**Лабараторна робота №6**

З дисципліни «Програмування»

Виконав: Землянський Едуард

Група: КВ-22

**Вимоги до програми**

1. Демонстрація роботи програми повинна починатися з виводу на

екран заставки, на якій будуть дані про виконавця роботи, та меню, яке

передбачатиме можливість перегляду наступних пунктів:

- постановка задачі відповідно до варіанта, її розв’язання,

- виведення двох- та трьохвимірного зображення заданої в варіанті

літери з можливістю зміни її параметрів, наприклад, кольору,

- вихід з програми.

2. Логічно відокремлені частини алгоритма (введення даних, розв’язок

задачі, виведення результату, тощо) оформити у вигляді функцій.

3. При тестуванні програми значення початкових даних підібрати

таким чином, щоб в алгоритмі виконання завдання були перевірені всі

ситуації, які можуть виникнути для заданого алгоритма, тобто

продемонструвати повну коректність та універсальність алгоритма.

Вказівки до виконання роботи:

4. При роботі можна використовувати бібліотеку string.h, графічні

бібліотеки тощо.

5. Використовувати додаткові масиви не можна.

**Варіант**

12. Задано непустий текст довжиною до 255 символів, що містить у собі

слова, у кожному з яких від 1 до 5 малих латинських літер; між сусідніми

словами – коми, за останнім словом – крапка. Надрукувати :

а) усі різні слова, вказавши для кожного слова кількість його

входжень у текст;

б) усі слова в алфавітному порядку.

**Літера:**



**Текст програми:**

#include <windows.h>

#include <gl/gl.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

#include <ctype.h>

LRESULT CALLBACK WindowProc(HWND, UINT, WPARAM, LPARAM);

void EnableOpenGL(HWND hwnd, HDC\*, HGLRC\*);

void DisableOpenGL(HWND, HDC, HGLRC);

void gotoxy(int x, int y){

COORD CRD;

CRD.X = x;

CRD.Y = y;

SetConsoleCursorPosition(GetStdHandle(STD\_OUTPUT\_HANDLE), CRD);

}

void borders()

{

int i;

/\*corners\*/

gotoxy(5, 5);

printf("%c", 201);

gotoxy(45, 5);

printf("%c", 187);

gotoxy(5,16);

printf("%c", 200);

gotoxy(45,16);

printf("%c", 188);

/\*vertical lines\*/

for(i = 6; i < 16; i++)

{

gotoxy(5, i);

printf("%c", 186);

gotoxy(45, i);

printf("%c", 186);

}

/\*horizontal lines \*/

for (i = 6; i < 45; i++)

{

gotoxy(i, 5);

printf("%c", 205);

gotoxy(i, 16);

printf("%c", 205);

}

}

void choice1(){

struct dict{

char word[6];

int count;

};

char text[255];

printf("Enter words via coma to count them: ");

scanf("%s", text);

struct dict words[255];

int words\_counter = 0, char\_counter = 0;

for (int i = 0; i <= strlen(text); i++){

if (text[i] == '.'){

words\_counter++;

break;

}

else if (isalpha(text[i])){

words[words\_counter].word[char\_counter] = text[i];

char\_counter++;

}

else if (text[i] == ',' || text[i] == '\0'){

words[words\_counter].word[char\_counter] = '\0';

words\_counter++;

char\_counter = 0;

}

}

int equal;

for (int i = 0; i < words\_counter; i++){

words[i].count = 0;

for (int j = 0; j < words\_counter; j++){

equal = 1;

if (strncmp(words[i].word, words[j].word, strlen(words[j].word)) == 0){

words[i].count++;

}

}

}

int counter = 0, ascii\_code;

struct dict demo[words\_counter];

for (int i = 97; i <= 122; i++){

for (int j = 0; j < words\_counter; j++){

ascii\_code = words[j].word[0];

if (ascii\_code == i){

equal = 0;

for (int c = 0; c < counter; c++){

if (strncmp(words[j].word, demo[c].word, strlen(demo[c].word)) == 0){

equal = 1;

}

}

if(!equal){

demo[counter] = words[j];

counter++;

}

}

}

}

for (int j = 0; j < counter; j++){

printf("%s - %d\n", demo[j].word, demo[j].count);

}

}

void choice2(int red, int green, int blue){

glEnable(GL\_LINE\_SMOOTH);

glLineWidth(5);

glBegin(GL\_LINE\_STRIP);

glColor3f(red, green, blue);

glVertex2f(-0.5, 0.375); // 1

glVertex2f(-0.375, 0.5); // 2

glVertex2f(0.375, 0.5); // 4

glVertex2f(0.5, 0.375); // 5

glVertex2f(0.5, 0.125); // 6

glVertex2f(0.375, 0); // 7

glVertex2f(0, 0); // 8

glVertex2f(0.375, 0); // 9

glVertex2f(0.5, -0.125); // 10

glVertex2f(0.5, -0.375); // 11

glVertex2f(0.375, -0.5); // 12

glVertex2f(-0.375, -0.5); // 13

glVertex2f(-0.5, -0.375); // 14

glEnd();

