STUDENT

DIT633-UYO-YMB

TENTAMEN

DIT632-DIT633 march 2025

Kurskod	--
Bedömningsform	--
Starttid	18.03.2025 14:00
Sluttid	18.03.2025 18:00
Bedömningsfrist	--
PDF skapad	26.11.2025 15:23
Skapad av	Lisa Lindén

Instructions for the exam

DIT 633 - Development of Embedded and Real-time systems

Welcome to this edtion of the DIT633 examination!

This exam should be an individual work for you. You are not allowed to use any outside help.

If you are allowed to use a compiler, there is a link to an online one, which will open in a separate window. You can test the code in the online compiler, but **you must remember to copy-paste it back to the exam**, otherwise your code will disappear once you close the window.

The same is true for TinkerCad, please remember to copy-paste the code from TinkerCad to the exam. Please note that the log-in to TinkerCad is different this time, so please read the instruction carefully (available in the TinkerCad question), especially regarding your **class nickname and class code:**

Please use the following procedure to log in to TinkerCad:
- go to log-in
- choose "Students with Class Code"
*- use the class code "*IBM V6R XJU*"*
- use join with nickname. Your anonymous code is your nickname - without the dashes and using small letters, e.g., DIT633-0001-ABC becomes dit6330001abc as your nickname

You are NOT allowed to access code that was not written during the exam (e.g., from your previous assignments). This will be considered plagiarism.

You are not allowed to copy code from your colleagues or any other external source.

Remember: In programming questions, if the code does not compile, you get 0 points for the question!

Grading scale:
50% correct - 3
65% correct - 4
85% correct - 5

Good luck!

/Miroslaw
031 772 1081

1 Reading pointers

Please choose the right interpretation of "X" in each of the statements:

int **x;

- x is a variable of type int
- x is a reference to the variable of type int
- x is a pointer to a pointer to a variable of type int ✓
- x is a pointer to function that returns a variable of type int

int (*x)[] ();

- x as pointer to array of function returning int ✓
- x is a pointer to a function that takes as an argument an array of integers
- x is a function which takes as an argument an array of pointers to variables of type int
- x is a function that takes an array as an argument and returns a pointer to int

char * (* (* * x [ ] [8])[] ( )) [ ];

- x is an array of array 8 of pointer to pointer to array of function returning pointer to array of pointer to char ✓
- x is an array of 8 pointers to pointer to function returning pointer to array of pointer to char
- x is a function that takes as input an array of 8 pointers to pointer to an array and returns a pointer to array of pointer to char
- x is array of array of 8 pointers to functions returning pointer to array of pointer to char

int * (* x) ( ) [];

- x is an array of pointers to functions that take no arguments and return pointers to int
- x is a pointer to function returning array of pointer to int ✓
- x is a function that takes as an argument a pointer to an array of pointers and returns a pointer to int
- x is a function that takes as argument an array of pointers to functions and return a pointer to int

char * (* x ( )) [20];

- x is an array of 20 pointers to functions returning char
- x is an array of pointers to functions returning pointers to functions returning pointers to char
- x is a function returning pointer to array 20 of pointer to char ✓
- x is a pointer to a function returning a pointer to an array of 20 elements of type char

---

Rätt. 5 av 5 poäng.

2 Sustainability

Please explain the concept of economic sustainability. Provide two examples of how we, software engineers, can positively influence the economic sustainability.

Skriv in ditt svar här

Economic sustainability refers to a business, or a country, being able to sustain itself over its lifespan through its own operations. This includes maximizing efficiency of operations, efficiency of resource distribution, as well as efficient planning for the future. An economically sustainable entity is also often environmentally sustainable, as, for example, the usage of renewable energy is more economically efficient in the long term.

1. As software engineers, we may directly impact the economic sustainability of our employer, for example, by working efficiently and delivering features on time, which contributes to timely fulfillment of the employer's plans. This improves economic sustainability at "build"-time.
2. Moreover, by writing code that is efficient at run-time, we ensure that the resources utilized by the hardware running our software will more efficiently use its resources. Efficient code minimizes runtime and memory complexity, and memory usage at runtime. An example of this is Deepmind optimizing Google's datacenters energy efficiency, which led to hundreds of millions of dollars being saved annually.

