

## Lesson 35

Objectives: -

- Write a program to write Hello World added with colours
- Write a program to sets up a timer to send a SIGALRM signal at a specified interval.

# "Hello, world!", ncurses style

## Function

This program implements the good old "Hello, world!" application using ncurses. It clears the screen, displays the phrase in the centre, waits for a few seconds, then exits.

## Programming Issues

All programs using ncurses must call `initscr()` before using any of the library functions. This function can fail, so we always check the return value.

We output the "Hello, world!" phrase using `mvaddstr`, or "move to the specified position and add a string there". Note that the coordinates are specified in the order of **row, column**, or **y, x**, and not the other way around.

Note that the above function (along with most other library functions) do not actually directly modify the display; they just modify data structures held in memory. To actually reflect our changes on the screen, we must call one of the `refresh()` functions.

We call the `sleep()` function to sleep for a few seconds, to show the full screen display for a moment before our command prompt returns. Note that `sleep()` here is a UNIX function and defined in `<unistd.h>`; if you are using a version of ncurses ported to a non-UNIX platform, your compiler may not have this header file or this function, though it will almost certainly have something similar.

Finally, we must clean up after ourselves, as we want the terminal to be in a useable state after our program exits. First, we must delete all of our windows using `delwin()`, including the one returned by `initscr()`. Then, we call `endwin()` and finally `refresh()` again, before we exit safely.

Note that when you are learning ncurses development, you will almost certainly foul up your terminal with alarming regularity. On UNIX, you can type `stty sane` at the command prompt to return the terminal to a useable state without needing to log out.

## Usage

You will need to link to the ncurses library in order to use the functions in it. On UNIX, this is typically done using the `-l` compiler switch, e.g.:

```
[paul@localhost src]$ cc -o curhello curhello.c -lncurses
```

On other systems, consult your compiler's documentation, or the ncurses documentation for more information.

Also note that in most of these ncurses examples, we assume a terminal with 80 columns and 25 rows. If your terminal has different dimensions, you may need to adapt the examples to suit.

## curhello.c

```
/*
```

```

CURHELLO.C
=====
(c) Copyright Paul Griffiths 1999
Email: mail@paulgriffiths.net

"Hello, world!", ncurses style.

*/

#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>           /* for
sleep() */
#include <curses.h>

int main(void) {

    WINDOW * mainwin;

    /* Initialize ncurses */

    if ( (mainwin = initscr()) == NULL ) {
        fprintf(stderr, "Error initialising
ncurses.\n");
        exit(EXIT_FAILURE);
    }

    /* Display "Hello, world!" in the centre of
the
screen, call refresh() to show our changes,
and
sleep() for a few seconds to get the full
screen effect */

    mvaddstr(13, 33, "Hello, world!");
    refresh();
    sleep(3);

    /* Clean up after ourselves */

    delwin(mainwin);
    endwin();
    refresh();

    return EXIT_SUCCESS;
}

```

# "Hello, world!" - now with added colour!

## Function

This program extends "Hello, world!" by outputting the phrase a number of times, with different foreground and background colors.

## Programming Issues

We must call `start_color()` to initialize colour operations in `ncurses`, before using any of the colour handling library functions. After doing this, we can call `has_color()` to find out whether the terminal in use supports colour at all.

To change color, we need to call `color_set()` and specify a "color pair" to use. A color pair simply consists of a foreground and background color. `ncurses` has a limited number of color pairs available (you can find out how many by checking the value of the constant `COLOR_PAIRS`).

In this program, we set up a number of color pairs using the `init_pair()` function together with the predefined color constants `ncurses` supplies (e.g. `COLOR_BLACK`, `COLOR_GREEN`). Once we have defined these, we loop through them, using `color_set()` to switch between pairs, printing the "Hello, world!" phrase in each one as we go.

