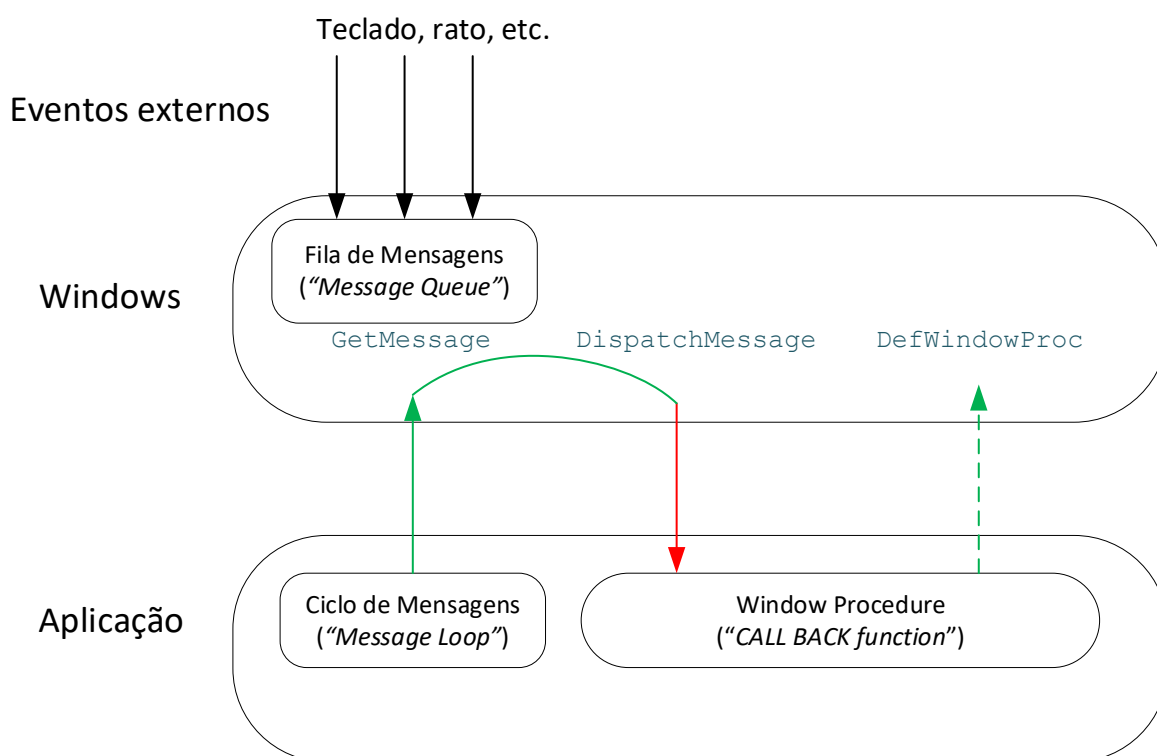


## INTRODUÇÃO

Neste trabalho pretende-se que o resultado do programa seja mostrado numa janela do Windows. Para isso há que esclarecer primeiramente alguns aspetos relativos à programação com janelas. Um programa do Windows® ou de outro sistema operativo semelhante é baseado em eventos (*“event-driven”*), isto quer dizer que o programa deve responder a eventos que acontecem de forma assíncrona, como por exemplo *clicks* ou movimentos no botão do rato, teclas pressionadas, etc.

A figura seguinte ilustra esquematicamente o que se pretende afirmar.



O Windows® apanha os vários eventos do teclado, do rato, etc., cada evento é convertido para uma mensagem sendo esta despachada de seguida para a janela apropriada. Por exemplo todas as mensagens do teclado vão diretamente para a janela que tem o **foco** (janela ativa). As mensagens do rato são despachadas de acordo com a posição do cursor, normalmente estas mensagens vão para a janela que está diretamente debaixo do cursor, etc.