Ord: 165

Besvarad.

3 Easter game

Easter game in C!

This year Santa and the Easter Bunny decided to play another game. Last year, they played a racing game, but it was not fair to the Santa, who is much older than the Easter Bunny. Therefore, this year they got inspired by the robot game from the work packages.

The game plays on a board of 255 x 255 squares. In each round, they both get to choose a random number (one byte). Based on the content of the byte, they get to move:
- bit 0, MSB /most significant bit/: vertical direction - 0 (up), 1 (down)
- bit 1: horizontal direction - 0 (left), 1 (right)
- bit 2-3: surprise: if the number if 11 then they get to draw one more time, otherwise nothing happens
- bit 4-5: the number of steps that they move vertically
- bit 6-7: the number of steps that they move horizontally

Santa starts at the position: 127,128 - 127 horizontal, 128 vertical
Easter bunny starts at the position: 128,127 - 128 horizontal, 127 vertical

They play until they meet on the board. The winner is the player who "catches" the other player, i.e., moves to a position which is occupied by the other player.

If they "bump" into the edge of the board, they "appear" on the other side, e.g. 255,100 -> 0, 100
During each draw, the program should print out the status, e.g.:
Draw #1:
>> Santa draws 0xFF, 0b1111 1111, which means: down, right, one more draw, 3 steps vertically, 3 steps horizontally, new position: 130,125
>> Santa draws again 0x45, ...
>> Easter Bunny draws, ...

Draw #2:
>> Santa...

Important: In this task, you must use bitshifting and bitmasking, you are not allowed to use arrays to store the bit values.

Grading:
- Correct implementation of the bit parsing: 2 points
- Correct implementation of the moves: 2 points
- Correct implementation of the printouts: 2 points
- Correct implementation of the "win" situation: 1 point
- Correct implementation of the "bump" condition: 1 point
- Comments: 2 points

