Санкт-Петербургский государственный политехнический университет
Институт компьютерных наук и технологий
Кафедра «Информационная безопасность автоматизированных систем»

К У Р С О В А Я Р А Б О Т А

Генератор кроссвордов

по дисциплине «Методы программирования»

Выполнили
студенты гр.13508/13 Палёный А.Э
<подпись> Романов А.
Преподаватель
<подпись> Вагисаров В.Б

«___» _____ 201__ г.

Санкт-Петербург
2016

ФОРМУЛИРОВКА ЗАДАНИЯ

Создать программу-кроссворд, в возможности которой входит:

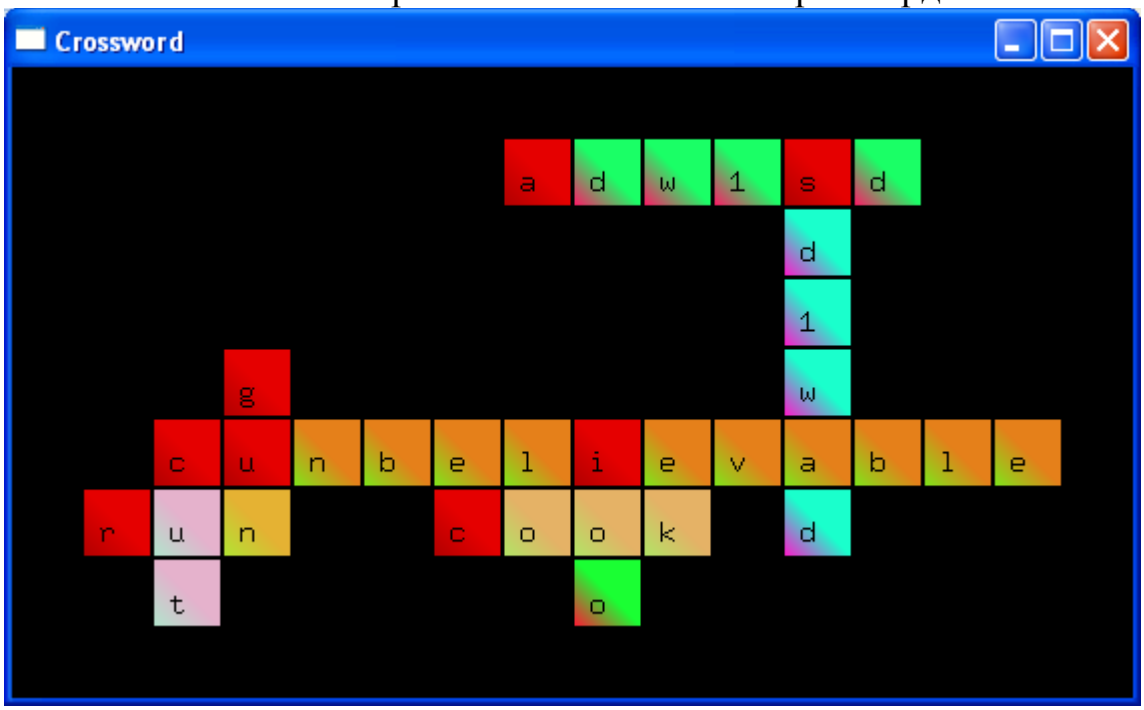
1) Чтение слов из файла и генерация кроссворда с максимальной площадью
2) Чтение слов из файла и генерация кроссворда с минимальной площадью

В программе обязательно должна быть проработана графическая часть, а также вычисление размера поля.

