***Test Task for the AUTOSAR program***

Task 1.1

* To make it possible to calculate minimum and maximum value of two arguments please implement both:
* "MIN(a, b)" and/or "MAX(a, b)" macro-functions and
* "Int min(int a, int b)" and/or "Int max(int a, int b)" regular functions
* Please execute them in such code like this:

void main()

{

int x1 = 3, x2 = 3;

int y1 = 5, y2 = 5;

printf("MIN(%d, %d) = %d;\n", x, y, MIN(x1++, y1));

printf("MIN(%d, %d) = %d;\n", x, y, MIN(++x1, y1));

printf("min(%d, %d) = %d;\n", x, y, min(++x2, y2));

printf("min(%d, %d) = %d;\n", x, y, min(x2++, y2));

}

* Please provide the output and justify it. Everything would be interesting to know

#include <iostream>

#define MIN(a,b) a < b ? a : b

#define MAX(a,b) a > b ? a : b

int min(int a, int b)

{

if (a > b) return b;

else if (a < b) return a;

}

int max(int a, int b)

{

if (a < b) return b;

else if (a > b) return a;

}

void main()

{

int x1 = 3, x2 = 3;

int y1 = 5, y2 = 5;

int x3 = 3, x4 = 3;

int y3 = 5, y4 = 5;

printf("MIN(%d, %d) = %d;\n", x1, y1, MIN(x1++, y1));

printf("MIN(%d, %d) = %d;\n", x1, y1, MIN(++x1, y1));

printf("min(%d, %d) = %d;\n", x2, y2, min(++x2, y2));

printf("min(%d, %d) = %d;\n", x2, y2, min(x2++, y2));

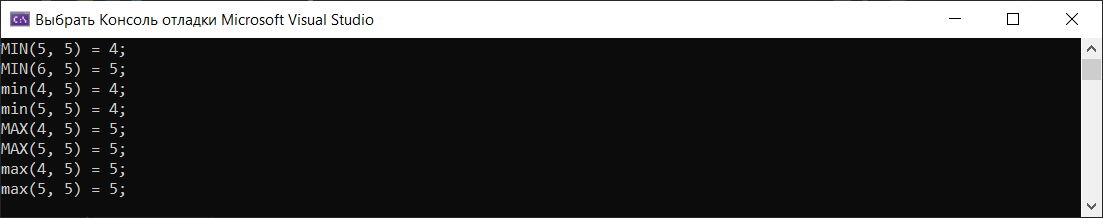
printf("MAX(%d, %d) = %d;\n", x3, y3, MAX(x3++, y3));

printf("MAX(%d, %d) = %d;\n", x3, y3, MAX(++x3, y3));

printf("max(%d, %d) = %d;\n", x4, y4, max(++x4, y4));

printf("max(%d, %d) = %d;\n", x4, y4, max(x4++, y4));

}



The screenshot shows the results of both macro-functions and regular functions. Macro-functions and regular functions involve different variables and different values respectively. In the program, the macro-function MIN(a,b) is called twice, because during these calls the value of the first, argument changes, the value of “x1” at the output equals 5, but when the first macro-function is called it equals 4. It follows that it is better to pass the final value as an argument, and not an expression as it is done in the program, because the expression will be evaluated as many times as this parameter is used in the body of the macro-function.

Task 1.2

* To make it possible (much simpler) to interact with HW elements via registers please implement next macros:
* #define SET\_BIT(x, p) // set (set to 1) proper bit into the "x" byte by "p" position
* #define CLR\_BIT(x, p) // clear (set to 0) proper bit into the "x" byte by "p" position
* #define TGL\_BIT(x, p) // flip/toggle proper bit into the "x" byte by "p" position
* #define CHK\_BIT(x, p) // check proper bit into the "x" byte by "p" position
* To make it possible to output the data in human readable binary format please implement the function like "void Int2BitStr(int x, char \* str, int size)" that shall convert an integer variable into the string in binary format like: "0b11001100" after that this string could be printed with the use of the "%s" "printf()" formatting descriptor
* Finally, please implement a main function with the use of these macros and please print the output after you have used the macros.