Skriv in ditt svar här

```
1  #include <stdio.h> // provides console I/O
2  #include <time.h> // provides time()
3  #include <stdbool.h> // provides bool type
4  #include <stdlib.h> // provides rand()
5
```

```
6  //#define DEBUG
7  #ifdef DEBUG
8  #include <unistd.h> // provides sleep(), only needed in debug mode
9  #endif
10
11 #define FIELD_SIZE 256 // the size of the playing field, 0 up to and not includi
12
13 // the stream to put logs into (stdout/stderr). set to NULL for a significantly
14 #define OUTPUT_STREAM stdout // could theoretically be a file on disk, but would
       point
15
16 // structure representing a position in 2D space
17 typedef struct {
18     int x; // x coordinate
19     int y; // y coordinate
20 } pos_t; // define as typename `pos_t`
21
22 #define PLAYER_C 2 // player count
23 typedef enum { SANTA, BUNNY } player_enum; // enumerate players
24
25 // structure representing a player
26 typedef struct {
27     pos_t pos; // player's position
28     char* name; // player's name
29 } player_t; // define as typename `player_t`
30
31 // random_byte() - returns a random byte (number in range 0..255)
32 unsigned char random_byte() {
33     return rand() % (1 << 8); // the maximum number is 2^8 - 1
34 }
35
36 // print_as_hex() - prints the given byte in hex format into the given stream
37 void print_as_hex(unsigned char byte, FILE* output) {
38     if (output == NULL) return; // if no output provided, return
39
40     fputs("0x", output); // print the start of the hex
41     char chunk = (byte >> 4) & 0b1111; // get the 4 most significant bits
42     if (chunk < 10) { // if its a digit
43         fputc(chunk + '0', output); // print the corresponding digit character
44     } else { // if its a letter
45         fputc((chunk - 10) + 'A', output); // print the corresponding letter cha
46     }
47
48     chunk = byte & 0b1111; // get the 4 least significant bits
49     if (chunk < 10) { // if its a digit
50         fputc(chunk + '0', output); // print the corresponding digit character
51     } else { // if its a letter
52         fputc((chunk - 10) + 'A', output); // print the corresponding letter cha
53     }
54 }
55
56 // print_as_hex() - prints the given byte in hex format into the given stream
57 void print_as_bin(unsigned char byte, FILE* output) {
58     if (output == NULL) return; // if no output provided, return
59
60     fputs("0b", output); // print the start of the binary
61     for (int i = 7; i >= 0; i--) { // iterate through byte starting at MSB
62         unsigned char bit = (byte >> i) & 1; // get the bit
63         fputc(bit + '0', output); // put the corresponding digit char
64         if (i == 4) fputc(' ', output); // if its the middle, print the separato
65     }
66 }
67
68 // apply_move() - applies the given movement byte to the given player,
69 // logs the movement into the given output stream
70 // returns true if the player shall take another turn
71 bool apply_move(player_t* player, unsigned char move, FILE* output) {
72     char dy = (move >> 7) & 1; // MSB / bit0 - vertical direction
73     char dx = (move >> 6) & 1; // bit1 - horizontal direction
74
75     char y_steps = ((move >> 2) & 0b11); // bits 4-5 - steps vertically
76     char x_steps = (move & 0b11); // bits 6-7 (LSB) - steps horizontally
77
78     bool extra_turn = ((move >> 4) & 0b11) == 0b11; // true if bits 2-3 are exac
```

```
    bool extra_turn = ((move >> 4) & 0b11) == 0b11; // true if bits 2-3 are exac
79
80      // if the output stream is provided, print logs in a format defined by the r
81      if (output != NULL) {
82          fprintf(output,  "\n>> %s draws ", player->name); // player's name
83          print_as_hex(move, output); // the bit as hex
84          fputs(", ", output);
85          print_as_bin(move, output); // the bit as binary
86          fprintf(output, ", which means: ");
87          // the movement
88          fprintf(output,
89              "%s, %s, %s, %d steps vertically, %d steps horizontally, ",
