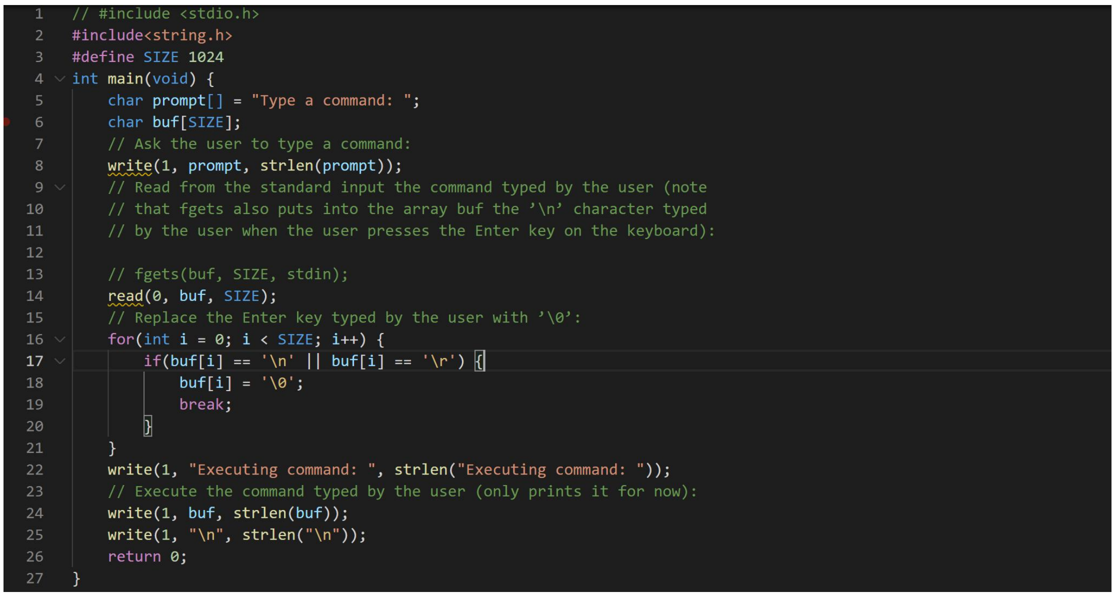
Question 1

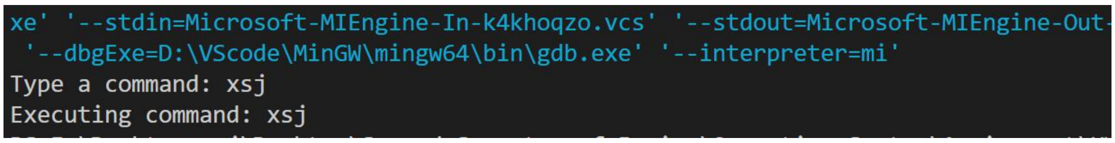
How to solve problem:

1. Replace "fgets" with "read" function and replace "printf" with "write" function where "fgets" and "printf" need to the library "#include $<$ stdio.h $>$ ", thus we should remove the library line from the beginning of the program.
2. The function "$write$": int write (int handle, void $*$ buf, int len). The $handle$ is the file handle for which the file pointer is to be obtained and $write$ is 1. buf is the content to be written and the len is the length of the file to be written; The function "$read$": int read (int handle, void $*$ buf, int len). The $handle$ is the file handle for which the file pointer is to be obtained and $read$ is 0. buf is the buffer to hold for what to read and the len is the length of the file to be read.
3. All the $printf$ should be replaced by $write$ including the "Type a command: " and "Executing command: " as reminder.
4. Remember to add the $\backslash n$ in the end of the write buf message. It also uses the $write$ function.



```c
// #include <stdio.h>
#include<string.h>
#define SIZE 1024
int main(void) {
    char prompt[] = "Type a command: ";
    char buf[SIZE];
    // Ask the user to type a command:
    write(1, prompt, strlen(prompt));
    // Read from the standard input the command typed by the user (note
    // that fgets also puts into the array buf the '\n' character typed
    // by the user when the user presses the Enter key on the keyboard):

    // fgets(buf, SIZE, stdin);
    read(0, buf, SIZE);
    // Replace the Enter key typed by the user with '\0':
    for(int i = 0; i < SIZE; i++) {
        if(buf[i] == '\n' || buf[i] == '\r') {
            buf[i] = '\0';
            break;
        }
    }
    write(1, "Executing command: ", strlen("Executing command: "));
    // Execute the command typed by the user (only prints it for now):
    write(1, buf, strlen(buf));
    write(1, "\n", strlen("\n"));
    return 0;
}
```

