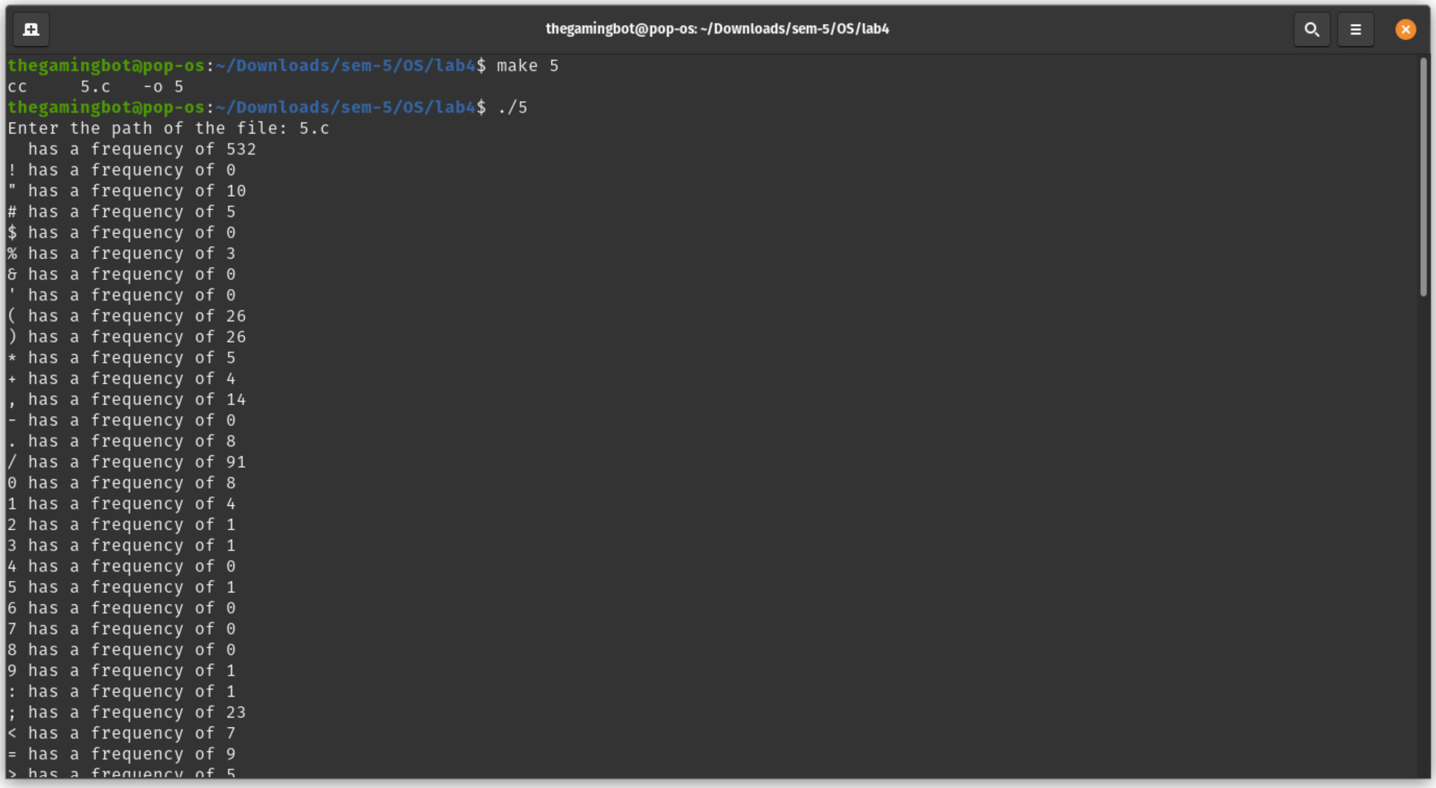
**C code:**

```c
//          __  __                               _             __          __
//         / /_/ /_  ___  ____ _____ _____ ___  (_)___  ____ _/ /_  ____  / /_
//        / __/ __ \/ _ \/ __ `/ __ `/ __ `__ \/ / __ \/ __ `/ __ \/ __ \/ __/
//       / /_/ / / /  __/ /_/ / /_/ / / / / / / / / / / /_/ / /_/ / /_/ / /_
//       \__/_/ /_/\___/\__, /\__,_/_/ /_/ /_/_/_/ /_/\__, /_.___/\____/\__/
//                     /____/                        /____/
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

void histogram(char* filePath, char x);
void recursion(char* filePath, int i);

int main(){
   char filePath[100];
   printf("Enter the path of the file: ");
   scanf("%s", filePath);

   recursion(filePath, 0);
   return 0;
}

void histogram(char* filePath, char x){
   FILE* fp = fopen(filePath, "r");
   char c;
   int frequency = 0;
   do{
       c = fgetc(fp);
       if(feof(fp))
           break;
       if(c == x)
           frequency++;
   }
   while(1);
   printf("%c has a frequency of %d\n", x, frequency);
}
```