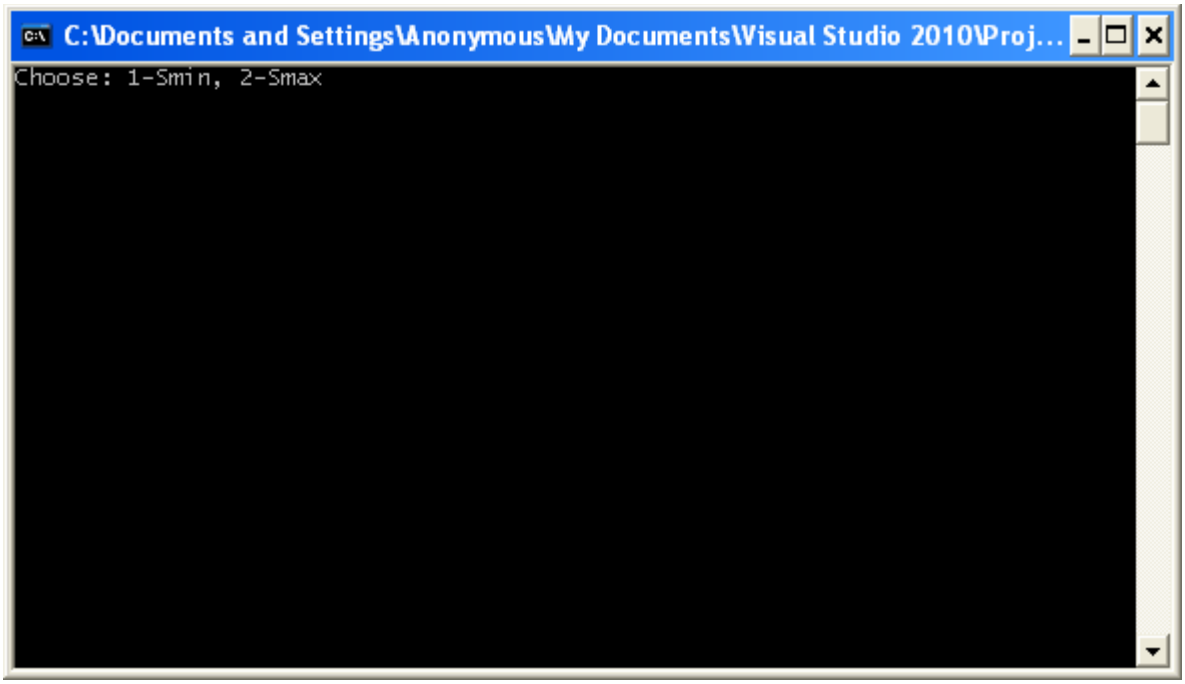
1. ХОД РАБОТЫ

Нами была проделана работа по созданию программы – «Генератор кроссвордов».
Данная программа состоит из нескольких частей, которые находятся в разных файлах – «Main.c», «GraphicalPart.c», «GraphicalPart.h». Каждый файл имеет свою функцию, а именно:

1) – «GraphicalPart.h» - Это заголовочный файл, который связывает между собой два остальных файла с кодом. В нём содержится описание функции, которая находится в «GraphicalPart.c» и которую вызывает «Main.c»
2) – «GraphicalPart.c» - в нём содержится описание всей графической части программы. Эта часть, с помощью библиотеки OpenGL, создаёт «конечное» окно и отрисовывает на нём весь кроссворд.



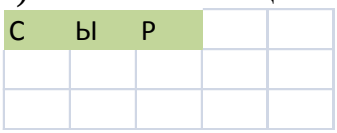
3) – «Main.c» - Это самый главный файл программы. В нём реализовано:
    a. Считывание слов из файла и запись их в линейный список
    b. Запрос у пользователя, какой он хочет получить кроссворд
    c. Формирование поля кроссворда, каждый элемент которого является отдельной структурой состоящей из:
        i. Поля «word» в котором записываются слова
        ii. Поля «color» в котором записывается номер слова для дальнейшей генерации цвета клетки поля
        iii. Поля «logic» в котором записывается информация о том, как расположено слово, элементом которого он является
    d. Запись слов в «поле»
    e. Передача поля в «GraphicalPart.c»
    f. Учёт того, что поместились все слова



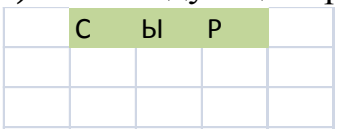
2. ЛОГИКА ПОСТАНОВКИ СЛОВА

Для того чтобы найти самую «удачную» позицию, программа сначала пробегает поле по горизонтали в поисках этой позиции, а потом по вертикали. «Удачной» - называется позиция, в которой слово будет иметь максимальное число пересечений с другими словами.