## curhell2.c

```
/*  
  
CURHELL2.C  
=====
```

(c) Copyright Paul Griffiths 1999  
Email: mail@paulgriffiths.net

```
"Hello, world!", ncurses style (now in colour!)
```

```

*/

#include <stdlib.h>
#include <stdio.h>
#include <unistd.h> /* for
sleep() */
#include <curses.h>

int main(void) {

    WINDOW * mainwin;

    /* Initialize ncurses */

    if ( (mainwin = initscr()) == NULL ) {
        fprintf(stderr, "Error initialising
ncurses.\n");
        exit(EXIT_FAILURE);
    }

    start_color(); /*
Initialize colours */

    /* Print message */

    mvaddstr(6, 32, " Hello, world! ");

    /* Make sure we are able to do what we want.
If
has_colors() returns FALSE, we cannot use
colours.
COLOR_PAIRS is the maximum number of colour
pairs
we can use. We use 13 in this program, so we
check
to make sure we have enough available.
*/

    if ( has_colors() && COLOR_PAIRS >= 13 ) {

        int n = 1;

        /* Initialize a bunch of colour pairs,
where:

            init_pair(pair number, foreground,
background);

            specifies the pair.
*/

```

```

init_pair(1, COLOR_RED, COLOR_BLACK);
init_pair(2, COLOR_GREEN, COLOR_BLACK);
init_pair(3, COLOR_YELLOW, COLOR_BLACK);
init_pair(4, COLOR_BLUE, COLOR_BLACK);
init_pair(5, COLOR_MAGENTA, COLOR_BLACK);
init_pair(6, COLOR_CYAN, COLOR_BLACK);
init_pair(7, COLOR_BLUE, COLOR_WHITE);
init_pair(8, COLOR_WHITE, COLOR_RED);
init_pair(9, COLOR_BLACK, COLOR_GREEN);
init_pair(10, COLOR_BLUE, COLOR_YELLOW);
init_pair(11, COLOR_WHITE, COLOR_BLUE);
init_pair(12, COLOR_WHITE, COLOR_MAGENTA);
init_pair(13, COLOR_BLACK, COLOR_CYAN);

/* Use them to print of bunch of "Hello,
world!"s */

while ( n <= 13 ) {
    color_set(n, NULL);
    mvaddstr(6 + n, 32, " Hello, world! ");
    n++;
}

/* Refresh the screen and sleep for a
while to get the full screen effect */

refresh();
sleep(3);

/* Clean up after ourselves */

delwin(mainwin);
endwin();
refresh();

return EXIT_SUCCESS;
}

```

---

# Basic ncurses keyboard input

## Function

Because it cannot assume much about the environment in which it will be run, ANSI C provides no mechanism for unbuffered, direct keyboard input (i.e. it cannot retrieve a single character, and cannot interpret cursor keys or other non-character keys). This makes handling interactive input somewhat difficult. `ncurses` provides some platform specific methods of doing this.

This program simply accepts keypresses from the user and immediately prints the hexadecimal key code, and a textual description of the key pressed.

## Programming Issues

After initialisation, the program calls `noecho()` to prevent the key being echoed to the screen, and `keypad()` to enable the cursor and other keys to be detected by the program. The `getch()` function is then used to retrieve each character at a time, until the user hits 'q'.

In our `intprtkey()` function, we return a string describing the key pressed. If the key was a printable character, we just return a pointer to a string containing that character only. Otherwise, we use an array of structs to hold the key code of a selection of other keys (as defined in `<ncurses.h>`) along with their short descriptions.

## curin1.c

```
/*  
  
    CURIN1.C  
    =====  
    (c) Copyright Paul Griffiths 1999  
    Email: mail@paulgriffiths.net  
  
    Demonstrating basic ncurses single key input.  
  
*/  
  
#include <stdlib.h>  
#include <ctype.h>  
#include <ncurses.h>
```

```

char * intprtkey(int ch);

int main(void) {

    WINDOW * mainwin;
    int ch;

    /* Initialize ncurses */

    if ( (mainwin = initscr()) == NULL ) {
        fprintf(stderr, "Error initializing
ncurses.\n");
        exit(EXIT_FAILURE);
    }

    noecho(); /* Turn off key
echoing */
    keypad(mainwin, TRUE); /* Enable the
keypad for non-char keys */

    /* Print a prompt and refresh() the screen
*/

    mvaddstr(5, 10, "Press a key ('q' to
quit)...");
    mvprintw(7, 10, "You pressed: ");
    refresh();

    /* Loop until user presses 'q' */

    while ( (ch = getch()) != 'q' ) {

        /* Delete the old response line, and print
a new one */

        deleteln();
        mvprintw(7, 10, "You pressed: 0x%x (%s)",
ch, intprtkey(ch));
        refresh();
    }

    /* Clean up after ourselves */

    delwin(mainwin);
    endwin();
    refresh();

    return EXIT_SUCCESS;
}