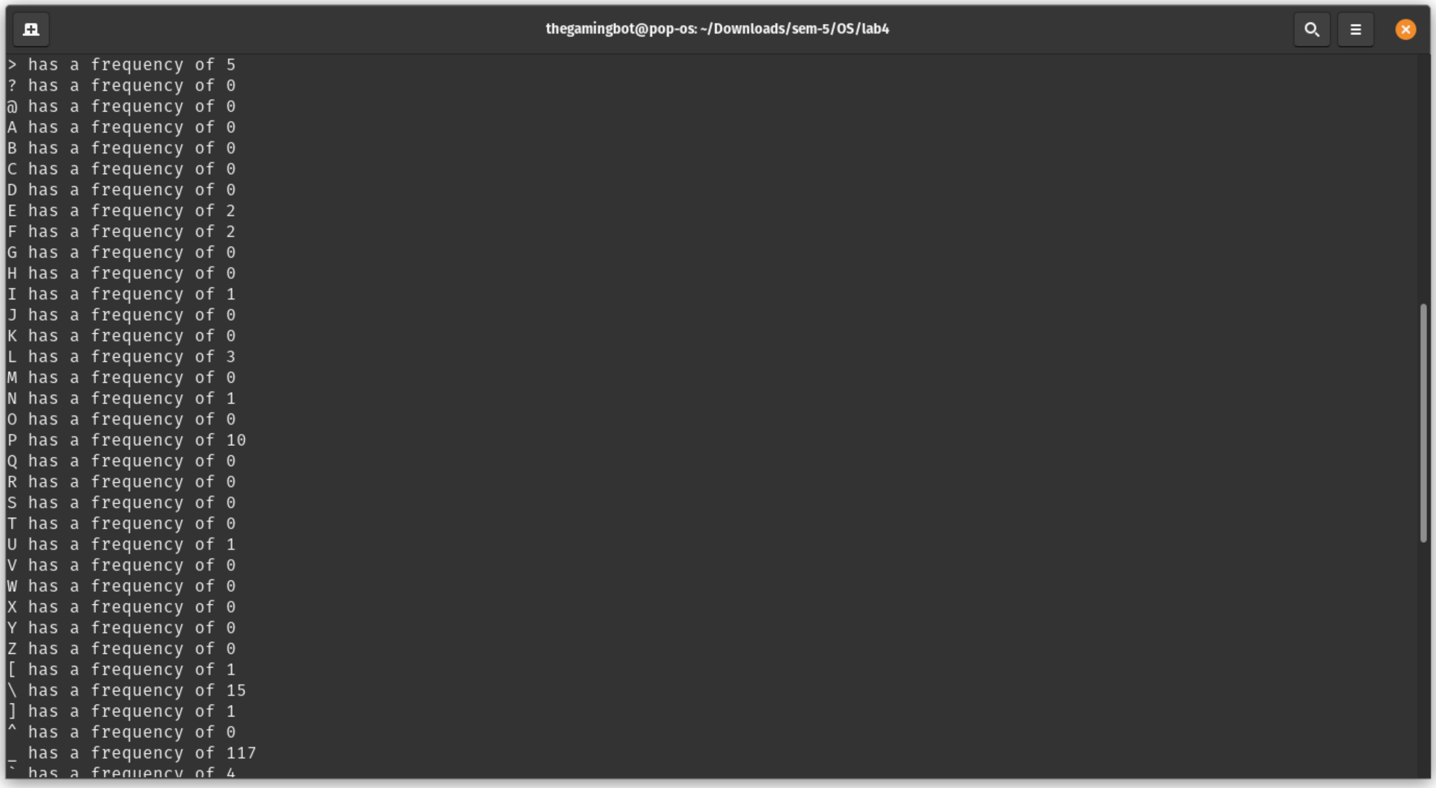
```c
void recursion(char* filePath, int i){
   if(i < 95){
       pid_t pid = vfork();
       if(pid < 0){
           printf("Fork failed..\n");
           exit(1);
       }
       else if (pid == 0){
           char x = i + 32;
           histogram(filePath, x);
           exit(0);
       }
       else{
           wait(NULL);
           recursion(filePath, i + 1);
       }
   }
}
```

**Explanation:** A new child is created for each of the ASCII characters from 32 to 127. Each child loops through the file to find the letter assigned to it.

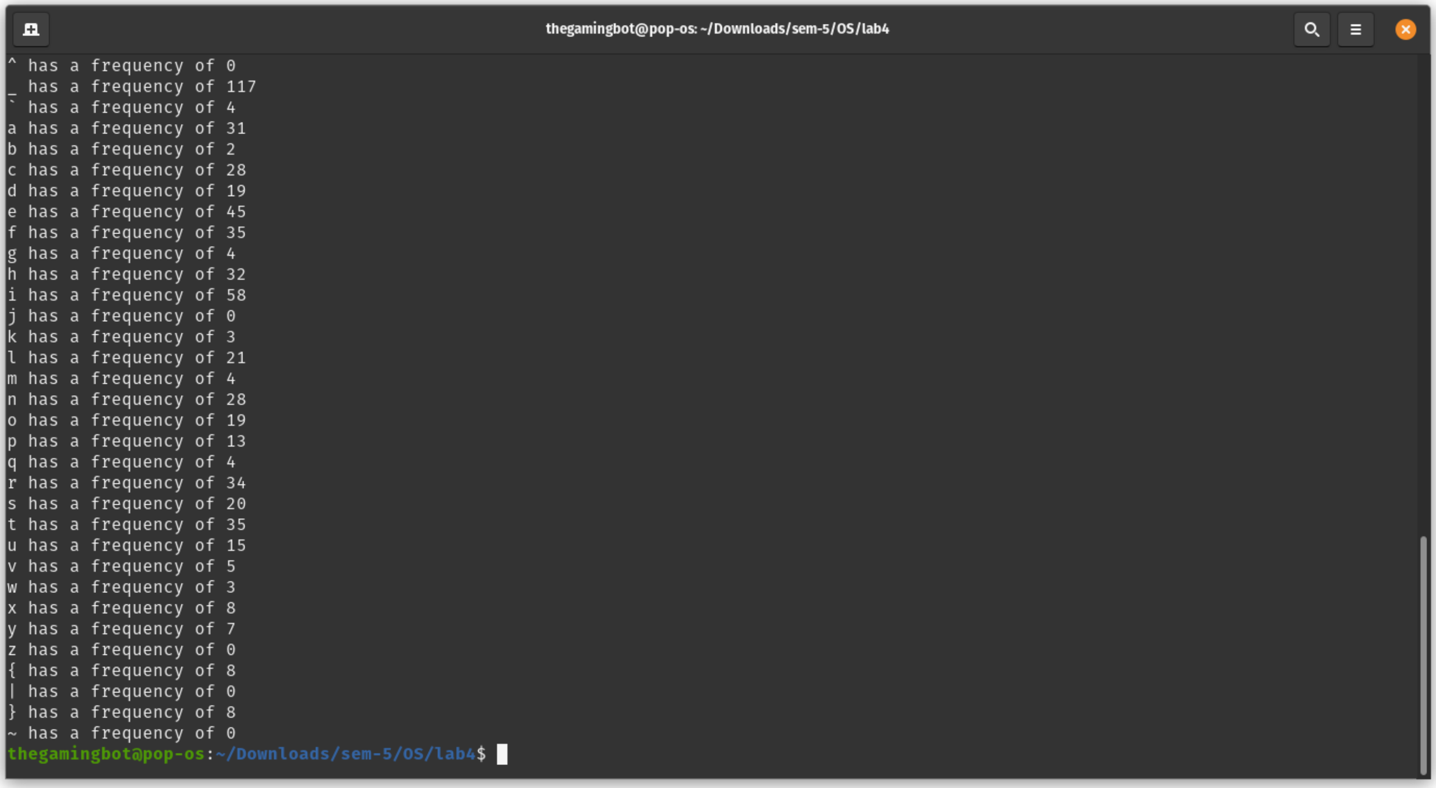
**Output:**



```
thegamingbot@pop-os:~/Downloads/sem-5/OS/lab4$ make 5
cc     5.c   -o 5
thegamingbot@pop-os:~/Downloads/sem-5/OS/lab4$ ./5
Enter the path of the file: 5.c
  has a frequency of 532
! has a frequency of 0
" has a frequency of 10
# has a frequency of 5
$ has a frequency of 0
% has a frequency of 3
& has a frequency of 0
' has a frequency of 0
( has a frequency of 26
) has a frequency of 26
* has a frequency of 5
+ has a frequency of 4
, has a frequency of 14
- has a frequency of 0
. has a frequency of 8
/ has a frequency of 91
0 has a frequency of 8
1 has a frequency of 4
2 has a frequency of 1
3 has a frequency of 1
4 has a frequency of 0
5 has a frequency of 1
6 has a frequency of 0
7 has a frequency of 0
8 has a frequency of 0
9 has a frequency of 1
: has a frequency of 1
; has a frequency of 23
< has a frequency of 7
= has a frequency of 9
> has a frequency of 5
```



```
> has a frequency of 5
? has a frequency of 0
@ has a frequency of 0
A has a frequency of 0
B has a frequency of 0
C has a frequency of 0
D has a frequency of 0
E has a frequency of 2
F has a frequency of 2
G has a frequency of 0
H has a frequency of 0
I has a frequency of 1
J has a frequency of 0
K has a frequency of 0
L has a frequency of 3
M has a frequency of 0
N has a frequency of 1
O has a frequency of 0
P has a frequency of 10
Q has a frequency of 0
R has a frequency of 0
S has a frequency of 0
T has a frequency of 0
U has a frequency of 1
V has a frequency of 0
W has a frequency of 0
X has a frequency of 0
Y has a frequency of 0
Z has a frequency of 0
[ has a frequency of 1
\ has a frequency of 15
] has a frequency of 1
^ has a frequency of 0
_ has a frequency of 117
` has a frequency of 4
```



```
^ has a frequency of 0
_ has a frequency of 117
` has a frequency of 4
a has a frequency of 31
b has a frequency of 2
c has a frequency of 28
d has a frequency of 19
e has a frequency of 45
f has a frequency of 35
g has a frequency of 4
h has a frequency of 32
i has a frequency of 58
j has a frequency of 0
k has a frequency of 3
l has a frequency of 21
m has a frequency of 4
n has a frequency of 28
o has a frequency of 19
p has a frequency of 13
q has a frequency of 4
r has a frequency of 34
s has a frequency of 20
t has a frequency of 35
u has a frequency of 15
v has a frequency of 5
w has a frequency of 3
x has a frequency of 8
y has a frequency of 7
z has a frequency of 0
{ has a frequency of 8
| has a frequency of 0
} has a frequency of 8
~ has a frequency of 0
thegamingbot@pop-os:~/Downloads/sem-5/OS/lab4$
```