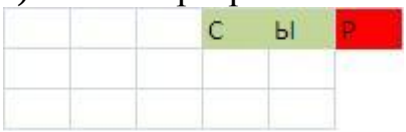
1) С такой позиции программа начинает проверку



2) Это следующая проверка

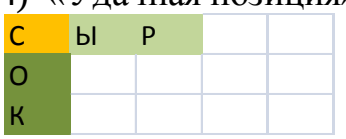


3) Также программа может найти неудачную позицию



Она это учитывает и поэтому отбрасывает это вариант

4) «Удачная позиция»



3. ИСХОДНЫЙ КОД

<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<🡐=====🡒>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

---«Main.c»---

```
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
#include <Windows.h>
#include "GraphicalPart.h"

struct Resault
{
	int Rate;
	int Col;
	int Line;
	int Posit;
};

struct Words
{
	char *word;
	int length;
	char key;
	struct Words *left, *right;
	int count;
} typedef Words;

int Entr(char **str)
{
	int i=0;
	char ch=0;
	*str=(char*)malloc(sizeof(char));

	for(i=1; ch = getchar(); i++)
	{
		if(ch!='\n')
		{
			*str=(char*)realloc(*str,sizeof(char)*(i+1));
			str[0][i-1]=ch;
		}else
			if(i!=1)	break;
			else i--;
	}
	i--;
	str[0][i]=0;

	return i;
}

void Add(struct Words *Head,struct Words *Current)
{
	int i;
	struct Words *prev;

	if(Head == Current || Head == NULL)	return;
	while(Head!=NULL)
	{
		prev=Head;
		i=(Head->length > Current->length) ? 1:0;
		if(i == 1) {		Head=Head->left;}
		else			{		Head=Head->right;}
	}
	if(i == 1) prev->left=Current;
	else			prev->right=Current;
	return;
}
```

```
Words *CreateWords(int *count)
{
	Words *Head=(Words*)malloc(sizeof(Words));
	if(Head==NULL){ printf("Error");	getchar();};
	Head->left=NULL;
	Head->right=NULL;
	Head->word=NULL;
	Head->key=NULL;
	Head->count=*count;
	Head->word=(char*)malloc(sizeof(char));
	if(Head==NULL){ printf("Error");	getchar();};
	*count+=1;
	return Head;
}

void GetWords(Words **data, int *Line, int *Col)
{
	char c;
	int i=0, max=0,n=0;
	int count=1;
	Words *Head=CreateWords(&count);
	Words *Dump=Head;

	FILE *Output=fopen("Output.txt", "rb+");
	if( !Output ){	printf("File didn't found"); getchar(); exit(EXIT_SUCCESS);}

	while(fscanf(Output, "%c",&c)!= EOF)
	{
		i++;
		Dump->word=(char*)realloc(Dump->word, sizeof(char)*i);

		if(c=='\n')
		{
			Dump->word[i-1]=0;
			Dump->length=i-2;
			if((Dump->length)>max)	max=Dump->length;
			Add(Head, Dump);
			Dump=CreateWords(&count);
			i=0;
			n++;
		}else	Dump->word[i-1]=c;
	}
	if(i!=0)
	{
		Dump->word=(char*)realloc(Dump->word, sizeof(char)*(i+1));
		Dump->word[i-1]=c;
		Dump->word[i]=0;
		Dump->length=i;
		if((Dump->length)>max)	max=Dump->length;
		Add(Head, Dump);
		i=0;
		n++;
	}else{
		free(Dump->word);
		free(Dump);
	}
	*Col=max;
	*Line=n;
	*data=Head;
}
//-----------------

int PasteX(Protoblast **Pole, Words *Head, int i, int j, int count)
{
	if(Pole[i][j+count].word==Head->word[count])
	{

		if(Pole[i][j+count].logic!=1)  return 1;
	}
```

```
	return 0;
}

int PasteY(Protoblast **Pole, Words *Head, int i, int j, int count)
{
	if(Pole[i+count][j].word==Head->word[count])
	{

		if(Pole[i+count][j].logic!=2)  return 1;
	}
	return 0;
}

int ZeroPasteX(Protoblast **Pole, Words *Head, int i, int j, int count)
{
	if(Pole[i][j+count].word==0)
	{
		if(Pole[i][j+count].logic!=1) return 1;
	}
	return 0;
}

int ZeroPasteY(Protoblast **Pole, Words *Head, int i, int j, int count)
{
	if(Pole[i+count][j].word==0)
	{
		if(Pole[i+count][j].logic!=2)  return 1;
	}
	return 0;
}

void ToX(Words *Head,Protoblast **Pole, struct Resault *Stat, int Line, int Col)
{
	int i,j,count,key=0;

	for(i=0; i<Line; i++)
	{
		count=0;
		for(j=0; j<Col; j++)
		{
			if((Head->length+j) <= Col)
			{
				for(count=0; count < (Head->length); count++)
				{
					if(ZeroPasteX(Pole, Head, i, j, count)) ;
					else{
						if(PasteX(Pole, Head, i, j, count))
							key++;
						else	break;
					}
				}
				if(count==Head->length)
				{
					if(Stat->Rate<key)
					{
						Stat->Col=j;
						Stat->Line=i;
						Stat->Posit=1;
						Stat->Rate=key;
					}
				}
				key=0;
			}
		}
	}
}

void ToY(Words *Head,Protoblast **Pole, struct Resault *Stat, int Line, int Col)
{
	int i,j,count,key=0;
```

```
	for(j=0; j<Col; j++)
	{