90              (dy == 1) ? "down" : "up",
91              (dx == 1) ? "right" : "left",
92              extra_turn ? "one more draw" : "last draw",
93              y_steps,
94              x_steps
95          );
96      }
97
98      // transform directions to be -1 or 1 for ease of calculation
99      dx = (dx * 2) - 1; // multiply by 2 to get 0 or 2, then decrement to get -1
100     dy = ((~dy & 1) * 2) - 1; // invert the dy bit before shifting, then apply t
101
102     // apply the movement with wrapping (0..FIELD_SIZE not inclusive)
103     player->pos.y = (player->pos.y + dy * y_steps) % FIELD_SIZE;
104     player->pos.x = (player->pos.x + dx * x_steps) % FIELD_SIZE;
105
106     if (output != NULL) { // if an output stream is provided
107         fprintf(output, "new position: %d,%d", player->pos.x, player->pos.y); //
108     }
109
110     return extra_turn; // return true if the player shall take another turn
111 }
112
113 // same_pos() - returns true if the given positions are identical
114 bool same_pos(pos_t* pos1, pos_t* pos2) {
115     return (pos1->x == pos2->x) && (pos1->y == pos2->y);
116 }
117
118 // main() - entry point.
119 // simulates a game between 2 players.
120 // draws a random byte each turn, parses and applies it to a player according to
121 // the game ends when one player "catches" another, and the winner is printed
122 int main()
123 {
124     srand(time(NULL)); // initialize the random seed
125
126     // initialize the Santa player according to the requirements
127     player_t santa = {
128         .pos = {
129             .x = 127,
130             .y = 128
131         },
132         .name = "Santa"
133     };
134
135     // initialize the Bunny player according to the requirements
136     player_t bunny = {
137         .pos = {
138             .x = 128,
139             .y = 127
140         },
141         .name = "Easter Bunny"
142     };
143
144     player_enum current_player_idx = SANTA; // index of the current player
145     // players will move in the order they are listed in the player_t enum
146
147     unsigned int turn_n = 1; // turn counter
148     if (OUTPUT_STREAM != NULL) { // print the turn number if the output stream i
149         printf("\n\nDraw #%d:", turn_n);
150     }
151
```

```
152     for (;;){ // infinite loop [ that looks like a cute walrus :) ]
153
154         unsigned char move = random_byte(); // get the move to apply to a player
155
156         bool moves_again = false; // if the player gets to move again, will be r
157         switch(current_player_idx) {
158             case SANTA:
159                 moves_again = apply_move(&santa, move, OUTPUT_STREAM); // apply
                        repeats the move
160                 break;
161             case BUNNY:
162                 moves_again = apply_move(&bunny, move, OUTPUT_STREAM); // apply
                        repeats the move
163                 break;
164         }
165
166         if (same_pos(&santa.pos, &bunny.pos)) break; // if the positions match,
                current player
167         if (!moves_again) { // if the player does not get to draw again
168             current_player_idx = current_player_idx + 1; // increment the player
169             if (current_player_idx >= PLAYER_C) { // if the last player just pla
170                 current_player_idx = current_player_idx % PLAYER_C; // reset the
171                 turn_n++; // increase the turn counter
172                 if (OUTPUT_STREAM != NULL) { // print the turn number if the out
173                     printf("\n\nDraw #%d:", turn_n);
174                 }
175             }
176         }
177 #ifdef DEBUG
178         sleep(2);
179 #endif
180     }
181
182     switch(current_player_idx) {
183         case SANTA:
184             printf("Santa wins");
185             break;
186         case BUNNY:
187             printf("Easter Bunny");
188             break;
189     }
190
191     printf(" wins after %d turns!\n", turn_n);
192
193     return 0;
194 }
195
```