**Question:** Develop a multiprocessing version of matrix multiplication. Say for a result 3*3 matrix the most efficient form of parallelization can be 9 processes, each of which computes the net resultant value of a row (matrix1) multiplied by column (matrix2). For programmers convenience you can start with 4 processes, but as I said each result value can be computed parallel independent of the other processes in execution.

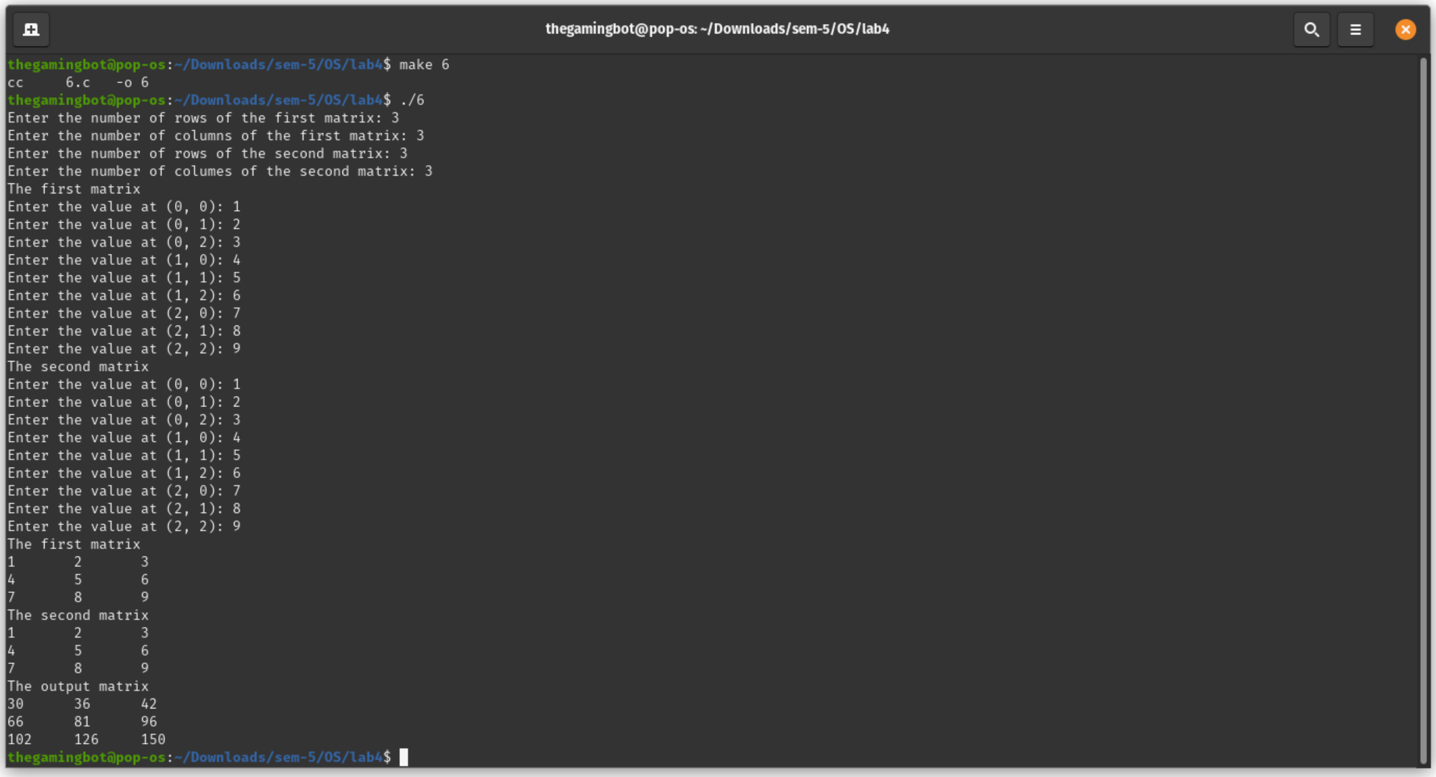
**C code:**

```
//          __  __                               _             __          __
//         / /_/ /_  ___  ____ _____ _____ ___  (_)___  ____ _/ /_  ____  / /_
//        / __/ __ \/ _ \/ __ `/ __ `/ __ `__ \/ / __ \/ __ `/ __ \/ __ \/ __/
//       / /_/ / / /  __/ /_/ / /_/ / / / / / / / / /_/ / /_/ / /_/ / /_
//       \__/_/ /_/\___/\__, /\__,_/_/ /_/ /_/_/_/ /_/\__, /_.___/\____/\__/
//                     /____/                        /____/
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <string.h>
int main(){
   int m1, n1, m2, n2;
   printf("Enter the number of rows of the first matrix: ");
   scanf("%d", &m1);
   printf("Enter the number of columns of the first matrix: ");
   scanf("%d", &n1);
   printf("Enter the number of rows of the second matrix: ");
   scanf("%d", &m2);
   printf("Enter the number of columns of the second matrix: ");
   scanf("%d", &n2);
   if (n1 != m2){
       printf("Matrix multiplication not possible.\n");
       exit(1);
   }
   int arr1[m1][n1], arr2[m2][n2], out[m1][n2];
   printf("The first matrix\n");
   for(int i = 0; i < m1; i++){
       for(int j = 0; j < n1; j++){
           printf("Enter the value at (%d, %d): ", i, j);
           scanf("%d", &arr1[i][j]);
       }
   }
```

```
    printf("The second matrix\n");
    for(int i = 0; i < m2; i++){
        for(int j = 0; j < n2; j++){
            printf("Enter the value at (%d, %d): ", i, j);
            scanf("%d", &arr2[i][j]);
        }
    }
    printf("The first matrix\n");
    for(int i = 0; i < m1; i++){
        for(int j = 0; j < n1; j++){
            printf("%d\t", arr1[i][j]);
        }
        printf("\n");
    }
    printf("The second matrix\n");
    for(int i = 0; i < m2; i++){
        for(int j = 0; j < n2; j++){
            printf("%d\t", arr2[i][j]);
        }
        printf("\n");
    }
    for (int i = 0; i < m1; i++){
        for (int j = 0; j < n2; j++){
            out[i][j] = 0;
        }
    }
    for(int i = 0; i < m1; i++){
        for(int j = 0; j < n2; j++){
            pid_t pid = vfork();
            if(pid == 0){
                for(int k = 0; k < n1; k++)
                    out[i][j] += arr1[i][k] * arr2[k][j];
                exit(0);
            }
        }
    }
    printf("The output matrix\n");
    for(int i = 0; i < m1; i++){
        for(int j = 0; j < n2; j++){
            printf("%d\t", out[i][j]);
        }
        printf("\n");
    }
}
```