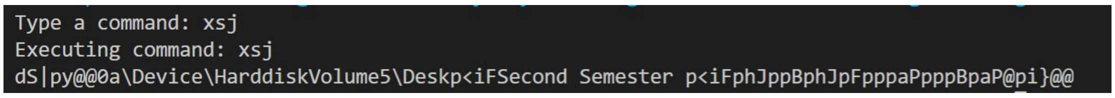
And the run results as follow:



```
xe' '--stdin=Microsoft-MIEngine-In-k4khoqzo.vcs' '--stdout=Microsoft-MIEngine-Out-
 '--dbgExe=D:\VScode\MinGW\mingw64\bin\gdb.exe' '--interpreter=mi'
Type a command: xsj
Executing command: xsj
```

Problem encountered and solution:

In the function read, the parameter len can be $SIZE$ 1024 which means up that to 1024 characters can be read. But in the $write$ function, the parameter cannot be $SIZE$ 1024. That's since buf is the buffer address, it assigns the space to you as much as the length you write. So we should use the function $strlen()$ to get the size of characters you write. Or you will get the other address information in the buffer, which are usually a lot of garbled code. Show as follow:

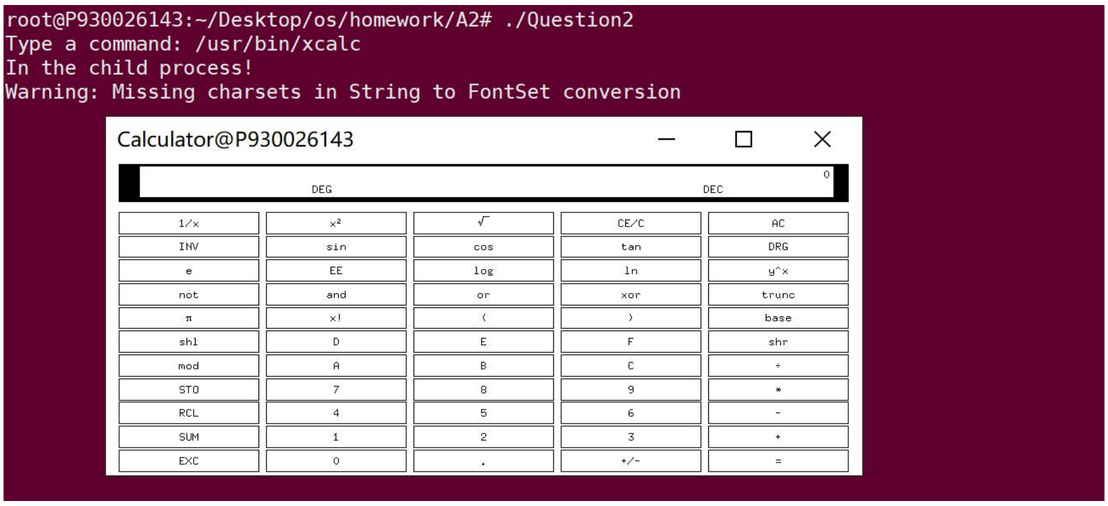


```
Type a command: xsj
Executing command: xsj
dS|py@@0a\Device\HarddiskVolume5\Deskp<iFSecond Semester p<iFphJppBphJpFpppaPpppBpaP@pi}@@
```

