Universidade Nova de Lisboa

Faculdade de Ciências e Tecnologia

Departamento de Informática

Cadeira de Programação 2

Relatório sobre o 4º Guião

Conteúdo:

Faz parte deste relatório, códigos de funções mais importantes criadas para este projecto.

Constam neste relatório, indicações sobre as tasks A, B, C, G, H e I, entregues pelo sistema MOOSHAK.

TASK A

Objectivo:

Era objectivo desta task, a criação de um sistema para ler uma string de comandos direccionais e indicar quantos lados tinha o objecto criado através da implementação desses comandos. Era também necessário indicar se o objecto era fechado ou aberto. Poligono ou Não Poligono.

Funções:

int GetSides()

Serve a presente função para calcular o numero de lados do objecto criado.

```
int GetSides()
{
	int lados = 0;
	for(int i=0; i<=tamanho_da_cadeia-1; i++)
	{
		if(comandos[i] != comandos[i+1])
			lados++;
	}
	if(comandos[0] == comandos[tamanho_da_cadeia-1])
	{
		lados--;
	}
	return lados;
}
```

Esta função do tipo int, retorna o número de lados da figura.

bool IsClosed()

Serve a presente função para indicar se o objecto era fechado ou aberto.

```
bool IsClosed()
{
	int u_ = 0;
	int r_ = 0;
	int d_ = 0;
	int l_ = 0;
	for(int i=0; i<static_cast<int>(comandos.size()); i++)
	{
```

```
                        if(comandos[i] == 'u')
                        {
                                u_++;
                        }
                        if(comandos[i] == 'r')
                        {
                                r_++;
                        }
                        if(comandos[i] == 'd')
                        {
                                d_++;
                        }
                        if(comandos[i] == 'l')
                        {
                                l_++;
                        }
                }
                if((l_==r_) && (u_==d_) && u_!=0 && d_!=0 && l_!=0 && r_!=0)
                        return true;
                else
                        return false;
        }
```

Esta função retorna true para Poligono e false para Não Poligono.

TASK C

Objectivo:

Era objectivo desta TASK, calcular a àrea do objecto caso este fosse poligono.

Funções:

double AreaPoligono()

Serve a presente função para calcular a àrea do poligono.

```
double AreaPoligono()
{
        double valor = 0;
        for(int i=0; i<static_cast<int>(Vertices.size())-1; i++)
        {
                valor += ( (Vertices[i].X()*Vertices[i+1].Y()) - (Vertices[i+1].X()*Vertices[i].Y()) );
        }
        valor = valor/2;
        return valor;
}
```

Esta função foi baseada na fórmula indicada no guiao do projecto.

$$½(\sum(x*y1-x1*y))$$

De forma a obter resultados correctos no output, foi usada a seguinte técnica de programação.

```
Utilities u;
u.Read(std::cin);
u.GetVertices();
double area=u.AreaPoligono();
if(u.IsPoligono())
{
```

```
            if(area>=1)
                std::cout << static_cast<int>(area) << std::endl;
            else
                std::cout << static_cast<int>((area*-1)) << std::endl;
    }
    else
            std::cout << "NO" << std::endl;
```

Era necessário fazer o modulo da àreas dos poligonos para que todas viessem positivas.

TASK G

Objectivo:

Era objectivo desta task, a impressão na consola de uma matriz constituidas por 1's e 0's.

void Matriz_FILL()

Serve a presente função para preencher a matriz.

```
    void Matriz_FILL()
    {
            int x = 0;
            int y = 0;
            std::vector<int> temp;
            for(int j=0; j<=C; j++)
            {
                temp.push_back(0);
            }
            for(int i=0; i<=R; i++)
            {
                Matriz.push_back(temp);
            }
            for(int i=0; i<N; i++)
            {
                Matriz[y][x] = 1;
                x = (2+x)%C;
                y = (2+y)%R;
            }
    }
```

O preenchimento da matriz estava sujeito a duas funções, indicadas no guião do projecto.

$$x=(2+x)\%C$$
$$y=(2+y)\%R$$

void Matriz_WRITE()