#include <iostream>

#include <bitset>

#include <cstring>

#include <string>

using namespace std;

#define SET\_BIT(x, p) x | (1 << p) // set (set to 1) proper bit into the "x" byte by "p" position

#define CLR\_BIT(x, p) x & ~(1 << p) // clear (set to 0) proper bit into the "x" byte by "p" position

#define TGL\_BIT(x, p) x ^ (1 << p) // flip/toggle proper bit into the "x" byte by "p" position

#define CHK\_BIT(x, p) x & (1 << p) ? "Set" : "Not set" // check proper bit into the "x" byte by "p" position

void Int2BitStr(int x, char \*str, int size)

{

string bin\_form("");

int mask = 1;

for (int i = 0; i < size; i++)

{

if ((mask & x) >= 1)

bin\_form = "1" + bin\_form;

else

bin\_form = "0" + bin\_form;

mask <<= 1;

}

str = \_strdup(bin\_form.c\_str());

printf("0b%s\n",str);

}

void main()

{

int x1 = 11, p1 = 2, x2=13, p2=0, x3=10, p3=1, size1=5, size2=7, size3= 10;

char \*str1{}, \*str2{}, \*str3{}, \*str4{};

printf("setting proper bit into the %d byte by %d position: %d = ", x1, p1, SET\_BIT(x1, p1));

Int2BitStr(SET\_BIT(x1, p1), str1, size1);

printf("clearing proper bit into the %d byte by %d position: %d = ", x2, p2, CLR\_BIT(x2, p2));

Int2BitStr(CLR\_BIT(x2, p2), str2, size2);

printf("fliping proper bit into the %d byte by %d position: %d = ", x3, p3, TGL\_BIT(x3, p3));

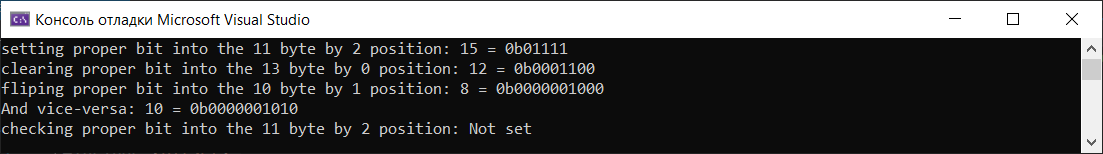
Int2BitStr(TGL\_BIT(x3, p3), str3, size3);

printf("And vice-versa: %d = ", TGL\_BIT(TGL\_BIT(x3, p3), p3));

Int2BitStr(TGL\_BIT(TGL\_BIT(x3, p3), p3), str4, size3);

printf("checking proper bit into the %d byte by %d position: %s\n", x1, p1, CHK\_BIT(x1, p1));

}



Task 2

Please visit this website: "https://wokwi.com/". And, using the design tools, please design the system that shall

satisfy next requirements:

1. The Board shall be Arduino Uno – 1 pc
2. Sensors:
   1. Potentiometer – 1 pc
3. The actuators:
   1. LEDs – 4 pcs (Red, Green, Blue, Yellow)
4. LEDs shall blink sequentially starting from the Red LED with the various period.
5. Period shall depend on the Potentiometer position:

* min value 100 ms
* max value 10 000 ms

**sketch.ino:**

void setup() {

  // put your setup code here, to run once:

**Serial**.begin(115200);

  if (!**Serial**) **Serial**.println("serial is not ok");

  pinMode(3, OUTPUT);

  pinMode(4, OUTPUT);

  pinMode(5, OUTPUT);

  pinMode(6, OUTPUT);

  pinMode(A1, INPUT);

}

void loop() {

  // put your main code here, to run repeatedly:

  int value = analogRead(A1);

  int a;

**Serial**.print("Potentiometer value: ");

**Serial**.println(value);

  if (value >0 && value <= 50 ) a=random(100,500);

  if (value >51 && value <= 100 ) a=random(501,1000);

  if (value >101 && value <= 150 ) a=random(1001,1500);

  if (value >151 && value <= 200 ) a=random(1501,2000);

  if (value >201 && value <= 250 ) a=random(2001,2500);

  if (value >251 && value <= 300 ) a=random(2501,3000);

  if (value >301 && value <= 350 ) a=random(3001,3500);

  if (value >351 && value <= 400 ) a=random(3501,4000);

  if (value >401 && value <= 450 ) a=random(4001,4500);

  if (value >451 && value <= 500 ) a=random(4501,5000);

  if (value >501 && value <= 550 ) a=random(5001,5500);

  if (value >551 && value <= 600 ) a=random(5501,6000);

  if (value >601 && value <= 650 ) a=random(6001,6500);

  if (value >651 && value <= 700 ) a=random(6501,7000);

  if (value >701 && value <= 750 ) a=random(7001,7500);

  if (value >751 && value <= 800 ) a=random(7501,8000);

  if (value >801 && value <= 850 ) a=random(8001,8500);

  if (value >851 && value <= 900 ) a=random(8501,9000);

  if (value >901 && value <= 950 ) a=random(9001,9500);

  if (value >951 && value <= 1100 ) a=random(9501,10000);

