Примеры:

```
    [STAThread]
        static void Main(string[] args)
        {
          Console.WriteLine ("Тест нажмите Ентер.....");
          Console.ReadKey(true);
          Click_Enter(null, null); // вот она
        }

     static void Click_Enter(object sender, KeyEventArgs e)
        {
            if (e.KeyCode == Keys.Enter)  // Ентер срабатывала
            {
                MessageBox.Show("Start Enter");
            }
        }
```

Перехват Alt+ F4

```
public Form1()
        {
            InitializeComponent();
            this.KeyPreview = true;
        }

        protected override void OnKeyDown(KeyEventArgs e)
        {
            base.OnKeyDown(e);
            if (e.KeyCode == Keys.F4&&e.Alt)
            {
                MessageBox.Show("Тест");
                e.Handled = true;
            }
        }
```

Нажатие на стрелки:

```
if (e.KeyCode == Keys.Down||e.KeyCode ==Keys.Left||e.KeyCode
==Keys.Right||e.KeyCode ==Keys.Up)
            {
                MessageBox.Show("Тест");
                e.Handled = true;
            }
```

Глобальные горячие клавиши RegisterHotKey

```
#include <windows.h>

#define HOTKEX 1

/*  Declare Windows procedure  */

LRESULT CALLBACK WindowProcedure (HWND, UINT, WPARAM, LPARAM);
```

```
/*  Make the class name into a global variable  */
char szClassName[ ] = "WindowsApp";

int WINAPI WinMain (HINSTANCE hThisInstance,
                    HINSTANCE hPrevInstance,
                    LPSTR lpszArgument,
                    int nFunsterStil)

{
    HWND hwnd;               /* This is the handle for our window */
    MSG messages;            /* Here messages to the application are saved */
    WNDCLASSEX wincl;        /* Data structure for the windowclass */

    /* The Window structure */
    wincl.hInstance = hThisInstance;
    wincl.lpszClassName = szClassName;
    wincl.lpfnWndProc = WindowProcedure;      /* This function is called by
windows */
    wincl.style = CS_DBLCLKS;                 /* Catch double-clicks */
    wincl.cbSize = sizeof (WNDCLASSEX);

    /* Use default icon and mouse-pointer */
    wincl.hIcon = LoadIcon (NULL, IDI_APPLICATION);
    wincl.hIconSm = LoadIcon (NULL, IDI_APPLICATION);
    wincl.hCursor = LoadCursor (NULL, IDC_ARROW);
    wincl.lpszMenuName = NULL;                 /* No menu */
    wincl.cbClsExtra = 0;                      /* No extra bytes after the
window class */
    wincl.cbWndExtra = 0;                      /* structure or the window
instance */
    /* Use Windows's default color as the background of the window */
    wincl.hbrBackground = (HBRUSH) COLOR_BACKGROUND;

    /* Register the window class, and if it fails quit the program */
    if (!RegisterClassEx (&wincl))
        return 0;

    /* The class is registered, let's create the program*/
    hwnd = CreateWindowEx (
           0,                   /* Extended possibilites for variation */
           szClassName,         /* Classname */
           "Windows App",       /* Title Text */
           WS_OVERLAPPEDWINDOW, /* default window */
           CW_USEDEFAULT,       /* Windows decides the position */
           CW_USEDEFAULT,       /* where the window ends up on the screen */
           544,                 /* The programs width */
           375,                 /* and height in pixels */
           HWND_DESKTOP,        /* The window is a child-window to desktop */
           NULL,                /* No menu */
           hThisInstance,       /* Program Instance handler */
           NULL                 /* No Window Creation data */
           );

    /* Make the window visible on the screen */
    ShowWindow (hwnd, SW_SHOW);

    /* Run the message loop. It will run until GetMessage() returns 0 */
    while (GetMessage (&messages, NULL, 0, 0))
    {
        /* Translate virtual-key messages into character messages */
        TranslateMessage(&messages);
        /* Send message to WindowProcedure */
        DispatchMessage(&messages);
    }
```

```
    /* The program return-value is 0 - The value that PostQuitMessage() gave
*/
    return messages.wParam;
}


/*  This function is called by the Windows function DispatchMessage()  */

LRESULT CALLBACK WindowProcedure (HWND hwnd, UINT message, WPARAM wParam,
LPARAM lParam)
{
    switch (message)                  /* handle the messages */
    {

case WM_CREATE:

     RegisterHotKey(hwnd,HOTKEX,MOD_CONTROL,VK_F12);

     break;

     case WM_HOTKEY:

if (HOTKEX == (int)wParam){MessageBoxA(hwnd,"Hi","Info",MB_OK);}

break;
        case WM_DESTROY:
            PostQuitMessage (0);       /* send a WM_QUIT to the message queue
*/
            break;
        default:                      /* for messages that we don't deal with
*/
            return DefWindowProc (hwnd, message, wParam, lParam);
    }

    return 0;
}
```