Besvarad.

4 Amdahls law

Implementing and simulating Amdahls law

The Amdahls law:

$$speedup = \frac{1}{S + \frac{1-S}{N}}$$

Where:
S - serial portion of the program
N - number of processing cores

Your task is:
1. Write a function with the signature `double speedup(double S, unsigned int N)` to calculate the speedup of the program given the Amdahl's law.
2. Write a `main()` function which 100 times randomly chooses S and N and displays the results in the following format:

Choice #1: S = 0.33, N = 8, speedup = 2.4
Choice #2: ...

Grading:
1. Correct implementation of the function: 2 points
2. Correct implementation of main: 2 points
3. Commenting: 2 points
4. Fail-safety: 2 points

Skriv in ditt svar här

```c
#include <stdio.h> // provides console I/O functions
#include <stdlib.h> // provides rand()
#include <time.h> // provides time()

#define CHOICES 100 // amount of choices to simulate
#define MAX_N 16 // the maximum amount of processing cores in a choice

// speedup(S,N) - calculates the maximum speedup gained by using parallel process
// according to Amdahl's Law, given S - serial portion of the program ()
double speedup(double S, unsigned int N) {
    if (N == 0) return -1; // prevent division by zero
    if (S < 0 || S > 1) return -1;
    return 1 / (S + ((1 - S) / N)); // return the result according to the formula
}

// main() - entry point. Generates an S and an N `CHOICES` times,
// and calculates the speedup according to Amdahl's Law and prints it
int main()
{
    srand(time(NULL)); // create the random seed using current time
    for (int i = 0; i < CHOICES; i++) {
        double S = (rand() % 101) / 100.0; // get an integer between 0 and 100, a
        unsigned int N = (rand() % MAX_N) + 1; // get an integer in range 1..MAX_
        printf("Choice #%d: S = %.2f, N = %d: ", i + 1, S, N); // print the curre

        double res = speedup(S, N); // get the result
        if (res == -1) { // check for an error
            printf("Unable to calculate speedup\n"); // print error
```

```
28              printf("Unable to calculate speedup\n"); // print error
29              continue; // skip this loop
30          }
31
32          printf("speedup = %.2f\n", res); // print the result if there was no erro
33      }
34
35      return 0; // exit with success
36  }
37
```

Besvarad.