**Explanation:** A new process is created for each index of the output matrix. It computers the output and exits.

**Output:**

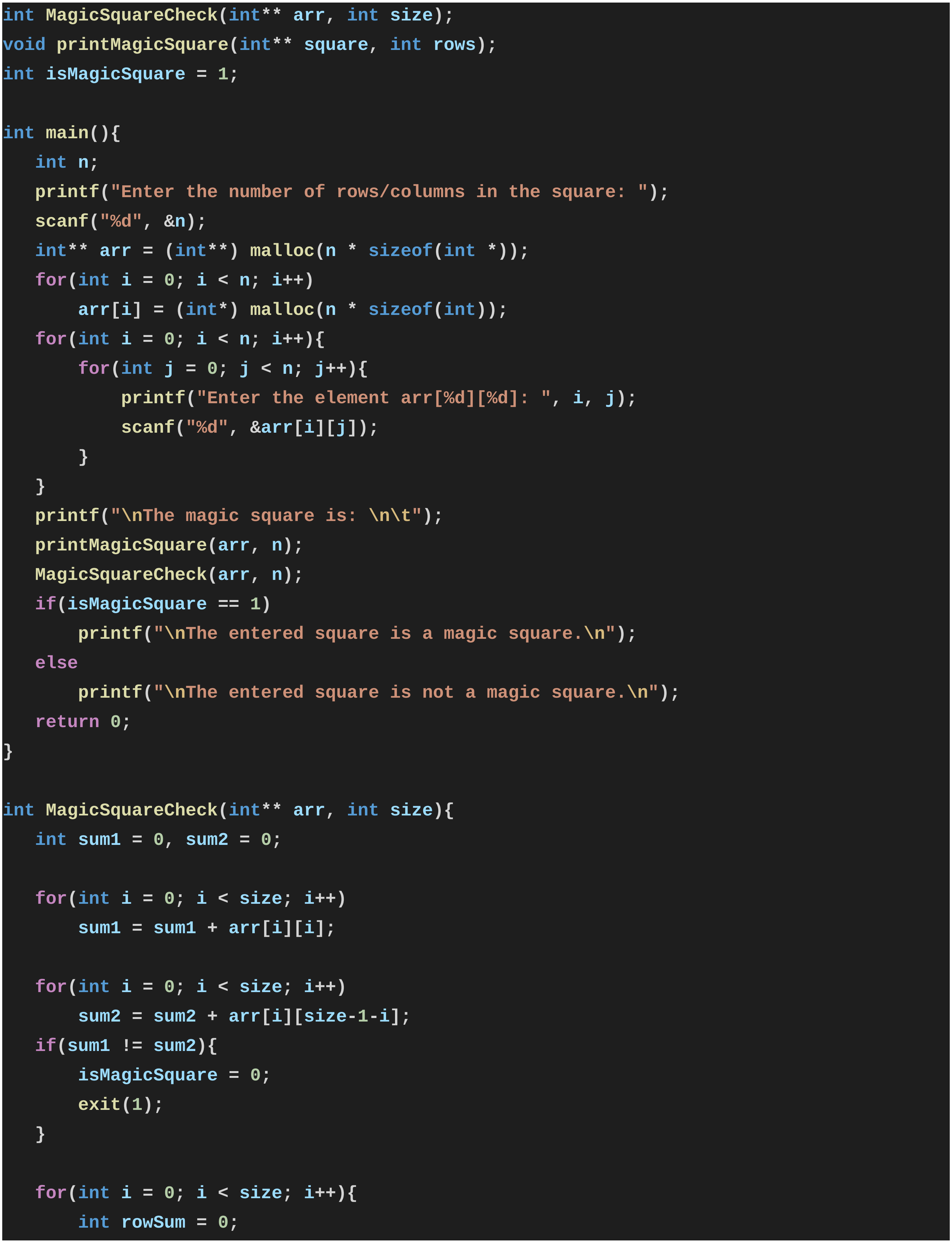


```
thegamingbot@pop-os:~/Downloads/sem-5/OS/lab4$ make 6
cc     6.c   -o 6
thegamingbot@pop-os:~/Downloads/sem-5/OS/lab4$ ./6
Enter the number of rows of the first matrix: 3
Enter the number of columns of the first matrix: 3
Enter the number of rows of the second matrix: 3
Enter the number of columes of the second matrix: 3
The first matrix
Enter the value at (0, 0): 1
Enter the value at (0, 1): 2
Enter the value at (0, 2): 3
Enter the value at (1, 0): 4
Enter the value at (1, 1): 5
Enter the value at (1, 2): 6
Enter the value at (2, 0): 7
Enter the value at (2, 1): 8
Enter the value at (2, 2): 9
The second matrix
Enter the value at (0, 0): 1
Enter the value at (0, 1): 2
Enter the value at (0, 2): 3
Enter the value at (1, 0): 4
Enter the value at (1, 1): 5
Enter the value at (1, 2): 6
Enter the value at (2, 0): 7
Enter the value at (2, 1): 8
Enter the value at (2, 2): 9
The first matrix
1       2       3
4       5       6
7       8       9
The second matrix
1       2       3
4       5       6
7       8       9
The output matrix
30      36      42
66      81      96
102     126     150
thegamingbot@pop-os:~/Downloads/sem-5/OS/lab4$ 
```

**Question:** Develop a parallelized application to check for if a user input square matrix is a magic square or not. No of processes again can be optimal as w.r.t to matrix exercise above.