```



```

                                { KEY_F(9),
"Function key 9" },
                                { KEY_F(10),
"Function key 10" },
                                { KEY_F(11),
"Function key 11" },
                                { KEY_F(12),
"Function key 12" },
                                { -1,
"<unsupported>" }
};
static char keych[2] = {0};

if ( isprint(ch) && !(ch & KEY_CODE_YES)) {

    /* If a printable character */

    keych[0] = ch;
    return keych;
}
else {

    /* Non-printable, so loop through our array
of structs */

    int n = 0;

    do {
        if ( keys[n].code == ch )
            return keys[n].name;
        n++;
    } while ( keys[n].code != -1 );

    return keys[n].name;
}

return NULL;          /* We shouldn't get here
*/
}

```

# Basic ncurses windows

## Function

This program introduces the window operations ncurses makes possible. It simply shows a window in the centre of the screen, and allows the user to move it around using the cursor keys, and the HOME and END keys.

## Programming Issues

After initializing `ncurses`, switching off `echo` and enabling the keypad, we create a new window using `subwin()`, specifying our main window as the parent. We give this window a border using the `box()` function, and add some text to it using `mvwaddstr()`. The coordinates passed to `mvwaddstr()` are now relative to the origin of our new window, and not to the screen.

We then use `getch()` to get some keypresses from the user. We catch the cursor keys, HOME and END, and modify the coordinates of our window accordingly. Then, we call `mvwin()` to move it. Note that we do not need to call `refresh()` after moving our window.

Finally we clean up. Notice that we call `delwin()` on our subwindow before calling it on the parent.

### curwin1.c

```
/*
   CURWIN1.C
   =====
   (c) Copyright Paul Griffiths 1999
   Email: mail@paulgriffiths.net

   Moving windows with ncurses.
*/

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

int main(void) {

    WINDOW * mainwin, * childwin;
    int      ch;

    /* Set the dimensions and initial
       position for our child window */

    int      width = 23, height = 7;
```

```

int      rows = 25, cols = 80;
int      x = (cols - width) / 2;
int      y = (rows - height) / 2;

/* Initialize ncurses */

if ( (mainwin = initscr()) == NULL ) {
    fprintf(stderr, "Error initialising
ncurses.\n");
    exit(EXIT_FAILURE);
}

/* Switch of echoing and enable keypad (for
arrow keys) */

noecho();
keypad(mainwin, TRUE);

/* Make our child window, and add
a border and some text to it. */

childwin = subwin(mainwin, height, width, y,
x);
box(childwin, 0, 0);
mvwaddstr(childwin, 1, 4, "Move the window");
mvwaddstr(childwin, 2, 2, "with the arrow
keys");
mvwaddstr(childwin, 3, 6, "or HOME/END");
mvwaddstr(childwin, 5, 3, "Press 'q' to
quit");

refresh();

/* Loop until user hits 'q' to quit */

while ( (ch = getch()) != 'q' ) {

    switch ( ch ) {

        case KEY_UP:
            if ( y > 0 )
                --y;
            break;

        case KEY_DOWN:
            if ( y < (rows - height) )
                ++y;
            break;

        case KEY_LEFT:
            if ( x > 0 )
                --x;
            break;
    }
}

```

```

        case KEY_RIGHT:
            if ( x < (cols - width) )
                ++x;
            break;

        case KEY_HOME:
            x = 0;
            y = 0;
            break;

        case KEY_END:
            x = (cols - width);
            y = (rows - height);
            break;

    }

    mvwin(childwin, y, x);
}

/* Clean up after ourselves */

delwin(childwin);
delwin(mainwin);
endwin();
refresh();

return EXIT_SUCCESS;
}

```

# Worms (with ncurses)

## Function

This program implements a version of the "Worms" arcade game, using ncurses.

## Programming Issues

### Game logic

The object of the game is to:

- direct the worm to collect worm food, whilst
- avoiding hitting anything else (including yourself)

The controls are:

- Left arrow or 'g' to direct the worm left
- Right arrow or 'j' to direct the worm right
- Up arrow or 'y' to direct the worm up
- Down arrow or 'n' to direct the worm down
- 'q' to quit

The program sets up a timer to send a SIGALRM signal at a specified interval. Every time this signal is received, the worm is moved. The current direction of the worm is stored in a global variable.