		count=0;
		for(i=0; i<Line; i++)
		{
			if((Head->length+i) <= Line)
			{
				for(count=0; count<Head->length; count++)
				{
					if(ZeroPasteY(Pole, Head, i, j, count)) ;
					else{
						if(PasteY(Pole, Head, i, j, count))
							key++;
						else	break;
					}
				}
				if(count==Head->length)
				{
					if(Stat->Rate<key)
					{
						Stat->Col=j;
						Stat->Line=i;
						Stat->Posit=2;
						Stat->Rate=key;
					}
				}
				key=0;
			}
		}
	}
	return;
}

void Optimiation(Words *Head,Protoblast **Pole, int *sta, int Line, int Col, int key)
{
	int i;
	int Second=0;
	struct Resault Stat;

	if(*sta>-1)
	{
		if(!Head->key)
		{
			if(key==1)
			{
				ToX(Head, Pole, &Stat, Line, Col);
				ToY(Head, Pole, &Stat, Line, Col);
			}else{
				Stat.Col=0;
				Stat.Line=Head->count-1;
				Stat.Posit=1;
				Stat.Rate=1;
			}
		}else{
			return;
		}

		if(Stat.Rate)
		{
			if(Stat.Posit==1)
			{
				for(i=0; i<Head->length; i++)
				{
					Pole[Stat.Line][Stat.Col+i].word=Head->word[i];
					if(i==0 || Pole[Stat.Line][Stat.Col+i].color==-1)
							Pole[Stat.Line][Stat.Col+i].color=-1;
					else		Pole[Stat.Line][Stat.Col+i].color=Head->count;
					Pole[Stat.Line][Stat.Col+i].logic=Stat.Posit;
				}
```

```
                        }else{
                                for(i=0; i<Head->length; i++)
                                {
                                        Pole[Stat.Line+i][Stat.Col].word=Head->word[i];
                                        if(i==0 || Pole[Stat.Line+i][Stat.Col].color==-1)
                                                        Pole[Stat.Line+i][Stat.Col].color=-1;
                                        else            Pole[Stat.Line+i][Stat.Col].color=Head->count;
                                        Pole[Stat.Line+i][Stat.Col].logic=Stat.Posit;
                                }
                        }
                        Head->key=1;
                }else{
                        if(Second)      printf("Error");
                }
        }else{
                *sta=1;
                Line/=2;
                Col=Col/2-Head->length/2;
                for(i=0; i<(Head->length); i++)
                {
                        Pole[Line][Col+i].word=Head->word[i];
                        Pole[Line][Col+i].color=-1;
                        Pole[Line][Col+i].logic=1;
                }
                Head->key=1;
        }
}

void WriteToMem(struct Words *Head, Protoblast **Pole, int *sta, int Line, int Col, int key)
{
        if(Head == NULL) return;

        WriteToMem(Head->right, Pole, sta, Line, Col, key);
//-----------------
        //Juntos(Head, Pole);
        Optimiation(Head, Pole, sta, Line, Col, key);
//-----------------
        WriteToMem(Head->left, Pole, sta, Line, Col, key);
        return;
}

char destr(Words *Head)
{
        char c=1;
        if(Head==NULL)   return c;
        c=Head->key;
        c*=destr(Head->left);
        c*=destr(Head->right);

        if(Head->word!=NULL)       free(Head->word);

        free(Head);
        return c;
}

int main()
{
        Protoblast **Pole;
        int Col, Line,x,y,i, length=0;
        Words *Head;
        int sta=0,dump=0;
        char *str;

                GetWords(&Head, &Line, &Col);

                Line*=2;
                Col*=2;

                Pole=(Protoblast**)malloc(sizeof(Protoblast*)*Line);
                for(i=0; i!=Line; i++)
```

```
		{
			Pole[i]=(Protoblast*)calloc(1,sizeof(Protoblast)*Col);
		}

		while(length!=1 && (i!=1 || i!=2))
		{
			printf("Choose: 1-Smin, 2-Smax\n");
			length=Entr(&str);
			printf("\n");
			i=atoi(str);
			free(str);
		}

		if(i==1)
		{
			sta=-1;
			WriteToMem(Head, Pole, &sta, Line, Col, 1);
			WriteToMem(Head, Pole, &sta, Line, Col, 1);
		}//Smin
		if(i==2)
		{
			sta=1;
			WriteToMem(Head, Pole, &sta, Line, Col, 2);
		}


		if(destr(Head))
		{
			MainInitGraphics(Pole, Line, Col);
		}else{
			for(i=0; i!=Line; i++)
			{
				free(Pole[i]);
			}
			free(Pole);
			printf("Error");
			getchar();
		}
	}
```