}

void choice3(int red, int green, int blue){

glEnable(GL\_LINE\_SMOOTH);

glLineWidth(5);

glBegin(GL\_LINE\_STRIP);

glColor3f(red, green, blue);

glVertex2f(-0.5, 0.375); // 1

glVertex2f(-0.375, 0.5); // 2

glVertex2f(0.375, 0.5); // 4

glVertex2f(0.5, 0.375); // 5

glVertex2f(0.5, 0.125); // 6

glVertex2f(0.375, 0); // 7

glVertex2f(0, 0); // 8

glVertex2f(0.375, 0); // 9

glVertex2f(0.5, -0.125); // 10

glVertex2f(0.5, -0.375); // 11

glVertex2f(0.375, -0.5); // 12

glVertex2f(-0.375, -0.5); // 13

glVertex2f(-0.5, -0.375); // 14

glEnd();

glBegin(GL\_LINE\_STRIP);

glColor3f(red, green, blue);

glVertex2f(-0.5 + 0.1025, 0.375 + 0.1025); // 1

glVertex2f(-0.375 + 0.1025, 0.5 + 0.1025); // 2

glVertex2f(0.375 + 0.1025, 0.5 + 0.1025); // 4

glVertex2f(0.5 + 0.1025, 0.375 + 0.1025); // 5

glVertex2f(0.5 + 0.1025, 0.125 + 0.1025); // 6

glVertex2f(0.375 + 0.1025, 0 + 0.1025); // 7

glVertex2f(0 + 0.1025, 0 + 0.1025); // 8

glVertex2f(0.375 + 0.1025, 0 + 0.1025); // 9

glVertex2f(0.5 + 0.1025, -0.125 + 0.1025); // 10

glVertex2f(0.5 + 0.1025, -0.375 + 0.1025); // 11

glVertex2f(0.375 + 0.1025, -0.5 + 0.1025); // 12

glVertex2f(-0.375 + 0.1025, -0.5 + 0.1025); // 13

glVertex2f(-0.5 + 0.1025, -0.375 + 0.1025); // 14

glEnd();

glBegin(GL\_LINES);

glColor3f(red, green, blue);

glVertex2f(-0.5, 0.375); // 1

glVertex2f(-0.5 + 0.1025, 0.375 + 0.1025); // 1

glVertex2f(0, 0); // 8

glVertex2f(0 + 0.1025, 0 + 0.1025); // 8

glVertex2f(-0.5, -0.375); // 14

glVertex2f(-0.5 + 0.1025, -0.375 + 0.1025); // 14

glEnd();

}

void menu(char \*title, char \*choice){

char \*line1 = title;

char line2[] = "Eduard Zemlyanski";

char line3[] = "KV-22";

char line4[] = "1.Task";

char line5[] = "2.Letter 2D";

char line6[] = "3.Letter 3D";

char line7[] = "4.Exit program";

char line8[] = "Make a choice 1/2/3/4: ";

int space(char \*line){

return (40 - strlen(line))/2 + 5;

}

gotoxy(space(line1),7);

puts(line1);

gotoxy(space(line2),8);

puts(line2);

gotoxy(space(line3),9);

puts(line3);

// menu

gotoxy(space(line4),11);

puts(line4);

gotoxy(space(line5),12);

puts(line5);

gotoxy(space(line6),13);

puts(line6);

gotoxy(space(line7),14);

puts(line7);

// end of menu

gotoxy(space(line8), 18);

printf("%s", line8);

\*choice = getchar();

system("cls");

switch (\*choice){

case '1':

choice1();

exit(1);

break;

case '2':

break;

case '3':

break;

case '4':

exit(1);

default:

puts("Wrong choice, try again\n");

Sleep(500);

system("cls");

borders();

menu(title, choice);

}

}

void program(char \*choice){

borders();

menu("Lab No.6", choice);

}

int WINAPI WinMain(HINSTANCE hInstance,

HINSTANCE hPrevInstance,

LPSTR lpCmdLine,

int nCmdShow)

{

// my code

char choice;

program(&choice);

int red = 0, green = 0, blue = 0;

printf("Enter color parameters (r,g,b, 0 to 255): ");

scanf("%d %d %d", &red, &green, &blue);

int condition = (red < 0 || red > 255 || green < 0 || green > 255 || blue < 0 || blue > 255);

if (condition){

while(condition){

system("cls");

puts("WRONG PARAMETERS ! TRY AGAIN.");

Sleep(1500);

system("cls");

printf("Enter color parameters (r,g,b, 0 to 255): ");

scanf("%d %d %d", &red, &green, &blue);

condition = (red < 0 || red > 255 || green < 0 || green > 255 || blue < 0 || blue > 255);

}

}

// end of my code

WNDCLASSEX wcex;

HWND hwnd;

HDC hDC;

HGLRC hRC;

MSG msg;

BOOL bQuit = FALSE;