The worm is represented as a linked list, typedef'd to WORM. Every time the SIGALRM signal is received:

- The next position of the worm's head is calculated using the stored direction
- That position is checked to see whether the worm has run into a wall, itself, or some food.
- If the worm has run into itself or a wall, the game ends.
- If not, a new node is added to the WORM list, and displayed on the screen.
- If the worm has run into some food, the node at the back of the list is left alone, so the worm becomes one node longer every time food is collected. Otherwise, the first worm node is deleted, to keep the moving worm at the same length.
- One point is gained every time the worm moves (to reward the player for simply not hitting anything), and ten points are gained for eating some wormfood.

While this signal handling is happening in the background, the main program calls `getch()` to get input from the player. If any of the direction keys are pressed, the global direction variable is updated. The actual change in direction will occur the next time the worm moves, i.e. when SIGALRM is next received.

## Curses

As usual, `ncurses` is initialised at the start, and then cleaned up again whenever the game ends. The `box()`

function is used to make the screen border, and `addch()` is used to display the worm. When deleting previous WORM nodes, `addch()` is used to simply output a space character over the top of the previous nodes, to remove them.

When placing a new piece of food, the game needs to know how many rows and columns the terminal has, in order that food should not be placed outside the game arena. To do this, `ioctl()` is used, specifying `TIOCGWINSZ` as the request to get the terminal size.

## Signal handling

Using `ncurses` presents us with a problem when the user quits the game other than through the normal route (i.e. by hitting 'q'). For instance, if the user hits CTRL-C to abort the program, and we do nothing, then `ncurses` will not clean up the terminal after itself, and the terminal may then be garbled.

To avoid this, we catch `SIGTERM` and `SIGINT`. When these signals are received, we clean up `ncurses` nicely, and then quit as normal.

A similar thing would have to be implemented when the game is temporarily stopped, then restarted (e.g. by hitting CTRL-Z). We would have to clean up `ncurses` when `SIGTSTP` is received, and reinitialise it and redraw the screen when `SIGCONT` is subsequently received. However, for simplicity, in this game we simply ignore `SIGTSTP` signals, effectively disabling CTRL-Z.

## Usage

There are no command line arguments. Simply type `./worms` or whatever your OS requires to run the game.

## main.c

```
/*
```

```

MAIN.C
=====
(c) Copyright Paul Griffiths 2000
Email: mail@paulgriffiths.net

Worms game using ncurses.

*/

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

#include <signal.h>
#include <sys/time.h>
#include <ncurses.h>

#include "worms.h"
#include "helper.h"

#define TIMESTEP 200000

void SetTimer(void);
void SetSignals(void);

WINDOW * mainwin;
int oldcur;

int main(void) {

    /* Seed RNG, set timer and register signal
    handlers */

    srand( (unsigned) time(NULL) );
    SetTimer();
    SetSignals();

    /* Initialize ncurses */

    if ( (mainwin = initscr()) == NULL ) {
        perror("error initialising ncurses");
        exit(EXIT_FAILURE);
    }

    noecho();
    keypad(mainwin, TRUE);
    oldcur = curs_set(0);

    /* Initialize worm and draw it */

```

```

InitWorm();
Draw();

/* Loop and get user input */
while ( 1 ) {
    int key = getch();

    switch ( key ) {

        case KEY_UP:
        case 'Y':
        case 'y':
            ChangeDir(UP);
            break;

        case KEY_DOWN:
        case 'N':
        case 'n':
            ChangeDir(DOWN);
            break;

        case KEY_LEFT:
        case 'G':
        case 'g':
            ChangeDir(LEFT);
            break;

        case KEY_RIGHT:
        case 'J':
        case 'j':
            ChangeDir(RIGHT);
            break;

        case 'Q':
        case 'q':
            Quit(USER_QUIT);
            break;

    }

}

/* We never get here */
return EXIT_SUCCESS;
}

/* Sets up the game timer */
void SetTimer(void) {

    struct itimerval it;

```

```

/* Clear itimer struct members */

timerclear(&it.it_interval);
timerclear(&it.it_value);

/* Set timer */

it.it_interval.tv_usec = TIMESTEP;
it.it_value.tv_usec    = TIMESTEP;
setitimer(ITIMER_REAL, &it, NULL);
}

/* Sets up signal handlers we need */
void SetSignals(void) {

    struct sigaction sa;

    /* Fill in sigaction struct */

    sa.sa_handler = handler;
    sa.sa_flags   = 0;
    sigemptyset(&sa.sa_mask);

    /* Set signal handlers */

    sigaction(SIGTERM, &sa, NULL);
    sigaction(SIGINT,  &sa, NULL);
    sigaction(SIGALRM, &sa, NULL);

    /* Ignore SIGTSTP */

    sa.sa_handler = SIG_IGN;
    sigaction(SIGTSTP, &sa, NULL);
}