Цель

Написать очень быструю и маленькую программу, скрывающую по CTRL+F12 заданные окна. При нажатии комбинации CTRL+F10 она должна показать спрятанные окна.
Входные данные:

```
TXT Файл вида
------------
Internet Explorer
The Bat!
Visual C++
911
------------
```

Если будут найдены окна, содержащие в своем заголовке указанные строки, они будут спрятаны.
В вышеуказанном примере будут спрятаны все окна IE, окно Microsoft Visual C++, окно почтовой программы "The Bat!" и все окна, в заголовках которых содержится комбинация символов "911".

Итак, писать будем на чистом Win32 API. Создадим окно, привяжем к нему горячие клавиши. По требованию будем осуществлять перебор видимых окон в системе и в заголовке каждого будем искать заданные комбинации символов.

Опции линкера

Если ничего не предпринимать, то нам не удастся получить в итоге файл менее 32 КБ(примерно). Поэтому пишем:

```
#pragma comment(linker,"/MERGE:.rdata=.text")
#pragma comment(linker,"/FILEALIGN:512 /SECTION:.text,EWRX
                        /IGNORE:4078")
#pragma comment(linker,"/ENTRY:New_WinMain")
#pragma comment(linker,"/NODEFAULTLIB")
```

На что теперь стоит обратить особое внимание? Обычно точка входа в программу выглядит так:

```
int WINAPI WinMain(HINSTANCE hInst,HINSTANCE hPrevInst,LPSTR szCmdLine,int
nCmdShow)
```

(кстати, для Win32 приложений второй параметр всегда NULL)

Но(!)... Так как мы отключили "Runtime library", нам теперь передается в этих параметрах разный мусор. Поэтому называем точку входа не WinMain а New_WinMain, которую объявим, как void New_WinMain(void), чтобы не забыть о том, что нам ничего не передается. А параметр HINSTANCE получаем функцией GetModuleHandle(NULL). Ах да, и выходить из программы будем функцией ExitProcess.

Теперь если собрать нашу пустую программку, которая ничего делать не будет, размер ее будет 1 Кб. Но нам нужно еще дописать 3 Кб кода. Продолжим.

Чтобы все дальнейшее было понятно даже новичку в программировании под Windows, я прокомментирую все.

Объявим кое-какие константы

Это понадобится для регистрации "горячих" клавиш функцией RegisterHotKey.

```
#define HOTKEYHIDE 1
#define HOTKEYSHOW 2
```

Размер буффера, куда будет считываться заголовок окна функцией GetWindowText.

```
#define SSZZ 256
```

Размер буфера, куда будет считываться файл со стоками фильтрации (используется в объявлении char FilterStrings[MAXFIL];)

```
#define MAXFIL 1024
```