Uma consequência desta situação é a obrigatoriedade na função *main* de cada aplicação existir um ciclo de mensagens:

```
int main()
{
    . . .
    MSG msg ;
    while( GetMessage( &msg, 0, 0, 0 ) > 0 )
    {
        DispatchMessage(&msg);
        . . .
    }
}
```

A outra parte importante de cada aplicação em Windows é a função “CALLBACK” que se define e que o sistema operativo chama quando está a despachar as mensagens que chegam e que é responsável por parte de cada aplicação de tratar apenas as mensagens que pretender.

```
LRESULT CALLBACK WindowProcedure
(HWND hwnd, unsigned int message, WPARAM wParam, LPARAM lParam)
{
    switch (message)
    {
        case WM_DESTROY:
            PostQuitMessage (0);
            return 0;
    }
    return DefWindowProc (hwnd, message, wParam, lParam );
}
```

Que como se verifica por este exemplo a aplicação apenas terminará quando um utilizador clicar na cruz da janela, iniciando uma mensagem do tipo WM\_DESTROY e que provocará que esta função devolva o valor zero o que fará com que o ciclo de mensagens da função *main* termine.



## PARTE 1

Pretende-se implementar um programa que desenhe círculos. Quando se clica com o rato na nova janela criada deve aparecer um círculo com uma determinada cor à escolha. Quando se clicar em cima de um círculo este deve desaparecer da janela.

Para que se possa trabalhar em janelas com o sistema operativo Windows® é necessário definir uma classe que representa as janelas típicas do SO Windows®.

**Classe Window** – representa uma janela do S.O. Windows®

Ficheiro Window.h

```
#pragma once
#include <Windows.h>
#include "Point.h"

class Window {
private:
    HWND windowId;
    Point cur_coord;
    bool bClick;
    static Window* object;
    static Window* GetObject();

public:
    Window();
    ~Window() { };
    bool Create(const char* sTitle);
    static LRESULT CALLBACK WindowProc (HWND wndId,
                                         unsigned int msg,
                                         WPARAM wp, LPARAM lp);

    HWND GetWindowId()    { return windowId; }
    bool HasClicked()     { return bClick; }
    Point GetPoint()      { return cur_coord; }
};
```



## Ficheiro Window.cpp

```
#include "Window.h"
Window* Window::object = NULL; // Objecto estático da classe Window que é utilizado
                                // dentro da função CALLBACK

Window::Window()
{
    windowId = NULL;
    object = this; // Uma vez criado o objecto estático é o próprio objecto
    bClick = false;
}

Window* Window::GetObject()
{
    return object;
}

bool Window::Create(const char* sTitle)
{
    WNDCLASSEX wndclass = { sizeof(WNDCLASSEX), CS_DBLCLKS, WindowProc,
                            0, 0, GetModuleHandle(0), LoadIcon(0, IDI_APPLICATION),
                            LoadCursor(0, IDC_ARROW), HBRUSH(CTLCOLOR_STATIC+1),
                            0, "myclass", LoadIcon(0, IDI_APPLICATION) };
    if( RegisterClassEx(&wndclass) )
    { // Função do SO Windows que cria janelas
        windowId = CreateWindowEx( 0, "myclass", sTitle,
                                    WS_OVERLAPPEDWINDOW, CW_USEDEFAULT, CW_USEDEFAULT,
                                    CW_USEDEFAULT, CW_USEDEFAULT, 0, 0, GetModuleHandle(0), 0 );
        if(windowId)
        {
            ShowWindow( windowId, SW_SHOWDEFAULT );
            return true;
        }
    }
    return false;
}

LRESULT CALLBACK Window:: WindowProc (HWND wndId, unsigned int msg,
                                       WPARAM wp, LPARAM lp)
{
    Window* window = GetObject();
    window->bClick = false;
    switch(msg) {
        case WM_DESTROY:
            PostQuitMessage(0) ;
            return 0;
        case WM_KEYDOWN:
            if (wp == VK_ESCAPE) {
                PostQuitMessage(0) ;
                return 0;
            }
            break;
        case WM_LBUTTONDOWN:
            window->bClick= true;
            window->cur_coord = Point(LOWORD(lp), HIWORD(lp));
            break;
        default:
            break;
    }
    return DefWindowProc(wndId, msg, wp, lp);
}
```

A função CALLBACK *WindowProc* que é passada para o SO.