5 Array programming

Write a program in C that reads 5 strings from the command line arguments, stores them in an array, and finds the shortest string in the array. The program should then remove the shortest string and return the array, now shorter by one element.

Requirements:

1. **Reading Strings:** The program should read 5 strings from the command line arguments and store them in an array.
2. **Finding the Shortest String:** The program should identify the shortest string in the array.
3. **Removing the Shortest String:** Once the shortest string is identified, the program should remove it from the array.
4. **Returning the Modified Array:** After removing the shortest string, the program should return the array, now containing one less element.
5. **Displaying Output:** The program should write the string that has been removed to the console and display all elements of the array both before and after the removal of the shortest string.

Typical usage: **main.exe string1 string23 string_is_ok another_string loremIpsum**

The program should contain a separate function to find and remove the shortest string. The program should have a main function that tests the functionality.

Grading Criteria:

- Correct functionality (as specified above): 3 points
- Comments: 2 points
- Function to find and remove the element: 2 points
- Main function to test the program: 2 points
- Safety checks (e.g., handling edge cases): 1 point

Skriv in ditt svar här

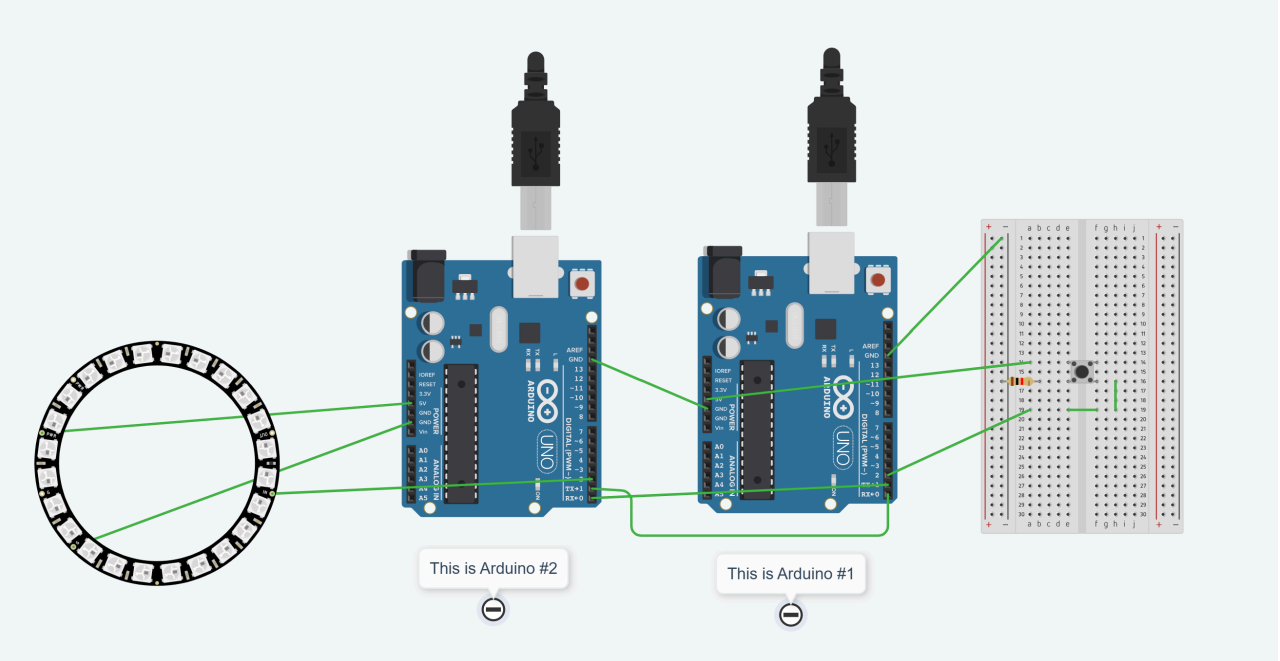
```c
#include <stdio.h> // provides console I/O
#include <stdlib.h> // provides memory utilities
#include <string.h> // provides string utilities

#define STR_C 5 // expected amount of strings

// structure representing an array of strings
typedef struct {
    char** data; // the array itself
    size_t len; // its length
} str_arr; // assign this a typename `str_arr`

// free_arr() - frees a dynamically allocated `str_arr`
void free_arr(str_arr* arr) {
    for (int i = 0; i < arr->len; i++) { // go through each element
        free(arr->data[i]); // free its memory
    }
    free(arr->data); // free the pointer to the array
    free(arr); // free the pointer to the `str_arr`
}

// remove_shortest() - removes the shortest element in the given array.
// prints the removed element and returns the new array without it as a dynamica
str_arr* remove_shortest(char** arr, size_t len) {
    str_arr* res = malloc(sizeof(res)); // allocate the result object
    res->data = NULL; // initialize data as null pointer
```

```
    res->len = 0; // initialize size as 0

    if (len == 0) return res; // if there were no elements, cannot remove anythi
    if (len == 1) { // if there was only 1, must remove only it
        printf("Removed: %s\n", arr[0]); // print the element
        return res; // return the empty str_arr
    }

    // assume the 0th string is shortest
    int min_idx = 0; // index of the smallest string in array
    int min_len = strlen(arr[0]); // length of the smallest string in array

    for (size_t i = 1; i < len; i++) { // go through the array
        int len = strlen(arr[i]); // get the length of the current element
        if (len < min_len) { // if it's the new minimum
        // re-assign the minimums to the current index and length
            min_len = len;
            min_idx = i;
        }
    }

    printf("Removed: %s\n", arr[min_idx]); // print the shortest string

    res->data = calloc(len - 1, sizeof(char*)); // allocate the resulting array
    char** p = res->data; // pointer to the 0th element for iteration

    for (int i = 0; i < len; i++) { // go through the elements of the initial ar
        if (i == min_idx) continue; // if its the shortest string, skip it
        *p = malloc(strlen(arr[i]) * sizeof(char)); // allocate space for the co
        strcpy(*p, arr[i]); // copy from the initial array into the result
        res->len++; // increment the length of the result array
        p++; // increment the pointer
    }

    return res; // return the result
}

// print_arr() - prints all elements of the given `str_arr` in one line
void print_arr(str_arr* arr) {
    printf("Contents of the array: "); // print the start of the output
    for (int i = 0; i < arr->len; i++) { // go through the array
        printf("%s ", arr->data[i]); // print the current element
    }
    printf("\n"); // finish the line
}

// main() - entry point. gets 5 strings from command line arguments,
// copies them into an array, prints it,
// then removes the shortest string, and prints the new array
int main(int argc, char* argv[])
{
    if (argc != 1 + STR_C) { // check the amount of arguments to be the string c
        printf("Invalid usage. Expected %d strings\n", STR_C); // print an error
        return 1; // exit with error code
    }

    char* strings[STR_C]; // declare the array of strings
    for (int i = 0; i < STR_C; i++) { // go through the arguments
        strings[i] = argv[i + 1]; // copy the string from the arguments into the
    }

    // put the string into a `str_arr`
    str_arr orig = {
        .data = strings,
        .len = STR_C
    };

    print_arr(&orig); // print the original values
    str_arr* new_arr = remove_shortest(strings, STR_C); // get the array with th
    print_arr(new_arr); // print the new array
    free_arr(new_arr); // free the resulting array
    // no need to free `orig` as it's on the stack

    return 0; // exit with success code
}
```

```
100         return 0; // exit with success code
101     }
102
```

Besvarad.

6 Two arduinos



Two arduinos working together!

In this task, you have to program both arduinos so that they realize the following functionality.