(Примечание: При желании можно сделать и выделение памяти динамически - найти файл, узнать его размер и выделить блок. Приблизительный пример:

```
// .....................
WIN32_FIND_DATA FindData;
HANDLE hFind=FindFirstFile(szFilterStringsFile,&FindData);
if (hFind!=INVALID_HANDLE_VALUE)
{
  i=(FindData.nFileSizeHigh * MAXDWORD) + FindData.nFileSizeLow;
  HGLOBAL hGA=GlobalAlloc(GMEM_ZEROINIT|GMEM_MOVEABLE,i+1);
  // (+ end-ZERO)
  if (hGA!=NULL)
  {
    LPVOID lpStrings=GlobalLock(hGA);
    DWORD dw;
    if (lpStrings!=NULL) ReadFile(hFile,lpStrings,i,&dw,NULL);
    }

  }
  FindClose(hFind);
  CloseHandle(hFile);
// ...............................
// Но так как вряд ли файл настроек у нас будет больше одного
// килобайта, я оставил статичный массив.
)
```

Массив хендлов окон (вряд ли будет у нас более 300 окон)

```
HWND aHwnd[300];
```

Кол-во инициализированных элементов в этом массиве

```
unsigned int cHwnd=0;
```

Дескрипторы окон - главное и два дочерних - кнопка "Hide" и кнопка "Edit filter strings"

```
HWND hwndMain, hwndButtonHide, hwndButtonEditFilter;
```

Тут будет что-то типа "c:\programs\winhider\winhider.settings.txt"

```
char szFilterStringsFile[MAX_PATH]="(c)2002 KMiNT21";
```

Соответственно, хендл файла с именем "что-то типа"

```
HANDLE hFile;
```

А это место, куда будем считывать все из этого файла

```
char FilterStrings[MAXFIL];
```

Функции

Обработка сообщений главного окна

```
LRESULT CALLBACK MainWndProc(HWND hwnd,UINT msg,WPARAM wParam,LPARAM lParam);
```

Функция, которая будет вызываться для каждого окна при переборе всех окон

```
static BOOL FAR PASCAL my_EnumWindowsProc(HWND hwnd, DWORD lParam);
```

Проверка наличия строки str2 в str1

```
BOOL Contain(char* str1, char* str2);
```

Скрывание с экрана очередного окна

```
inline void HideNext(HWND hwnd){ ShowWindow(aHwnd[cHwnd++]=hwnd,SW_HIDE); }
```

Возврат всех спрятанных окон на экран

```
inline void ShowAll(void) { while(cHwnd) ShowWindow(aHwnd[--cHwnd],SW_SHOW);}
```

Пройдемся по главным строкам функции NewWinMain

* Получим INSTANCE модуля. Это нам нужно для регистрации оконного класса

```
HINSTANCE hInst=GetModuleHandle(NULL);
```

* Зарегистрируем оконный класс

```
 WNDCLASS wc;
  wc.style = CS_HREDRAW|CS_VREDRAW ;
  wc.lpfnWndProc = (WNDPROC)MainWndProc;
  wc.hInstance = hInst;
  wc.hbrBackground = (HBRUSH)(COLOR_WINDOW);
  wc.lpszClassName = "CKMINT21WINDOWSHIDERPRO";
  wc.hCursor = LoadCursor(NULL,IDC_ARROW);
  wc.hIcon = LoadIcon(NULL,IDI_APPLICATION);
  wc.lpszMenuName=NULL;
  wc.cbClsExtra=0;
  wc.cbWndExtra=0;
  if (!RegisterClass(&wc)) MessageBox(0,"I can't register window
              class.","Error:",0), ExitProcess(0);
```

* Создаем главное окно приложения

```
hwndMain=CreateWindow(wc.lpszClassName,"Small windows hider!",
         WS_BORDER|WS_CAPTION|WS_SYSMENU|WS_MINIMIZEBOX,
         CW_USEDEFAULT,0,291,180, NULL, NULL, hInst, NULL);
```