Na declaração desta classe verifica-se que existe uma função que é declarada como estática (“static”) que é a forma possível de uma função numa classe poder ser chamada como uma função *CALLBACK* por parte do sistema operativo (`static LRESULT CALLBACK WindowProc`).

Os atributos desta classe são apenas aqueles que são necessários para a execução do programa proposto, i.e.:

- `windowId` – É o identificador da janela que se vai criar.
- `cur_coord` – Representa as coordenadas do cursor quando existir um click do rato em cima da janela.
- `bClick` – Variável booleana que indica se existiu ou não um click do rato em cima da janela.

A razão de existir um apontador para a classe *Window* e ter sido declarado como estático tem a ver a função *WindowProc* e o poder alterar valores do próprio objeto.

Como se quer desenhar círculos é necessário definir uma classe que represente círculos.

**Classe Circle** – É representado por um ponto (centro) e um raio.

Ficheiro *Circle.h*

```
#pragma once
#include <windows.h>
#include "Point.h"

class Circle{

private:
    Point center; // coordenadas do centro
    int radius;   // raio do círculo

public:
    Circle(); // construtor por omissão
    Circle(Point c, int r);

    void Draw(HWND wndId, long color); // Desenha graficamente o círculo
    bool Contains (Point p);           // verifica se o ponto está dentro do círculo
};
```

A implementação do método **Draw** da classe *Circle* é a seguinte:

## Ficheiro Circle.cpp

```
#include "Circle.h"
...
void Circle::Draw(HWND wndId, long color)
{
    if (wndId != NULL)
    {
        HDC DrawHDC = GetDC(wndId);
        // penstyle, width, color
        HPEN hNPen = CreatePen(PS_SOLID, 2, color);
        HPEN hOPen = (HPEN)SelectObject(DrawHDC, hNPen);
        HBRUSH hOldBrush;
        HBRUSH hNewBrush;
        hNewBrush = CreateSolidBrush(color);
        hOldBrush = (HBRUSH)SelectObject(DrawHDC, hNewBrush);
        Ellipse(DrawHDC, center.GetX()-radius, center.GetY()+radius,
                center.GetX()+radius, center.GetY()-radius);
        DeleteObject(SelectObject(DrawHDC, hOPen));
        DeleteObject(SelectObject(DrawHDC, hOldBrush));
        ReleaseDC(wndId, DrawHDC);
    }
}
```

Como se verifica pela definição das duas classes anteriores, existe uma outra classe que é necessário também definir, a classe **Point** que representa um ponto num plano XY.

**Classe Point** – representa um ponto num plano XY

## Ficheiro Point.h

```
#pragma once
class Point{
private:
    int x, y; // coordenadas do ponto
public:
    Point(); // construtor por omissão
    Point(int x0, int y0); //construtor para atribuição de novos valores

    void SetX(int new_x); // nova coordenada X
    int GetX(); // devolve a coordenada X
    void SetY(int new_y); // nova coordenada Y
    int GetY(); // devolve a coordenada Y
    float GetDistance(Point p2); // calcula a distancia a um ponto p2
};
```

É necessário completar o ficheiro Circle.cpp com as implementações das restantes funções da classe **Circle**. É também necessário criar o ficheiro Point.cpp com as implementações de todas as funções da classe **Point**.

A função *main* deve ser parecida com a que se indica de seguida:

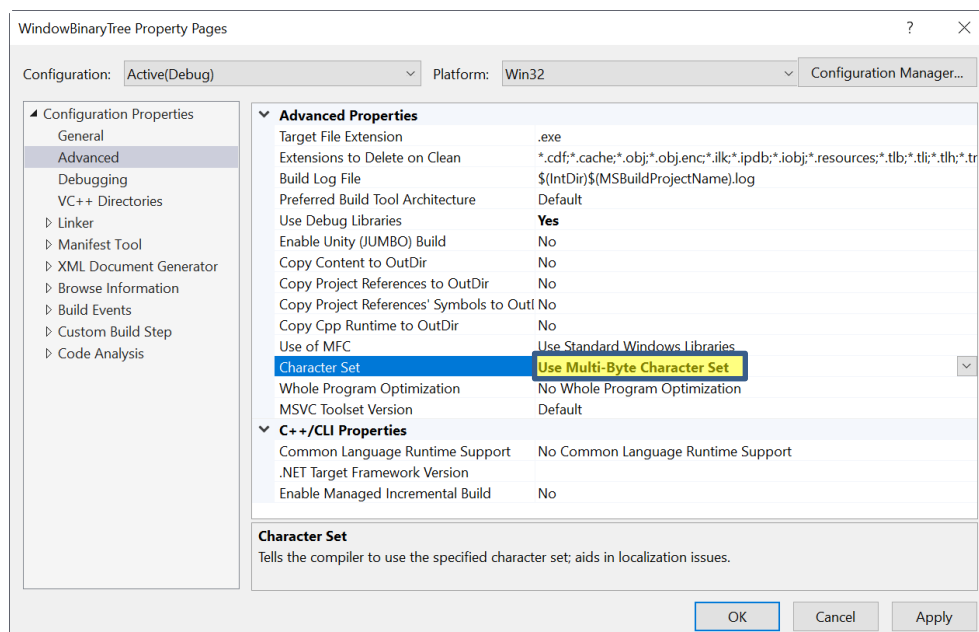
```
#include<stdio.h>
#include"Window.h"
#include"Circle.h"

#define VERDE RGB(0,255,0)
#define AZUL RGB(0,0,255)
#define VERMELHO RGB(255,0,0)

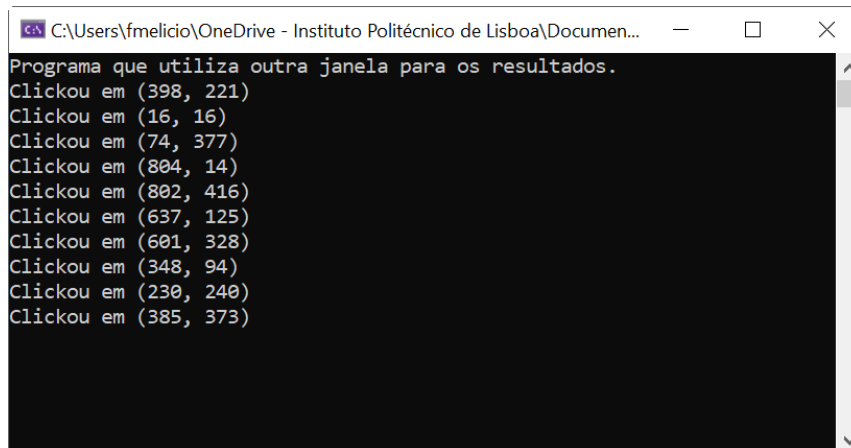
int main() {
    Window window;
    HWND windowId;
    printf("Programa que utiliza outra janela para os resultados.\n");
    if ( window.Create("My Window")) {
        if ((windowId=window.GetWindowId()) != NULL) {
            MSG msg ;
            while( GetMessage( &msg, 0, 0, 0 ) ) {
                DispatchMessage(&msg);
                if (window.HasClicked()){
                    printf("Clickou em (%d, %d)\n", window.GetPoint().GetX(),
                        window.GetPoint().GetY());

                    . . . // (Para Completar)
                }
            }
        }
    }
}
```