**C code:**

```
//          __  __                               _             __          __
//         / /_/ /_  ___  ____ _____ _____ ___  (_)___  ____ _/ /_  ____  / /_
//        / __/ __ \/ _ \/ __ `/ __ `/ __ `__ \/ / __ \/ __ `/ __ \/ __ \/ __/
//       / /_/ / / /  __/ /_/ / /_/ / / / / / / / / / / /_/ / /_/ / /_/ / /_
//       \__/_/ /_/\___/\__, /\__,_/_/ /_/ /_/_/_/ /_/\__, /_.___/\____/\__/
//                     /____/                        /____/
#include <unistd.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <stdlib.h>
```

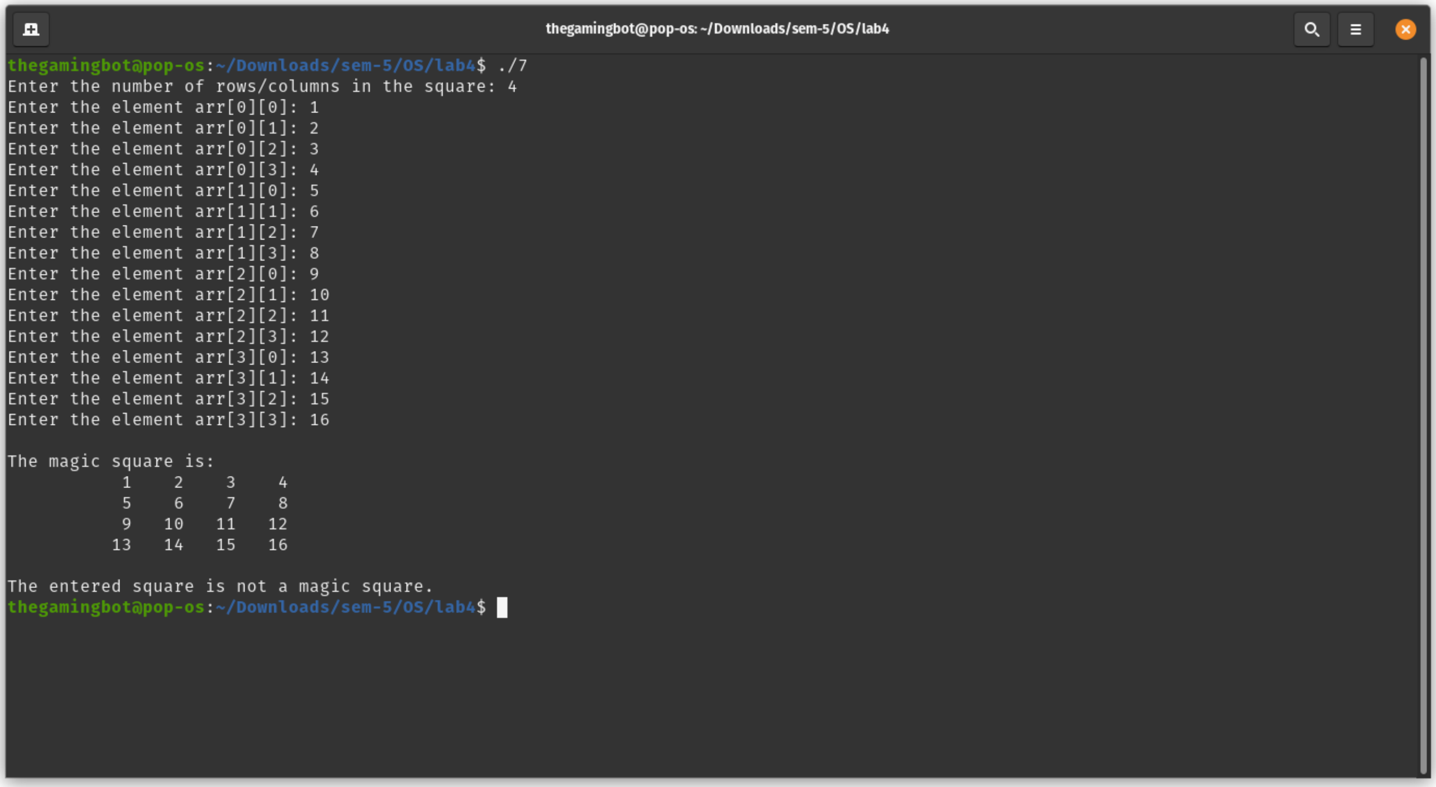


```c
int MagicSquareCheck(int** arr, int size);
void printMagicSquare(int** square, int rows);
int isMagicSquare = 1;

int main(){
    int n;
    printf("Enter the number of rows/columns in the square: ");
    scanf("%d", &n);
    int** arr = (int**) malloc(n * sizeof(int *));
    for(int i = 0; i < n; i++)
        arr[i] = (int*) malloc(n * sizeof(int));
    for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++){
            printf("Enter the element arr[%d][%d]: ", i, j);
            scanf("%d", &arr[i][j]);
        }
    }
    printf("\nThe magic square is: \n\t");
    printMagicSquare(arr, n);
    MagicSquareCheck(arr, n);
    if(isMagicSquare == 1)
        printf("\nThe entered square is a magic square.\n");
    else
        printf("\nThe entered square is not a magic square.\n");
    return 0;
}

int MagicSquareCheck(int** arr, int size){
    int sum1 = 0, sum2 = 0;

    for(int i = 0; i < size; i++)
        sum1 = sum1 + arr[i][i];

    for(int i = 0; i < size; i++)
        sum2 = sum2 + arr[i][size-1-i];
    if(sum1 != sum2){
        isMagicSquare = 0;
        exit(1);
    }

    for(int i = 0; i < size; i++){
        int rowSum = 0;
```