Question 2

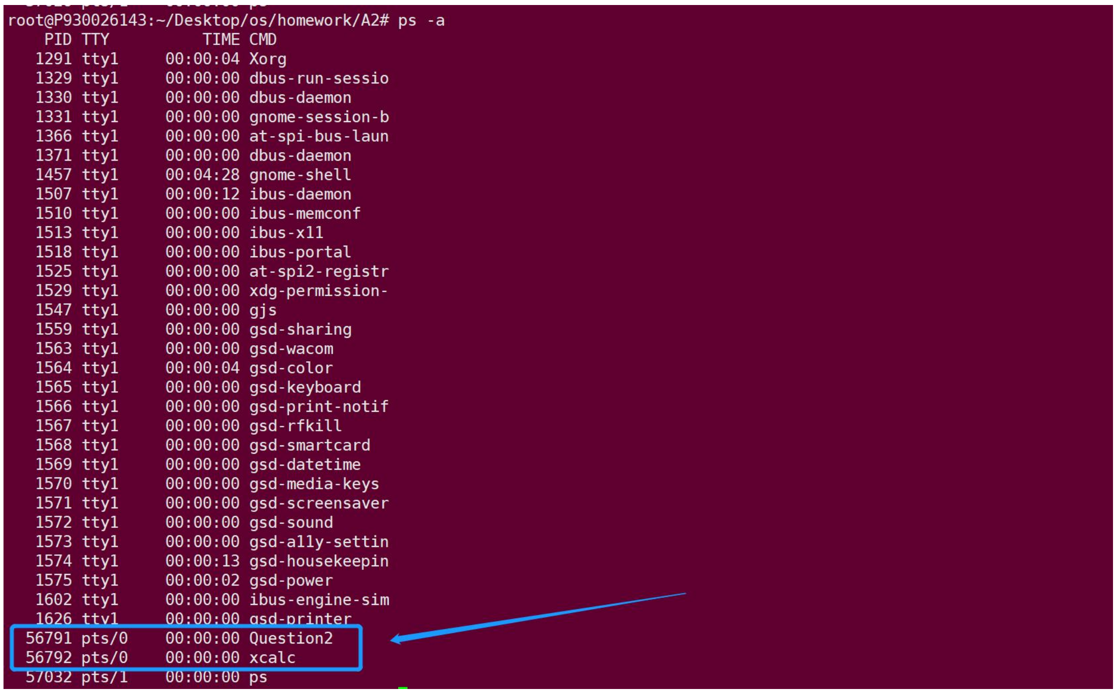
How to solve problem:

1. Create the child process by function $fork()$ below the read the $read$ buffer and Replace the Enter key typed by the user with $'\backslash 0'$. And we should add the judgment statement to check whether the child process is created successfully as well as distinguish the child process and the parent process. If $pid < 0$, it means that it fails to create the child process. If $pid = 0$, it means that it is the child process which the code in this part cannot be execute by the parent process. If $pid > 0$, it means that it is the parent process which the code in this part cannot be execute by the child process.
2. In the $pid = 0$ case, we should be going to use what the user typed as the name of the program to execute in the new process, we can use the function $execpl()$ to do that. In the $pid > 0$ case, we should make the parent process wait for the child process finish, we can use the $wait()$ function to do that. Definition: $pid\_t\ wait\ (int\ *\ status);$
3. $wait()$ function is used to temporarily stop the execution of the process until a signal arrives or the child process ends. The end status value of the child process is returned by the status parameter. Thus we should set the variable to represent status.
   It should add the header file $\#include\ < sys/wait.h >$.
4. $execpl()$ function is used to find the file from the PATH environment variable and execute. Definition: $int\ execlp(const\ char\ *\ file, const\ char\ *\ arg, \ldots);$ The last argument must be terminated by a NULL pointer. The function does not return on success, or -1 on failure. In other words, if the input file does not exist, it will become nothing to execute and nothing to happen. It should add the header file $\#include\ < unistd.h >$.
5. Then we use $gcc - o\ Question2\ Question2.c$ to compile the $.c$ file and open by the $./Question2$, just like follow:



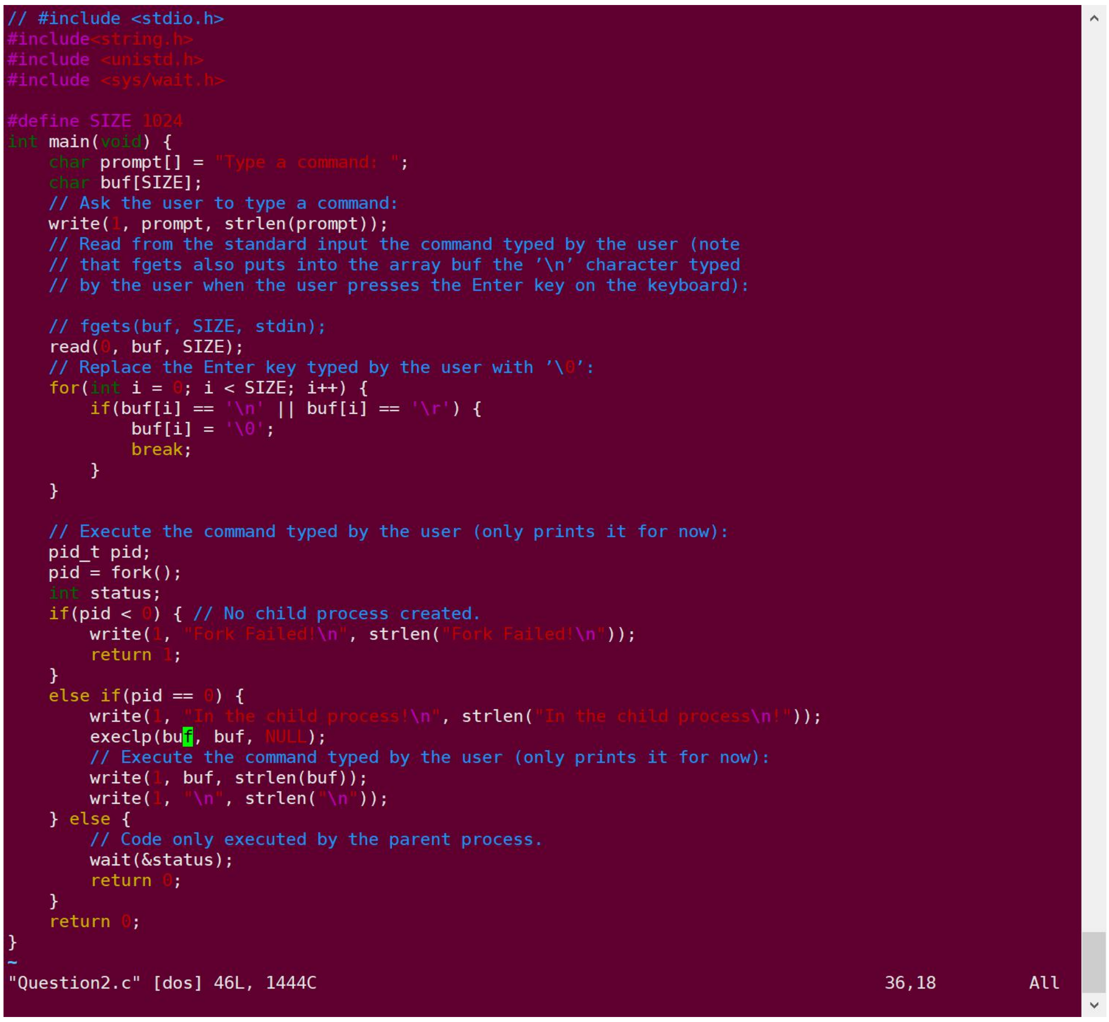
```
root@P930026143:~/Desktop/os/homework/A2# ./Question2
Type a command: /usr/bin/xcalc
In the child process!
Warning: Missing charsets in String to FontSet conversion
```