```

---

## worms.h

```

/*

WORMS.H
=====
(c) Copyright Paul Griffiths 2000
Email: mail@paulgriffiths.net

```

```

        Interface to worms functions
    */

    #ifndef PG_WORMS_H
    #define PG_WORMS_H

    /* Structures and typedefs */

    struct worm_part {
        struct worm_part * next;
        int x;
        int y;
    };

    typedef struct worm_part WORM;

    /* Macros */

    #define DOWN 1
    #define UP 2
    #define LEFT 3
    #define RIGHT 4

    #define WORMBIT 'O'
    #define EMPTY ' '
    #define WORMFOOD 'X'

    /* Function prototypes */

    void InitWorm(void);
    void MoveWorm(void);
    void Draw(void);
    void FreeWorm(void);

    #endif /* PG_WORMS_H */

```

---

## worms.c

```

/*
    WORMS.C

```

=====

(c) Copyright Paul Griffiths 2000  
Email: mail@paulgriffiths.net

Implementation of worms functions.

\*/

#include <stdlib.h>

#include <curses.h>

#include "worms.h"

#include "helper.h"

#define MOVESCORE 1

#define FOODSCORE 10

#define WORMSIZE 8

static WORM \* worm;  
static int dir = DOWN;  
static int rows, cols;  
int score = 0;

void PlaceFood(void);  
void Draw(void);

/\* Initialises the worm, and draws it for the  
first time \*/

void InitWorm(void) {

    WORM \* temp;  
    int x = 1, y = 1, i;

    /\* Allocate memory for worm \*/

    for ( i = 0; i < WORMSIZE; i++ ) {  
        if ( i == 0 ) {

            /\* If this is the first worm part  
\*/

            if ( (worm = malloc(sizeof(WORM))) ==  
NULL )

                Error\_Quit("couldn't allocate  
memory for worm");

                temp = worm;  
            }

            else {

                /\* If this is not the first worm  
part \*/

```

        if ( (temp->next =
malloc(sizeof(WORM)) ) == NULL )
            Error_Quit("couldn't allocate
memory for worm");
        temp = temp->next;
    }

    /* Set coordinates for this worm part
*/

    temp->x = x;
    temp->y = y++;
}
temp->next = NULL;

/* Get window size */

GetTermSize(&rows, &cols);
}

/* Draws game arena */
void Draw(void) {
    WORM * temp = worm;

    /* Draw border around arena */
    box(stdscr, 0, 0);

    /* Draw the worm */
    while ( temp ) {
        move(temp->y, temp->x);
        addch(WORMBIT);
        temp = temp->next;
    }

    /* Draw initial worm food */
    PlaceFood();
}

/* Moves the worm */
void MoveWorm(void) {
    WORM * temp = worm;
    int x, y, ch;

```

```

/* Move to last worm bit */
while ( temp->next != NULL )
    temp = temp->next;

/* Make a new worm bit */

if ( (temp->next = malloc(sizeof(WORM))) ==
NULL )
    Error_Quit("couldn't allocate memory in
MoveWorm()");

/* Determine position of new worm bit */

switch ( dir ) {
case DOWN:
    x = temp->x;
    y = temp->y + 1;
    break;

case UP:
    x = temp->x;
    y = temp->y - 1;
    break;

case LEFT:
    x = temp->x - 1;
    y = temp->y;
    break;

case RIGHT:
    x = temp->x + 1;
    y = temp->y;
    break;
}

/* Fill in new worm bit structure */

temp      = temp->next;
temp->next = NULL;
temp->x    = x;
temp->y    = y;

/* See if we hit anything */

move(y, x);
switch ( (ch = inch()) ) {
case EMPTY:

    score += MOVESCORE;