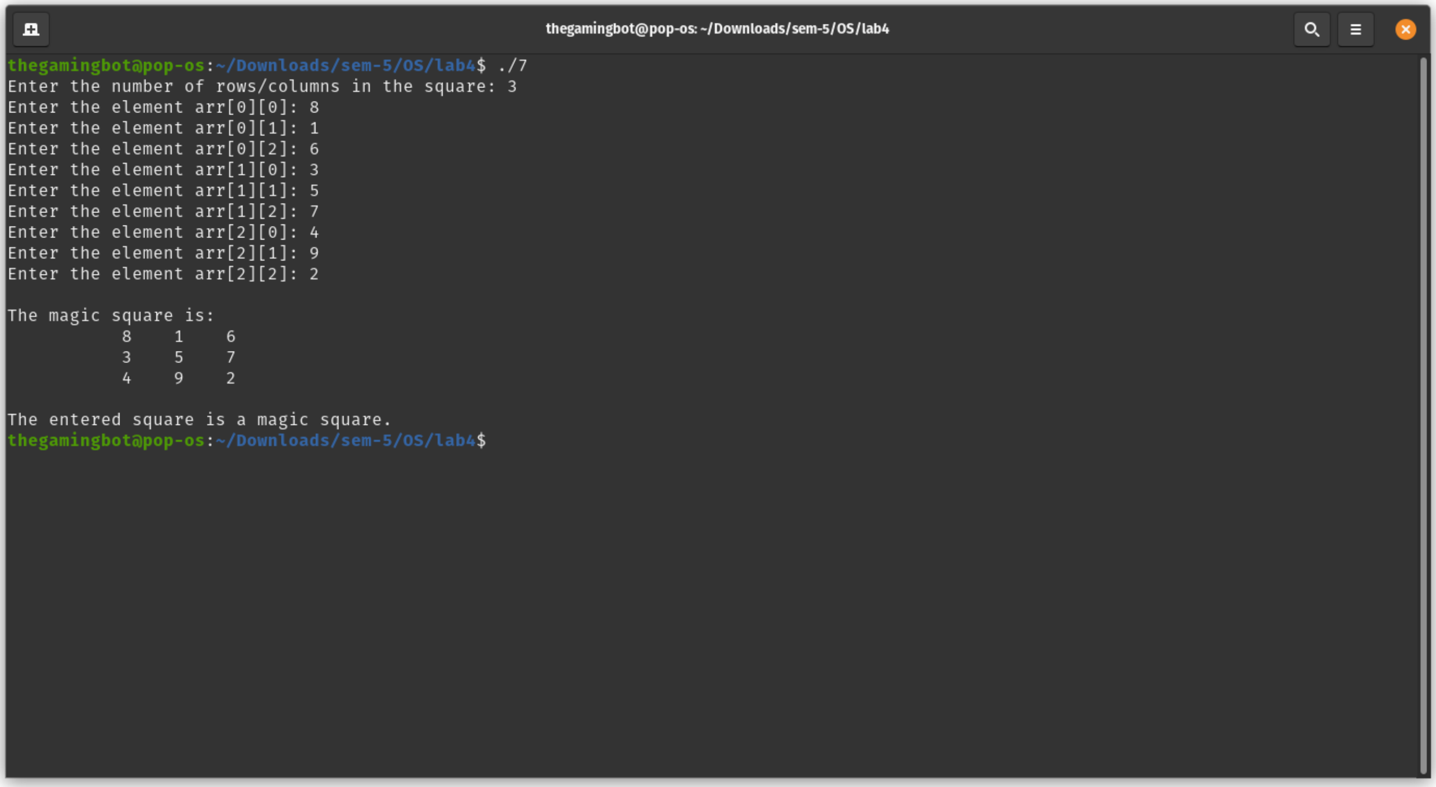
```
       pid_t pid = vfork();
       if(pid == 0){
           for(int j = 0; j < size; j++)
               rowSum += arr[i][j];
           if(rowSum != sum1){
               isMagicSquare = 0;
               exit(1);
           }
           exit(0);
       }
   }

   for(int i = 0; i < size; i++){
       int colSum = 0;
       pid_t pid = vfork();
       if(pid == 0){
           for(int j = 0; j < size; j++)
               colSum += arr[j][i];
           if(sum1 != colSum){
               isMagicSquare = 0;
               exit(1);
           }
           exit(0);
       }
   }
   return 1;
}

void printMagicSquare(int** square, int rows){
   for(int i = 0; i < rows; i++){
       for(int j = 0; j < rows; j++){
           printf("%4d ", square[i][j]);
       }
       printf("\n\t");
   }
}
```

**Explanation:** A new process is created for each row and column computation.

# Output:



```
thegamingbot@pop-os:~/Downloads/sem-5/OS/lab4$ ./7
Enter the number of rows/columns in the square: 4
Enter the element arr[0][0]: 1
Enter the element arr[0][1]: 2
Enter the element arr[0][2]: 3
Enter the element arr[0][3]: 4
Enter the element arr[1][0]: 5
Enter the element arr[1][1]: 6
Enter the element arr[1][2]: 7
Enter the element arr[1][3]: 8
Enter the element arr[2][0]: 9
Enter the element arr[2][1]: 10
Enter the element arr[2][2]: 11
Enter the element arr[2][3]: 12
Enter the element arr[3][0]: 13
Enter the element arr[3][1]: 14
Enter the element arr[3][2]: 15
Enter the element arr[3][3]: 16

The magic square is:
           1    2    3    4
           5    6    7    8
           9   10   11   12
          13   14   15   16

The entered square is not a magic square.
thegamingbot@pop-os:~/Downloads/sem-5/OS/lab4$ 
```



```
thegamingbot@pop-os:~/Downloads/sem-5/OS/lab4$ ./7
Enter the number of rows/columns in the square: 3
Enter the element arr[0][0]: 8
Enter the element arr[0][1]: 1
Enter the element arr[0][2]: 6
Enter the element arr[1][0]: 3
Enter the element arr[1][1]: 5
Enter the element arr[1][2]: 7
Enter the element arr[2][0]: 4
Enter the element arr[2][1]: 9
Enter the element arr[2][2]: 2

The magic square is:
           8    1    6
           3    5    7
           4    9    2

The entered square is a magic square.
thegamingbot@pop-os:~/Downloads/sem-5/OS/lab4$
```