We can find that there is a calculator comes to the screen and we can check the pts by the command $ps - a$, $xcalc$ task on the child process can be seen.



```
root@P930026143:~/Desktop/os/homework/A2# ps -a
    PID TTY          TIME CMD
   1291 tty1     00:00:04 Xorg
   1329 tty1     00:00:00 dbus-run-sessio
   1330 tty1     00:00:00 dbus-daemon
   1331 tty1     00:00:00 gnome-session-b
   1366 tty1     00:00:00 at-spi-bus-laun
   1371 tty1     00:00:00 dbus-daemon
   1457 tty1     00:04:28 gnome-shell
   1507 tty1     00:00:12 ibus-daemon
   1510 tty1     00:00:00 ibus-memconf
   1513 tty1     00:00:00 ibus-x11
   1518 tty1     00:00:00 ibus-portal
   1525 tty1     00:00:00 at-spi2-registr
   1529 tty1     00:00:00 xdg-permission-
   1547 tty1     00:00:00 gjs
   1559 tty1     00:00:00 gsd-sharing
   1563 tty1     00:00:00 gsd-wacom
   1564 tty1     00:00:04 gsd-color
   1565 tty1     00:00:00 gsd-keyboard
   1566 tty1     00:00:00 gsd-print-notif
   1567 tty1     00:00:00 gsd-rfkill
   1568 tty1     00:00:00 gsd-smartcard
   1569 tty1     00:00:00 gsd-datetime
   1570 tty1     00:00:00 gsd-media-keys
   1571 tty1     00:00:00 gsd-screensaver
   1572 tty1     00:00:00 gsd-sound
   1573 tty1     00:00:00 gsd-ally-settin
   1574 tty1     00:00:13 gsd-housekeepin
   1575 tty1     00:00:02 gsd-power
   1602 tty1     00:00:00 ibus-engine-sim
   1626 tty1     00:00:00 gsd-printer
  56791 pts/0    00:00:00 Question2
  56792 pts/0    00:00:00 xcalc
  57032 pts/1    00:00:00 ps
```

The $Question2.c$ is as follows:



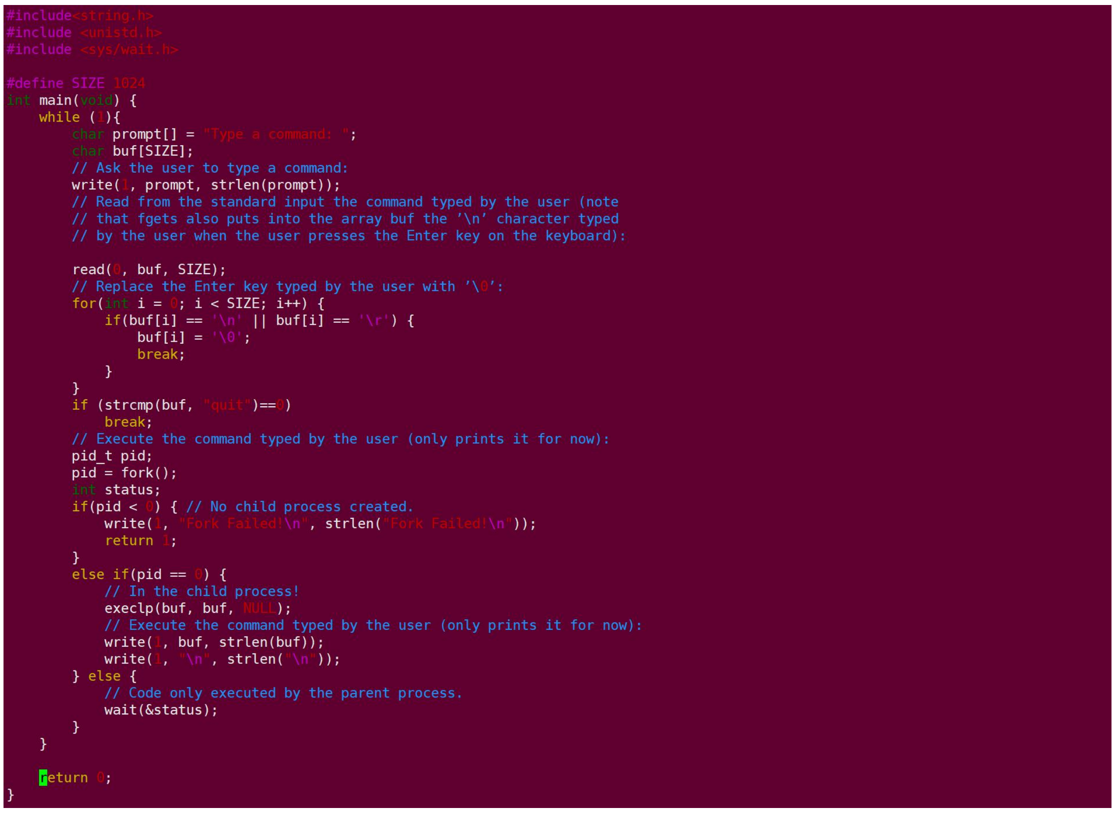
```
// #include <stdio.h>
#include<string.h>
#include <unistd.h>
#include <sys/wait.h>

#define SIZE 1024
int main(void) {
    char prompt[] = "Type a command: ";
    char buf[SIZE];
    // Ask the user to type a command:
    write(1, prompt, strlen(prompt));
    // Read from the standard input the command typed by the user (note
    // that fgets also puts into the array buf the ’\n’ character typed
    // by the user when the user presses the Enter key on the keyboard):

    // fgets(buf, SIZE, stdin);
    read(0, buf, SIZE);
    // Replace the Enter key typed by the user with ’\0’:
    for(int i = 0; i < SIZE; i++) {
        if(buf[i] == '\n' || buf[i] == '\r') {
            buf[i] = '\0';
            break;
        }
    }

    // Execute the command typed by the user (only prints it for now):
    pid_t pid;
    pid = fork();
    int status;
    if(pid < 0) { // No child process created.
        write(1, "Fork Failed!\n", strlen("Fork Failed!\n"));
        return 1;
    }
    else if(pid == 0) {
        write(1, "In the child process!\n", strlen("In the child process\n!"));
        execlp(buf, buf, NULL);
        // Execute the command typed by the user (only prints it for now):
        write(1, buf, strlen(buf));
        write(1, "\n", strlen("\n"));
    } else {
        // Code only executed by the parent process.
        wait(&status);
        return 0;
    }
    return 0;
}
~
"Question2.c" [dos] 46L, 1444C                                     36,18         All
```