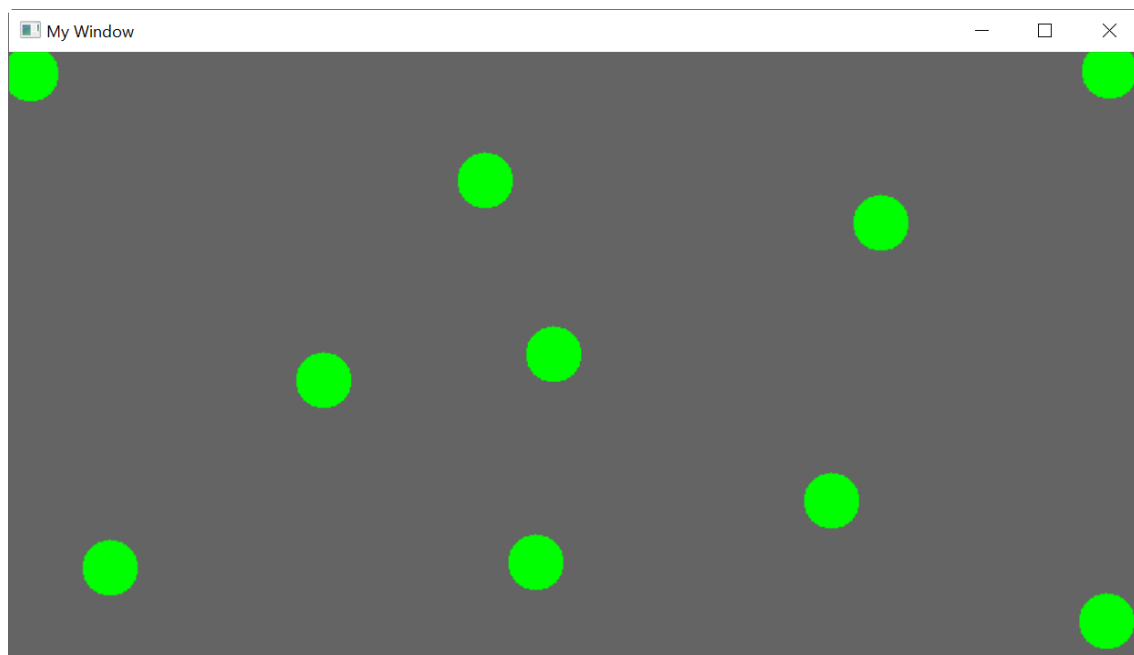
Selecione as propriedades do projeto e na página **Advanced** selecione corretamente o **Character Set** para que este programa funcione corretamente.



O resultado deste programa deve ser algo de semelhante ao que se indica na figura seguinte.



```
Programa que utiliza outra janela para os resultados.  
Clickou em (398, 221)  
Clickou em (16, 16)  
Clickou em (74, 377)  
Clickou em (804, 14)  
Clickou em (802, 416)  
Clickou em (637, 125)  
Clickou em (601, 328)  
Clickou em (348, 94)  
Clickou em (230, 240)  
Clickou em (385, 373)
```



## PARTE 2

Pretende-se que elabore um programa que utilize as classes definidas na parte 1 e que desenhe uma linha entre cada dois círculos de forma alternada, i.e., que desenhe uma linha entre o círculo 1 e 2, entre o 3 e 4 e assim sucessivamente. Conjuntamente deve ir indicando na janela de comando o valor atual da soma dos comprimentos das linhas.

Para isso é necessário definir a classe Line.



## Ficheiro Line.h

```
#pragma once
#include <windows.h>
#include "Point.h"

class Line{
private:
    Point pi, pf;    // pi - Initial point, pf - Final point
public:
    Line();    // Default constructor
    Line(Point p1, Point p2);

    void Draw(HWND wndId, long color); // Draw graphically a line between pi and pf
    double GetLength ();    // Length of the line
};
```

Cuja implementação da função **Draw** é a seguinte:

## Ficheiro Line.cpp

```
#include "Line.h"
...
void Line::Draw(HWND wndId, long color)
{
    if (wndId != NULL)
    {
        if (wndId != NULL)
        {
            HPEN hOPen;
            // penstyle, width, color
            HPEN hNPen = CreatePen(PS_SOLID, 2, color);
            HDC DrawHDC = GetDC(wndId);
            hOPen = (HPEN)SelectObject(DrawHDC, hNPen);
            // starting point of line
            MoveToEx(DrawHDC, pi.GetX(), pi.GetY(), NULL);
            // ending point of line
            LineTo(DrawHDC, pf.GetX(), pf.GetY());
            DeleteObject(SelectObject(DrawHDC, hOPen));
            ReleaseDC(wndId, DrawHDC);
        }
    }
}
```

O resultado gráfico deve ser semelhante ao que está indicado na figura seguinte.