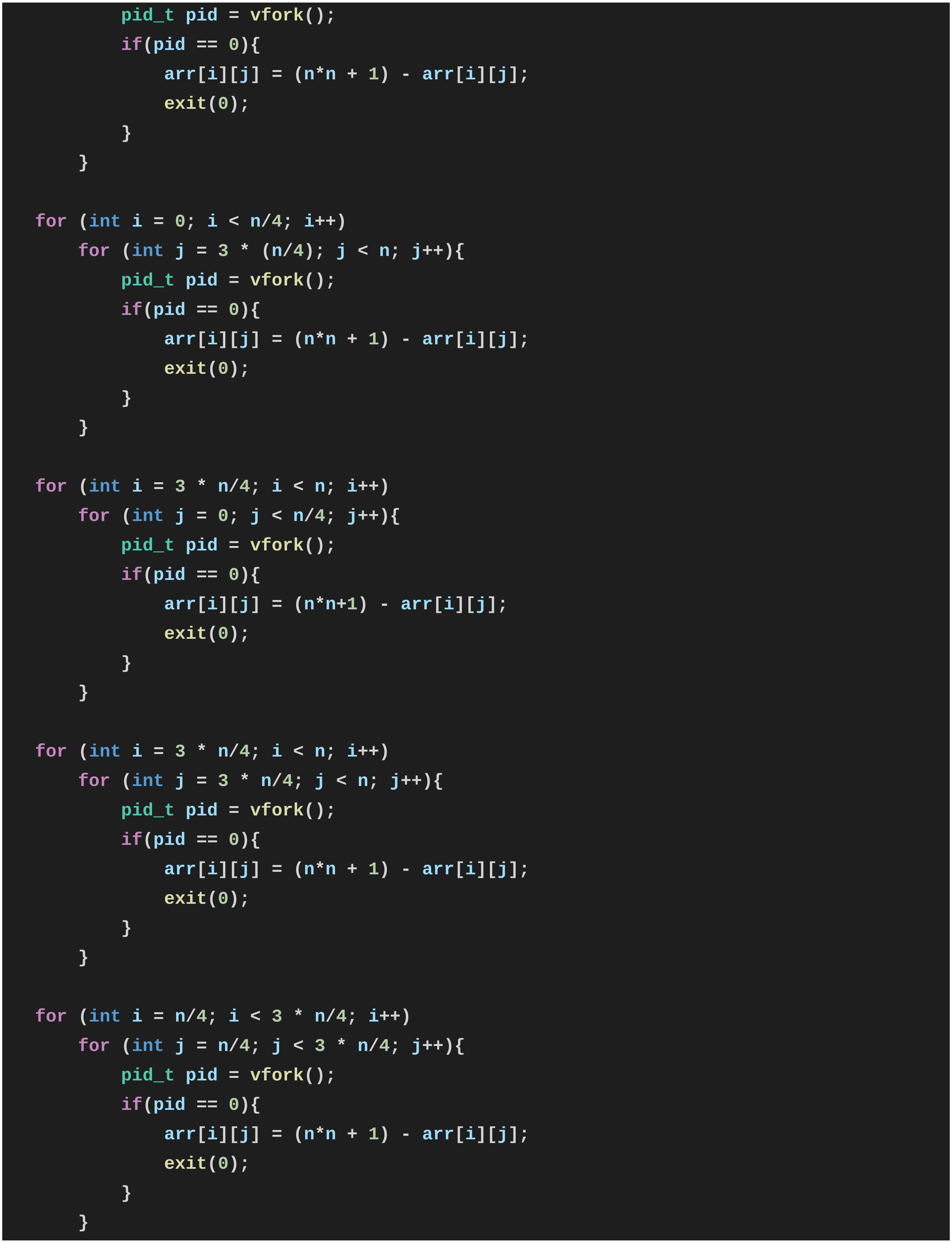
**Question:** Extend the above to also support magic square generation (u can take as input the order of the matrix..refer the net for algorithms for odd and even version...)

**C code:**

```c
//          __  __                               _             __          __
//         / /_/ /_  ___  ____ _____ _____ ___  (_)___  ____ _/ /_  ____  / /_
//        / __/ __ \/ _ \/ __ `/ __ `/ __ `__ \/ / __ \/ __ `/ __ \/ __ \/ __/
//       / /_/ / / /  __/ /_/ / /_/ / / / / / / / / / / /_/ / /_/ / /_/ / /_
//       \__/_/ /_/\___/\__, /\__,_/_/ /_/ /_/_/_/ /_/\__, /_.___/\____/\__/
//                     /____/                        /____/
#include <unistd.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <stdlib.h>

int** oddMagicSquare(int n) {
   if (n < 3 || n % 2 == 0)
       return NULL;

   int value = 0;
   int c = n / 2, r = 0;

   int** arr = (int**)malloc(n*sizeof(int*));

   for(int i = 0; i < n; i++)
       arr[i] = (int*)malloc(n*sizeof(int));

   while (++value <= n * n) {
       pid_t pid = vfork();
       if (pid == 0){
           arr[r][c] = value;
           if (r == 0) {
               if (c == n - 1) {
                   r++;
               }
               else {
                   r = n - 1;
                   c++;
               }
           }
```

```
            else if (c == n - 1) {
                r--;
                c = 0;
            }
            else if (arr[r - 1][c + 1] == 0) {
                r--;
                c++;
            }
            else {
                r++;
            }
            exit(0);
        }
    }
    return arr;
}

int** singlyEvenMagicSquare(int n) {
    if (n < 6 || n % 4 != 2)
        return NULL;

    int size = n * n;
    int half = n / 2;
    int subGridSize = size / 4, i;

    int** subGrid = oddMagicSquare(half);
    int gridFactors[] = {0, 2, 3, 1};
    int** arr = (int**)malloc(n*sizeof(int*));

    for(i=0;i<n;i++)
        arr[i] = (int*)malloc(n*sizeof(int));

    for (int r = 0; r < n; r++) {
        for (int c = 0; c < n; c++) {
            pid_t child = vfork();
            if(child == 0){
                int grid = (r / half) * 2 + (c / half);
                arr[r][c] = subGrid[r % half][c % half];
                arr[r][c] += gridFactors[grid] * subGridSize;
                exit(0);
            }
        }
```

```
   }

   int left = half / 2;
   int right = left - 1;

   for (int r = 0; r < half; r++)
       for (int c = 0; c < n; c++) {
           pid_t pid = vfork();
           if (pid == 0){
               if (c < left || c >= n - right || (c == left && r == left)) {
                   if (c == 0 && r == left)
                       exit(0);
                   int tmp = arr[r][c];
                   arr[r][c] = arr[r + half][c];
                   arr[r + half][c] = tmp;
               }
               exit(0);
           }
       }
   return arr;
}

int** doublyEvenMagicSquare(int n){
   if (n < 4 || n % 4 != 0)
       return NULL;

   int** arr = (int**)malloc(n*sizeof(int*));

   for(int i=0;i<n;i++)
       arr[i] = (int*)malloc(n*sizeof(int));

   for (int i = 0; i < n; i++)
       for (int j = 0; j < n; j++){
           pid_t pid = vfork();
           if(pid == 0){
               arr[i][j] = (n*i) + j + 1;
               exit(0);
           }
       }

   for (int i = 0; i < n/4; i++)
       for (int j = 0; j < n/4; j++){
```