<<<<<<<<<<<<<<<<<<<<<<<<<<<<==========>>>>>>>>>>>>>>>>>>>>>>>>>>>>

----------«GraphicalPart.h»----------

```
#include "glut.h"
//#include "vld.h"

struct Kletka
{
	char word;
	char logic;
	char color;
}typedef Protoblast;

struct Gam
{
	float r,g,b;
}typedef ColorRGB;

void MainInitGraphics(Protoblast **str, int Line, int Col);
```

<<<<<<<<<<<<<<<<<<<<<<<<<<<<==========>>>>>>>>>>>>>>>>>>>>>>>>>>>>

----------«GraphicalPart.c»----------

```
#include "GraphicalPart.h"
#include <malloc.h>
#include <windows.h>
//#include <vld.h>

#define QuadLength 35
```

```
Protoblast **ViewStr;
int x,y,iPresKey=0, LocalY;

void OptimumResize(int Line, int Col,int *ByX, int *ByY)
{
	int i,j;
	int counter=0;

//At-X
	for(i=0; i<Line; i++)
	{
		for(j=0; j<Col; j++)
		{
			counter+=ViewStr[i][j].logic;
		}
		if(counter==0)
		{
			*ByX-=1;
			for(j=0; j<Col; j++)
			{
				ViewStr[i][j].word=ViewStr[i][j].color=-2;
			}
		}else counter=0;
	}
//At-Y
	for(i=0; i<Col; i++)
	{
		for(j=0; j<Line; j++)
		{
			counter+=ViewStr[j][i].logic;
		}
		if(counter==0)
		{
			*ByY-=1;
			for(j=0; j<Line; j++)
			{
				ViewStr[j][i].word=ViewStr[j][i].color=-2;
			}
		}else counter=0;
	}
}

void SpecialInput(int key, int posX, int posY)
{
	int i;
	iPresKey=key;
	switch(key)
	{
	case GLUT_KEY_DOWN:
	{
		glutDestroyWindow(1);
		for(i=0; i<y; i++)
		{
			free(ViewStr[i]);
		}
		free(ViewStr);
		exit(0);
	}
	break;
	}
	glutPostRedisplay();
}

void IfPress()
{
	int i;
	switch(iPresKey)
	{
	case GLUT_KEY_UP:
```

```
                glColorMask(GL_FALSE, GL_TRUE, GL_TRUE, GL_TRUE);
        break;
        case GLUT_KEY_LEFT:
                glColorMask(GL_TRUE, GL_FALSE, GL_TRUE, GL_TRUE);
        break;
        case GLUT_KEY_RIGHT:
                glColorMask(GL_TRUE, GL_TRUE, GL_FALSE, GL_TRUE);
        break;
        }
}

void displayDrow()
{
        int posX=0, posY=(LocalY+1)*QuadLength, iCountX, iCountY;
        glClear(GL_COLOR_BUFFER_BIT);

        for(iCountY=0; iCountY<y; iCountY++)
        {
                for(iCountX=0; iCountX<x; iCountX++)
                {
                        if(ViewStr[iCountY][iCountX].color==-2)         continue;
                        posX+=QuadLength;

                        if(ViewStr[iCountY][iCountX].color!=-1)
                                        glColorMask(GL_TRUE, GL_TRUE, GL_TRUE, GL_TRUE);
                        else        glColorMask(GL_TRUE, GL_FALSE, GL_TRUE, GL_TRUE);
                        glBegin(GL_QUADS);
                                //Drow Quad
                        if(ViewStr[iCountY][iCountX].color!=0)          glColor4f(0.9, 0.7, 0.0,1.0);   //Choose Color
                        else                                                                    glColor4f(0.0, 0.0,
0.0, 0.0);
                                glVertex2i(posX+1, posY+QuadLength-1);                                  //
        LeftUp
                                if(ViewStr[iCountY][iCountX].color!=0)          glColor4f(0.7, 0.9, 0.0,1.0);   //Choose
Color
                        else                                                                    glColor4f(0.0, 0.0,
0.0, 0.0);
                                glVertex2i(posX+1, posY+1);
        //              LeftDown