When the user presses the button on Arduino #1, one LED should turn on the ring controlled by Arduino #2. When the user presses the button one more time, the second LED should turn on, etc.

When all LCDs are on, and the user presses the button again, then the first LED should change color. When the next time the user presses the button, the next LED should change color (the same as the previous LED), and so on and so on.

Conditions: you MUST use interrupts for the button.

The code for both arduinos should be the same, and you should use the right mechanisms to make sure that only one of this code is used. Similar to the way when we have code for the Linux and the Windows platform in the same file.

Hint! You can use the mechanisms of variability that we discussed in the course and during the guest lecture from Grundfos.

Grading:
- Correct implementation of interrupt: 2 points
- Correct implementation of communication: 2 points
- Correct implementation of the LEDs/ring: 2 points
- Comments: 2 points
- Variability management: 2 points

In this question you are allowed to use TinkerCad, in the resource link below.

Please use the following procedure to log in to TinkerCad:

- go to log-in

- choose "Students with Class Code"

*- use the class code "*IBM V6R XJU*"*

- use join with nickname. Your anonymous code is your nickname - without the dashes and using small letters, e.g., DIT633-0001-ABC becomes dit6330001abc as your nickname

Skriv in ditt svar här

```
#define RECEIVER
// THIS HAS TO BE DEFINED AS ONE OF THE FOLLOWING:
//  SENDER - has the button and sends the signal
//   RECEIVER - receives the signal and controls the LEDs

// define constants according to the board mode
#ifdef SENDER
// SENDER MODE CONTANTS
#define BUTTON_PIN 2 // input button pin

#elif defined(RECEIVER)
// RECEIVER MODE CONSTANTS
#define POLLING_MS 100 // delay between updating the status
#define RING_PIN 2 // neopixel ring pin
#include <Adafruit_NeoPixel.h> // Neopixel library
#define NUMPIXELS 24 // number of pixels in the ring

// initialize the ring
Adafruit_NeoPixel ring = Adafruit_NeoPixel(NUMPIXELS, RING_PIN, NEO_GRB + NEO_KHZ

// initialize the color as 0
int red = 0;
int green = 0;
int blue = 0;

char counter = 0; // the address of the pixel being modified

// rand_color() - picks a random color
void rand_color(){
  red = random(0, 255);
  green = random(0,255);
  blue = random(0, 255);
}
#endif

void setup()
{
  Serial.begin(9600); // initialize the serial at baud 9600
#ifdef SENDER
  // SENDER MODE SETUP
  pinMode(BUTTON_PIN, INPUT); // set the button as input
  // attach interrupt to the button
  attachInterrupt(digitalPinToInterrupt(BUTTON_PIN), button_pressed, RISING);

#elif defined(RECEIVER)
  // RECEIVER MODE SETUP
  randomSeed(analogRead(A0)); // use the noise from the analog port as a RNG seed
  rand_color(); // set the initial color

#else
  // NO MODE DEFINED
  Serial.println("Mode unknown, check source code!!!"); // print an error
  // light up the built-in LED to indicate an error physically
  pinMode(LED_BUILTIN, OUTPUT);
  digitalWrite(LED_BUILTIN, HIGH);

  // trap the program in an infinite loop
```

```
  // trap the program in an infinite loop
  nointerrupts();
  for (;;){}
#endif
}

void loop(){
#ifdef RECEIVER
  if (Serial.available() > 0) { // if there is data on the serial
    Serial.read(); // consume it, value is unused

    // set the color of the current pixel to the current color
    ring.setPixelColor(counter, ring.Color(red, green, blue));
    ring.show(); //..and write it to the ring

    // NOTE: the random colors might be very close and hard to notice!

    counter++; // increment the counter
    if (counter >= NUMPIXELS) { // if it was the last pixel
      rand_color(); // choose a new color
      counter = 0; // reset the counter
    }
  }
  delay(POLLING_MS); // wait between loops
#endif
}

#ifdef SENDER
void button_pressed() {
Serial.write(1); // send a signal to the receiver
}
#endif
```

Besvarad.