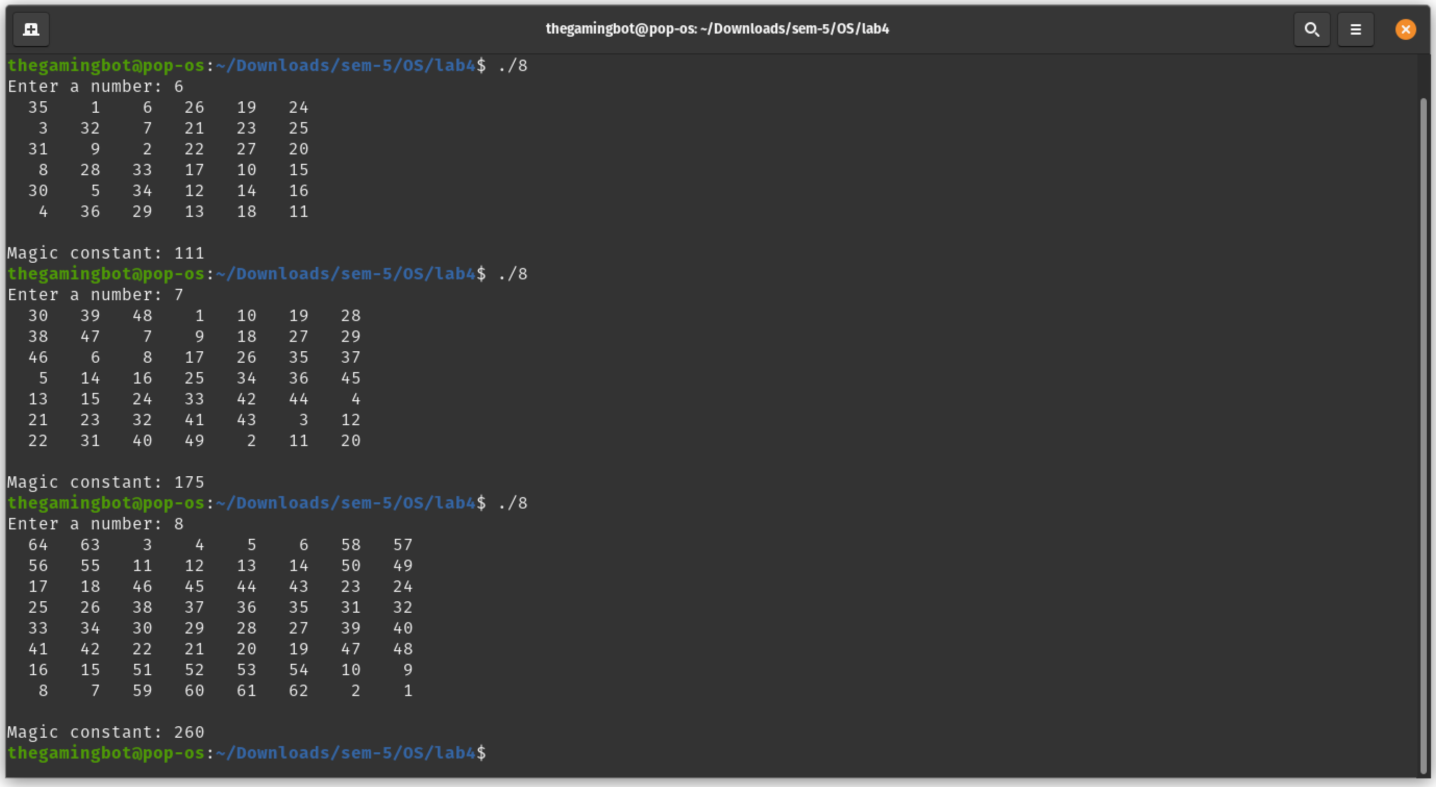
```
        pid_t pid = vfork();
        if(pid == 0){
            arr[i][j] = (n*n + 1) - arr[i][j];
            exit(0);
        }
    }

for (int i = 0; i < n/4; i++)
    for (int j = 3 * (n/4); j < n; j++){
        pid_t pid = vfork();
        if(pid == 0){
            arr[i][j] = (n*n + 1) - arr[i][j];
            exit(0);
        }
    }

for (int i = 3 * n/4; i < n; i++)
    for (int j = 0; j < n/4; j++){
        pid_t pid = vfork();
        if(pid == 0){
            arr[i][j] = (n*n+1) - arr[i][j];
            exit(0);
        }
    }

for (int i = 3 * n/4; i < n; i++)
    for (int j = 3 * n/4; j < n; j++){
        pid_t pid = vfork();
        if(pid == 0){
            arr[i][j] = (n*n + 1) - arr[i][j];
            exit(0);
        }
    }

for (int i = n/4; i < 3 * n/4; i++)
    for (int j = n/4; j < 3 * n/4; j++){
        pid_t pid = vfork();
        if(pid == 0){
            arr[i][j] = (n*n + 1) - arr[i][j];
            exit(0);
        }
    }
    return arr;
}

void printMagicSquare(int** square, int rows){
    for(int i = 0; i < rows; i++){
        for(int j = 0; j < rows; j++){
            printf("%4d ", square[i][j]);
        }
        printf("\n");
    }
    printf("\nMagic constant: %d\n", (rows * rows + 1) * rows / 2);
}

int main(){
    int n;
    printf("Enter a number: ");
    scanf("%d", &n);
    if (n <= 2){
        printf("n should be greater than 2\n");
        exit(1);
    }
    if (n % 2 == 1)
        printMagicSquare(oddMagicSquare(n), n);
    else{
        if (n % 4 == 2)
            printMagicSquare(singlyEvenMagicSquare(n), n);
        else if(n % 4 == 0)
            printMagicSquare(doublyEvenMagicSquare(n), n);
    }
    exit(0);
}
```

**Explanation:** Magic square creation has three different types of inputs.

- Odd number (2n + 1)
- Singly even number (4n + 2)
- Doubly even number (4n)

There are dedicated child process for computing each index of the magic square.

**Output:**



```
thegamingbot@pop-os:~/Downloads/sem-5/OS/lab4$ ./8
Enter a number: 6
  35    1    6   26   19   24
   3   32    7   21   23   25
  31    9    2   22   27   20
   8   28   33   17   10   15
  30    5   34   12   14   16
   4   36   29   13   18   11

Magic constant: 111
thegamingbot@pop-os:~/Downloads/sem-5/OS/lab4$ ./8
Enter a number: 7
  30   39   48    1   10   19   28
  38   47    7    9   18   27   29
  46    6    8   17   26   35   37
   5   14   16   25   34   36   45
  13   15   24   33   42   44    4
  21   23   32   41   43    3   12
  22   31   40   49    2   11   20

Magic constant: 175
thegamingbot@pop-os:~/Downloads/sem-5/OS/lab4$ ./8
Enter a number: 8
  64   63    3    4    5    6   58   57
  56   55   11   12   13   14   50   49
  17   18   46   45   44   43   23   24
  25   26   38   37   36   35   31   32
  33   34   30   29   28   27   39   40
  41   42   22   21   20   19   47   48
  16   15   51   52   53   54   10    9
   8    7   59   60   61   62    2    1

Magic constant: 260
thegamingbot@pop-os:~/Downloads/sem-5/OS/lab4$
```