                                if(ViewStr[iCountY][iCountX].color!=0)          glColor4f(0.9, 0.7, 0.0,1.0);   //Choose
Color
                        else                                                                    glColor4f(0.0, 0.0,
0.0, 0.0);
                                glVertex2i(posX+QuadLength-1, posY+1);                                  //
        RightDown
                                if(ViewStr[iCountY][iCountX].color!=0)          glColor4f(0.9, 0.7, 0.0, 1.0);  //Choose
Color
                        else                                                                    glColor4f(0.0, 0.0,
0.0, 0.0);
                                glVertex2i(posX+QuadLength-1, posY+QuadLength-1);   //          RightUp
                        glEnd();
                        if(ViewStr[iCountY][iCountX].word==0)           continue;
        //There ara nothing

                                glBegin(GL_LINE_STRIP);
                                //Drow frame
                                        glColor4f(0.0, 0.0, 0.0, 0.0);
                                        glVertex2i(posX, posY+QuadLength);                              //          LeftUp
                                        glVertex2i(posX, posY);                                                                 //
        LeftDown
                                        glVertex2i(posX+QuadLength, posY);                              //          RightDown
                                        glVertex2i(posX+QuadLength, posY+QuadLength);       //          RightUp
                                glEnd();

                                glColor4f(0.0, 0.0, 0.0, 0.0);
                                glRasterPos2i(posX+8, posY+8);
                                glutBitmapCharacter(GLUT_BITMAP_9_BY_15, ViewStr[iCountY][iCountX].word); //Drow text
                }
                if(posX!=0){
                        posX=0;
```

```
                posY-=QuadLength;
            }
            //posY-=QuadLength;
        }
        glFlush();
}

void MainInitGraphics(Protoblast **str, int Line, int Col)
{
        int iScreenHeight, iScreenWidth;
        int iCalX, iCalY;
        int LocalX;
        char i,c;

        FreeConsole();

        ViewStr=str;
        x=LocalX=Col;
        y=LocalY=Line;

        OptimumResize(Line, Col, &LocalX, &LocalY);

        iCalX=(LocalX+1)*QuadLength;
        iCalY=(LocalY+3)*QuadLength;

        iScreenWidth=glutGet(GLUT_SCREEN_WIDTH);
        iScreenHeight=glutGet(GLUT_SCREEN_HEIGHT);

        glutInitWindowSize(iCalX, iCalY);                                   //Window
Size
        glutInitWindowPosition((iScreenWidth-iCalX)/2, (iScreenHeight-iCalY)/2);  //Windows position
        glutCreateWindow("Crossword");                                      //Name of
the Window
        glClearColor(0.0,0.0,0.0,0.0);                                      //Start
Color
        gluOrtho2D(0.0, iCalX, 0.0, iCalY);                                 //Range of
coordinates

        glutSpecialFunc(SpecialInput);

        glutDisplayFunc(displayDrow);                                       //In this Funcion we era draw

        glutMainLoop();                                                     //i don't know why need this function
}
```