```

```

        /* Delete first worm bit */

        temp = worm->next;
        move(worm->y, worm->x);
        addch(EMPTY);
        free(worm);
        worm = temp;      /* Fallthrough */

    case WORMFOOD:

        /* Add new wormbit */

        move(y, x);
        addch(WORMBIT);
        if ( ch == WORMFOOD ) {

            /* Place some new food */

            PlaceFood();

            score += FOODSCORE;
        }

        refresh();
        break;

    case WORMBIT:
        Quit(HITSELF);

    default:
        Quit(HITWALL);
    }
}

/* Place food in the arena */

void PlaceFood(void) {

    int x, y;

    do {
        x = rand() % (cols - 3) + 1;
        y = rand() % (rows - 3) + 1;
        move(y, x);
    } while ( inch() != EMPTY );

    addch(WORMFOOD);
}

/* Changes the direction of travel */

void ChangeDir(int d) {

    WORM * temp = worm;

```

```

/* Move to last worm bit */
while ( temp->next != NULL )
    temp = temp->next;

/* Determine next position */
switch ( d ) {
case LEFT:
    if ( dir == RIGHT )
        return;
    move(temp->y, temp->x - 1);
    break;

case RIGHT:
    if ( dir == LEFT )
        return;
    move(temp->y, temp->x + 1);
    break;

case UP:
    if ( dir == DOWN )
        return;
    move(temp->y - 1, temp->x);
    break;

case DOWN:
    if ( dir == UP )
        return;
    move(temp->y + 1, temp->x);
    break;
}

dir = d;
}

/* Releases memory used by worm */
void FreeWorm(void) {
    WORM * temp = worm;

    while ( worm ) {
        temp = worm->next;
        free(worm);
        worm = temp;
    }
}

```

---

# helper.h

```
/*
HELPER.H
=====
(c) Copyright Paul Griffiths 2000
Email: mail@paulgriffiths.net

Interface to helper functions.
*/

#ifndef PG_HELPER_H
#define PG_HELPER_H

/* Macros */

#define USER_QUIT 1
#define HITSELF 2
#define HITWALL 3

/* Function prototypes */

void Error_Quit(char * msg);
void Quit(int reason);
void GetTermSize(int * rows, int * cols);
void handler(int signum);
void ChangeDir(int d);

#endif /* PG_HELPER_H */
```

---

# helper.c

```
/*
HELPER.C
=====
(c) Copyright Paul Griffiths 2000
Email: mail@paulgriffiths.net
```

```

    Helper functions.

*/

#include <stdlib.h>

#include <sys/ioctl.h>
#include <signal.h>
#include <curses.h>

#include "helper.h"
#include "worms.h"

/* Quit on error */

void Error_Quit(char * msg) {

    extern WINDOW * mainwin;
    extern int oldcur;

    /* Clean up nicely */

    delwin(mainwin);
    curs_set(oldcur);
    endwin();
    refresh();
    FreeWorm();

    /* Output error message and exit */

    perror(msg);
    exit(EXIT_FAILURE);
}

/* Quit successfully */

void Quit(int reason) {

    extern WINDOW * mainwin;
    extern int oldcur;
    extern int score;

    /* Clean up nicely */

    delwin(mainwin);
    curs_set(oldcur);
    endwin();
    refresh();
    FreeWorm();
}

```

```

/* Output farewell message */

switch ( reason ) {
case HITWALL:
    printf("\nYou hit a wall!\n");
    printf("Your score is %d\n", score);
    break;

case HITSELF:
    printf("\nYou ran into yourself!\n");
    printf("Your score is %d\n", score);
    break;

default:
    printf("\nGoodbye!\n");
    break;
}

exit(EXIT_SUCCESS);
}

/* Returns the x-y size of the terminal */
void GetTermSize(int * rows, int * cols) {

    struct winsize ws;

    /* Get terminal size */

    if ( ioctl(0, TIOCGWINSZ, &ws) < 0 ) {
        perror("couldn't get window size");
        exit(EXIT_FAILURE);
    }

    /* Update globals */

    *rows = ws.ws_row;
    *cols = ws.ws_col;
}

/* Signal handler */
void handler(int signum) {

    extern WINDOW * mainwin;
    extern int oldcur;

    /* Switch on signal number */

    switch ( signum ) {

    case SIGALRM:

```

```
        /* Received from the timer */

        MoveWorm();
        return;

    case SIGTERM:
    case SIGINT:

        /* Clean up nicely */

        delwin(mainwin);
        curs_set(oldcur);
        endwin();
        refresh();
        FreeWorm();
        exit(EXIT_SUCCESS);

    }
}
```

Questions: -

1. Write a program to write Hello World added with colours
2. Write a program to sets up a timer to send a SIGALRM signal at a specified interval.