Question 3

How to solve problem:

1. Based on the question 2, we should add a loop statement so that so that your program asks the user for input, creates a new child process, waits for the child process to end, then asks the user for input again and again.
2. We can use a while statement but not for statement because there is not a stated time of loop. And then we should add an end condition, when the buffer we read is equal to "$quit$", it should escape out of the loop before create the child process and finish the parent process.
3.

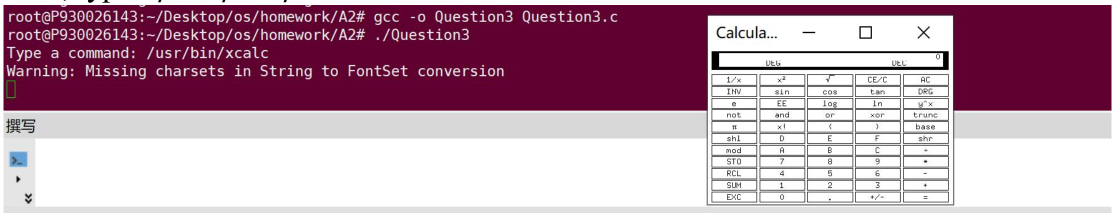
The $.c$ file code is as follow:



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

#define SIZE 1024
int main(void) {
    while (1){
        char prompt[] = "Type a command: ";
        char buf[SIZE];
        // Ask the user to type a command:
        write(1, prompt, strlen(prompt));
        // Read from the standard input the command typed by the user (note
        // that fgets also puts into the array buf the '\n' character typed
        // by the user when the user presses the Enter key on the keyboard):

        read(0, buf, SIZE);
        // Replace the Enter key typed by the user with '\0':
        for(int i = 0; i < SIZE; i++) {
            if(buf[i] == '\n' || buf[i] == '\r') {
                buf[i] = '\0';
                break;
            }
        }
        if (strcmp(buf, "quit")==0)
            break;
        // Execute the command typed by the user (only prints it for now):
        pid_t pid;
        pid = fork();
        int status;
        if(pid < 0) { // No child process created.
            write(1, "Fork Failed!\n", strlen("Fork Failed!\n"));
            return 1;
        }
        else if(pid == 0) {
            // In the child process!
            execlp(buf, buf, NULL);
            // Execute the command typed by the user (only prints it for now):
            write(1, buf, strlen(buf));
            write(1, "\n", strlen("\n"));
        } else {
            // Code only executed by the parent process.
            wait(&status);
        }
    }