И помещаем на него две кнопки. Как видим, кнопки имеют класс "BUTTON". Они являются дочерними окну hwndMain.

```
hwndButtonHide=CreateWindow("BUTTON","Hide!", WS_VISIBLE | WS_CHILD ,
               10,10,261,90, hwndMain, NULL, hInst, NULL);
ShowWindow(hwndButtonHide,SW_SHOW), UpdateWindow(hwndButtonHide);
hwndButtonEditFilter=CreateWindow("BUTTON","Edit filters",
                     WS_VISIBLE|WS_CHILD|WS_BORDER|WS_TABSTOP ,
                     10,110,261,30, hwndMain, NULL, hInst, NULL);
ShowWindow(hwndButtonEditFilter,SW_SHOW), UpdateWindow(hwndButtonEditFilter);
```

Наконец, показываем главное окно

```
ShowWindow(hwndMain,SW_SHOW), UpdateWindow(hwndMain);
```

Примечание: Так как кто-то этого может не знать, хочу отметить, что в языке C++ есть "операция следования" - запятая. Т.е. просто последовательно выполнятся обе функции

ShowWindow и UpdateWindow (как отдельный блок). В вышеуказанной строке можно было бы и просто поставить ";", а вообще иногда это помогает избавиться от огромного количества фигурных скобок {}, в тексте программы.

* Затем регистрируем в системе HotKeys. Они будут привязаны к главному окну, которому будут передавтся сообщения WM_HOTKEY.

```
RegisterHotKey(hwndMain,HOTKEYHIDE,MOD_CONTROL,VK_F12)
RegisterHotKey(hwndMain,HOTKEYSHOW,MOD_CONTROL,VK_F10)
```

* Затем считываем настройки из файла и запускаем главный цикл обработки оконных сообщений для текущего процесса.

```
MSG msg;
while(GetMessage(&msg,NULL,0,0)) TranslateMessage(&msg),
DispatchMessage(&msg);
```

Оконная процедура

```
// Тут все довольно стандартно. Делаем  switch (msg).
// ...
case WM_HOTKEY:
    if (HOTKEYSHOW == (int)wParam)
    // показываем все, что мы до этого прятали, а так же главное
    // окно программы
      ShowAll(), ShowWindow(hwnd,SW_SHOW);

    if (HOTKEYHIDE == (int)wParam)
    // Скрываем наше главное окно и запускаем перебор всех окон в
    // системе - EnumWindows. Теперь будет вызываться функция
    // my_EnumWindowsProc для каждого обнаруженного в системе окна.
      ShowWindow(hwnd,SW_HIDE), EnumWindows((int (__stdcall *)(struct
                               HWND__ *,long))my_EnumWindowsProc, 0);
    break;
// ...

// Если программу пытаются минимизировать, просто скрываем ее
// ........................
  case WM_SYSCOMMAND:
    if(SC_MINIMIZE == wParam) { ShowWindow(hwnd,SW_HIDE); return 0; }
    break;
    // Внимание, после ShowWindow(hwnd,SW_HIDE) мы пишем return 0,
    // вместо break. Почему? Да потому что не хотим, чтобы это
    // сообщение пошло дальше в систему. Мы его уже обработали
    // по-своему.
// ...
// А затем обрабатываем нажатия на кнопки.
    case BN_CLICKED:
      if (hwndButtonHide==(HWND)lParam)ShowWindow(hwndMain,SW_HIDE);
      if (hwndButtonEditFilter==(HWND)lParam)ShellExecute(NULL,"open",
                     szFilterStringsFile,NULL,NULL,SW_SHOWMAXIMIZED);
      break;
```

Рассмотрим функцию my_EnumWindowsProc

Пропустим все невидимые окна

```
if (!IsWindowVisible(hwnd)) return TRUE;
```

Получим TITLE очередного окна

```
GetWindowText(hwnd, szWindowsTitle, SSZZ)
```

Затем перебираем все стоки из файла настроек

```
  for(i=0;i<MAXFIL;i++)
    if (FilterStrings[i]) // если это начало строки, то
    {
     if (Contain(szWindowsTitle, FilterStrings+i)) HideNext(hwnd);
     // скроем окно, если эта строка содержится в szWindowsTitle
     while(FilterStrings[i]) i++;
     // сместим указатель на следующий 0
    }
```

Продолжаем дальнейший перебор окон

```
return TRUE;
```

(Если бы было return FALSE, перебор бы закончился.)