Serve a presente função para escrever na consola a matriz completa. Se o tamanho desta ultrapassar os 40 campos, entao serão impressos na consola os valores para o total de 0's e de 1's na matriz criada.

```
    void Matriz_WRITE()
    {
            if(!(C <= 20 && R <= 20))
            {
```

```
                int uns = 0;
                int zeros = 0;
                for(int i=0; i<R; i++)
                {
                        for(int j=0; j<C; j++)
                        {
                                if(Matriz[i][j] == 1)
                                        uns++;
                                else
                                        zeros++;
                        }
                }
                std::cout << zeros << " " << uns << std::endl;
        }
        else
        {
                for(int i=0; i<R; i++)
                {
                        for(int j=0; j<C; j++)
                        {
                                std::cout << Matriz[i][j];
                        }
                        std::cout << std::endl;
                }
        }
}
```

TASH H

Objectivo:

Esta task tinha como objectivo o calculo dos pontos vizinhos aos pontos dados pelo utilizador.

std::vector<mas::Point> VIZINHOS(mas::Point p)

Serve a presente função para obter um vector com todos os pontos vizinhos ao ponto p, passado como argumento para a presente função.

```
std::vector<mas::Point> VIZINHOS(mas::Point p)
{
        int x_ = static_cast<int>(p.X());
        int y_ = static_cast<int>(p.Y());
        std::vector<mas::Point> vizinhos;
        for(int i=-1; i<=1; i++)
        {
                for(int j=-1; j<=1; j++)
                {
                        if(((x_+i)>=0 && (x_+i)<C) && ((y_+j)>=0 && (y_+j)<R))
                        {
                                if(Matriz[y_+j][x_+i] == 1)
                                {
                                        mas::Point t;
                                        t.Set(x_+i, y_+j);
                                        vizinhos.push_back(t);
                                }
                        }
                }
        }
        return vizinhos;
}
```

TASK I

Objectivo:

Esta função apenas servia para obter vizinhos de pontos, para mais do que um conjunto de pontos dados pelo utilizador.

Era necessário nesta task que todos os conjuntos com mais do que 10 pontos, apenas apresentassem os 10 primeiros, indicando como 11º elemento um conjunto de reticencias (“...”).

TASK_I()

Serve a presente função para obter o resultado final desejado.

```
void TASK_I()
{
	Utilities u;
	u.MATRIZREAD(std::cin);
	int S;
	std::cin >> S;
	std::vector <std::set<mas::Point> > TodosVizinhos;
	for(int i=0; i<S;i++)
	{
		u.MATRIXREAD_PONTOS(std::cin);
		std::set<mas::Point> VizinhosAgrupados;
		for(int k=0;k<static_cast<int> (u.pontos.size());k++)
		{
			std::vector <mas::Point> PontosVizinhos = u.VIZINHOS(u.pontos[k]);
			for(int l=0;l<static_cast<int>(PontosVizinhos.size());l++)
			VizinhosAgrupados.insert(PontosVizinhos[l]);
		}
		TodosVizinhos.push_back(VizinhosAgrupados);
	}

	for(int i=0; i<S;i++)
	{
		std::cout << static_cast<int>(TodosVizinhos[i].size());
		if(static_cast<int>(TodosVizinhos[i].size())!=0)
		{
			int ctr = 0;
			for(std::set<mas::Point>::const_iterator j = TodosVizinhos[i].begin(); j != TodosVizinhos[i].
end(); j++)
			{
				if(ctr<10)
				{
					std::cout << " " << (*j).X() << " " << (*j).Y();
					}
					else
					{
						std::cout << " " << "...";
						break;
					}
				ctr++;
			}
		}
		std::cout << std::endl;
	}
}
```

CONCLUSAO:

Neste guião, foram usadas pela primeira vez MAPAS. Estes tornaram-se mais eficaz em relação às comuns matrizes de vectores.

Os MAPAS usando uma forma de pesquiza em àrvore, faziem com que uma simples pesquiza por numeros de chaves repetidas seja mais rápido do que numa matriz de vectores.

Devido a problemas de WA não consegui completar as tasks D (e derivadas), J e K.

De qualquer forma constam dos ficheiros entregues pelo sistema MOOSHAK, as funções que criei para as TASKS D e J.

Autor
João Rico
17488
Aluno da Licenciatura em Engenharia Informática