    return 0;
}
```

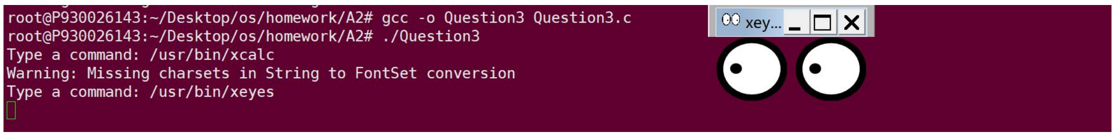
Then we use $gcc - o\ Question3\ Question3.c$ to compile the $.c$ file and open by the $./Question3$, just like follow:

First, typed $/usr/bin/xcalc$:



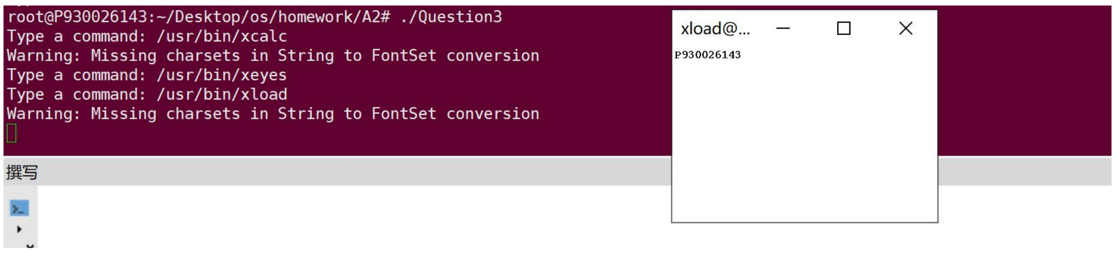
```
root@P930026143:~/Desktop/os/homework/A2# gcc -o Question3 Question3.c
root@P930026143:~/Desktop/os/homework/A2# ./Question3
Type a command: /usr/bin/xcalc
Warning: Missing charsets in String to FontSet conversion
```

Second, close the calculator and continue to type $/usr/bin/xeyes$, we will find a pair of eyes looking at the mouse arrow:



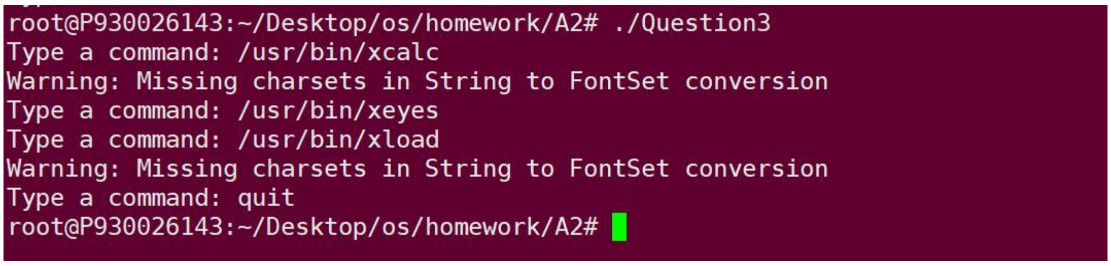
```
root@P930026143:~/Desktop/os/homework/A2# gcc -o Question3 Question3.c
root@P930026143:~/Desktop/os/homework/A2# ./Question3
Type a command: /usr/bin/xcalc
Warning: Missing charsets in String to FontSet conversion
Type a command: /usr/bin/xeyes
```

Third, close the eyes and continue to type $/usr/bin/xload$, we will see a textbox which contain the hostname pop out to the screen:



```
root@P930026143:~/Desktop/os/homework/A2# ./Question3
Type a command: /usr/bin/xcalc
Warning: Missing charsets in String to FontSet conversion
Type a command: /usr/bin/xeyes
Type a command: /usr/bin/xload
Warning: Missing charsets in String to FontSet conversion
```

In the end, close the textbox and continue to type $quit$, we will find that the program exits successfully:



```
root@P930026143:~/Desktop/os/homework/A2# ./Question3
Type a command: /usr/bin/xcalc
Warning: Missing charsets in String to FontSet conversion
Type a command: /usr/bin/xeyes
Type a command: /usr/bin/xload
Warning: Missing charsets in String to FontSet conversion
Type a command: quit
root@P930026143:~/Desktop/os/homework/A2# 
```