**Serial**.print("Period: ");

**Serial**.print(a);

**Serial**.println(" ms");

  digitalWrite(6, HIGH);

  delay(a);

  digitalWrite(6, LOW);

  digitalWrite(5, HIGH);

  delay(a);

  digitalWrite(5, LOW);

  digitalWrite(4, HIGH);

  delay(a);

  digitalWrite(4, LOW);

  digitalWrite(3, HIGH);

  delay(a);

  digitalWrite(3, LOW);

}

**diagram.json:**

{

  "version": 1,

  "author": "Anonymous maker",

  "editor": "wokwi",

  "parts": [

    {

      "type": "wokwi-arduino-uno",

      "id": "uno",

      "top": 0.6,

      "left": 28.2,

      "rotate": 90,

      "attrs": {}

    },

    {

      "type": "wokwi-potentiometer",

      "id": "pot1",

      "top": 113.9,

      "left": -163.4,

      "rotate": -90,

      "attrs": {}

    },

    {

      "type": "wokwi-led",

      "id": "led1",

      "top": -13.2,

      "left": 426.2,

      "attrs": { "color": "red" }

    },

    {

      "type": "wokwi-led",

      "id": "led2",

      "top": 92.4,

      "left": 426.2,

      "attrs": { "color": "green" }

    },

    {

      "type": "wokwi-led",

      "id": "led3",

      "top": 44.4,

      "left": 426.2,

      "attrs": { "color": "blue" }

    },

    {

      "type": "wokwi-led",

      "id": "led4",

      "top": 140.4,

      "left": 426.2,

      "attrs": { "color": "yellow" }

    },

    {

      "type": "wokwi-resistor",

      "id": "r1",

      "top": 29.4,

      "left": 326.4,

      "attrs": { "value": "1000" }

    },

    {

      "type": "wokwi-resistor",

      "id": "r2",

      "top": 87,

      "left": 326.4,

      "attrs": { "value": "1000" }

    },

    {

      "type": "wokwi-resistor",

      "id": "r3",

      "top": 183,

      "left": 326.4,

      "attrs": { "value": "1000" }

    },

    {

      "type": "wokwi-resistor",

      "id": "r4",

      "top": 135,

      "left": 326.4,

      "attrs": { "value": "1000" }

    }

  ],

  "connections": [

    [ "pot1:VCC", "uno:5V", "red", [ "v-3.65", "h49.4", "v-19.2", "h125.2" ] ],

    [ "pot1:GND", "uno:GND.2", "black", [ "h20.6", "v-0.2", "h86.4", "v-28.8", "h67.6" ] ],

    [ "pot1:SIG", "uno:A1", "green", [ "v-1.8", "h67.66", "v31" ] ],

    [ "led4:A", "r3:2", "yellow", [ "v0" ] ],

    [ "led2:A", "r4:2", "green", [ "v0" ] ],

    [ "led3:A", "r2:2", "blue", [ "v0" ] ],

    [ "led1:A", "r1:2", "red", [ "v0" ] ],

    [ "r3:1", "uno:3", "yellow", [ "v0" ] ],

    [ "r4:1", "uno:4", "green", [ "v0" ] ],

    [ "r2:1", "uno:5", "blue", [ "h-9.6", "v69.5" ] ],

    [ "r1:1", "uno:6", "red", [ "h-19.2", "v127.2" ] ],

    [ "led4:C", "uno:GND.1", "black", [ "v48", "h-153.2", "v-147.8" ] ],

    [ "led2:C", "uno:GND.1", "black", [ "h-143.6", "v-51.8" ] ],

    [ "led3:C", "uno:GND.1", "black", [ "v0" ] ],

    [ "led1:C", "uno:GND.1", "black", [ "h-18.8", "v53.8" ] ]

  ]

}