/\* register window class \*/

wcex.cbSize = sizeof(WNDCLASSEX);

wcex.style = CS\_OWNDC;

wcex.lpfnWndProc = WindowProc;

wcex.cbClsExtra = 0;

wcex.cbWndExtra = 0;

wcex.hInstance = hInstance;

wcex.hIcon = LoadIcon(NULL, IDI\_APPLICATION);

wcex.hCursor = LoadCursor(NULL, IDC\_ARROW);

wcex.hbrBackground = (HBRUSH)GetStockObject(BLACK\_BRUSH);

wcex.lpszMenuName = NULL;

wcex.lpszClassName = "GLSample";

wcex.hIconSm = LoadIcon(NULL, IDI\_APPLICATION);;

if (!RegisterClassEx(&wcex))

return 0;

/\* create main window \*/

hwnd = CreateWindowEx(0,

"GLSample",

"OpenGL Sample",

WS\_OVERLAPPEDWINDOW,

CW\_USEDEFAULT,

CW\_USEDEFAULT,

512,

512,

NULL,

NULL,

hInstance,

NULL);

ShowWindow(hwnd, nCmdShow);

/\* enable OpenGL for the window \*/

EnableOpenGL(hwnd, &hDC, &hRC);

/\* program main loop \*/

while (!bQuit)

{

/\* check for messages \*/

if (PeekMessage(&msg, NULL, 0, 0, PM\_REMOVE))

{

/\* handle or dispatch messages \*/

if (msg.message == WM\_QUIT)

{

bQuit = TRUE;

}

else

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

}

else

{

/\* OpenGL animation code goes here \*/

glClearColor(0.7f, 1.0f, 0.7f, 0.0f);

glClear(GL\_COLOR\_BUFFER\_BIT);

glPushMatrix();

if (choice == '2') choice2(red, green, blue);

else if (choice == '3') choice3(red, green, blue);

glPopMatrix();

SwapBuffers(hDC);

}

}

/\* shutdown OpenGL \*/

DisableOpenGL(hwnd, hDC, hRC);

/\* destroy the window explicitly \*/

DestroyWindow(hwnd);

return msg.wParam;

}

LRESULT CALLBACK WindowProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam)

{

switch (uMsg)

{

case WM\_CLOSE:

PostQuitMessage(0);

break;

case WM\_DESTROY:

return 0;

case WM\_KEYDOWN:

{

switch (wParam)

{

case VK\_ESCAPE:

PostQuitMessage(0);

break;

}

}

break;

default:

return DefWindowProc(hwnd, uMsg, wParam, lParam);

}

return 0;

}

void EnableOpenGL(HWND hwnd, HDC\* hDC, HGLRC\* hRC)

{

PIXELFORMATDESCRIPTOR pfd;

int iFormat;

/\* get the device context (DC) \*/

\*hDC = GetDC(hwnd);

/\* set the pixel format for the DC \*/

ZeroMemory(&pfd, sizeof(pfd));

pfd.nSize = sizeof(pfd);

pfd.nVersion = 1;

pfd.dwFlags = PFD\_DRAW\_TO\_WINDOW |

PFD\_SUPPORT\_OPENGL | PFD\_DOUBLEBUFFER;

pfd.iPixelType = PFD\_TYPE\_RGBA;

pfd.cColorBits = 24;

pfd.cDepthBits = 16;

pfd.iLayerType = PFD\_MAIN\_PLANE;

iFormat = ChoosePixelFormat(\*hDC, &pfd);

SetPixelFormat(\*hDC, iFormat, &pfd);

/\* create and enable the render context (RC) \*/

\*hRC = wglCreateContext(\*hDC);

wglMakeCurrent(\*hDC, \*hRC);

}

void DisableOpenGL (HWND hwnd, HDC hDC, HGLRC hRC)

{

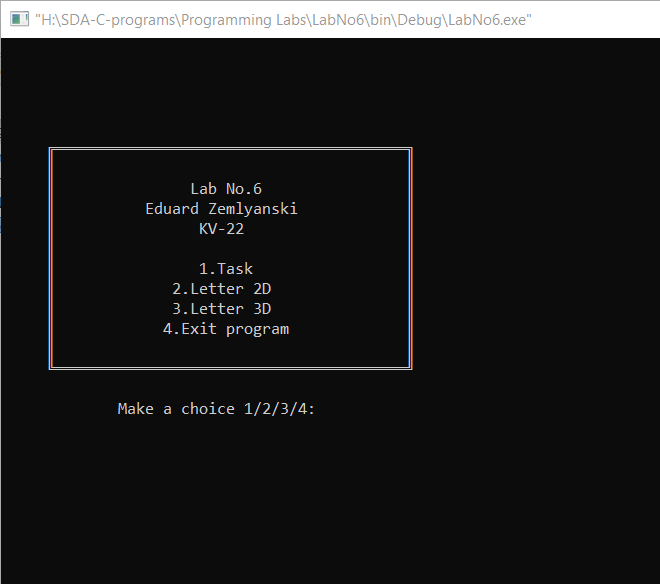
wglMakeCurrent(NULL, NULL);

wglDeleteContext(hRC);

ReleaseDC(hwnd, hDC);

}}

**Результати**



|  |  |
| --- | --- |
|  |  |