В остальных функциях особо описывать нечего.

codenet.ru›Языки **программирования**›Microsoft Visual C›esmall.php

Разбор оконной процедуры(Win-API)

Рассмотрим второй обязательный кусок нашей мини-программы. Это так называемая оконная процедура (хотя формально – это сишная функция !), названная у нас WndProc. Это название произвольное – в принципе мы могли назвать ее как угодно. Основное предназначение оконной процедуры – это обработка сообщений Windows. Например, вы нажимаете левую кнопку мыши и ваша программа получает сообщение Windows WM_LBUTTONDOWN. Вы изменяете размер окна и ваша программа получает сообщение WM_SIZE. Так вот, в оконной процедуре мы и обрабатываем сообщения Windows. Для этого мы пишем большой (как правило) switch, внутри которого пишем несколько case'ов. Если мы хотим, например, чтобы наша программа обращала внимание на изменение размера окна, то мы должны написать

```
switch (Message)
{
...
case WM_SIZE:
     //Обработчик сообщения WM_SIZE.
     ...
```

Если же мы такого не напишем, то наша программа никак реагировать на изменение размеров окна не будет. При этом соответствующее сообщение Windows программа все равно получит.

Мы писали минимальную программу, так что у нас обработчик только одного сообщения Windows – WM_DESTROY. Это сообщение окно получает при своем уничтожении. В

этом обработчике мы только делаем некоторые действия, связанные с уничтоженим нашего окна.

Обратите внимание на ветку default:

```
...
    default:
        return DefWindowProc(hwnd, Message, wparam, lparam);
        ...
```

Тут мы вызываем API-функцию DefWindowProc. Основное предназначение этой API-функции – это обработка сообщений Windows, которые не обрабатываются в нашей программе (т. е. тех, для которых нет своего case). При этом ничего особенно это функция не делает, но очередь из сообщений при этом двигается. И это самое важное – если у нас не было обработчика по умолчанию, то сообщения, не обрабатываемые нашей программой, забили бы очередь, и она бы встала – и даже те сообщения, для которых есть обработчики, никогда бы не были обработаны. Таким образом, основное предназначение функции DefWindowProc – это обработка (»проглатываение») тех сообщений, которые не обрабатываются в каком-нибудь case'е.

В оконную процедуру передаются 4 параметра:

```
LONG WINAPI WndProc(HWND hwnd, UINT Message, WPARAM wparam, LPARAM lparam)
{
    ...
```

Первый из этих параметров – это окно, в которое мы передаем сообщение Windows. Второй – это само сообщение. Третий и четвертый параметры – это дополнительные параметры для конкретного сообщения. Эти параметры будет разными для разных сообщений (и для некотрых сообщений Windows могут вообще не использоваться). Например, для сообщения WM_LBUTTONDOWN в дополнительных параметрах передаваются x и y той точки, в которой мы щелкнули, информация и том, были ли при этом нажати кнопки shift, atl и ctrl и другое, для сообщения WM_SIZE – новые размеры окна и т. д.. Еще обратите внимание, что эти же параметры у нас передаются в функцию DefWindowProc. Конкретно, что означают эти параметры для некоторого сообщения Windows, надо смотреть в справке по этому сообщению.

melinuxru.wordpress.com

https://melinuxru.wordpress.com/tag/win-api/

другие функции Win-API с небольшими примерами.