Zasoby systemu Windows są danymi przechowywanymi razem z programem w pliku .exe. Gdy Windows uruchamia program zasoby pozostają na dysku, a do pamięci są ładowane tylko wtedy, gdy Windows ich potrzebuje.
Do zasobów zaliczamy: ikony, kursory, mapy bitowe, ciągi znaków, zasoby definiowane przez użytkownika, menu, klawisze skrótu, okna dialogowe.
W fazie pisania programu zasoby definiujemy w „skrypcie zasobów". Jest to plik ASCII z rozszerzeniem .RC.

## Ikony

Odwołanie do pliku z ikoną w skrypcie zasobów wygląda następująco:

```
Moja_ikona ICON nazwa_pliku_ikony.ico
```

- wyrażenie to przypisuje ikonie nazwę „Moja_ikona".

Edytor obrazów wchodzący w skład Developer Studio pozwala na tworzenie plików .ico z większą ilością rysunków ikon. Podstawowe to:
ikona standardowa: 32x32 piksele w 16 kolorach
ikona monochromatyczna: 32x32 – czarny i biały kolor.
ikona mała: 16x16 w 16 kolorach
Można zdefiniować więcej ikon w jednym pliku. Windows użyje ikonę najodpowiedniejszą do wyświetlenia w danym momencie.
Rysując ikonę w Edytorze obrazów oprócz zwykłych kolorów możemy korzystać z kolorów specjalnych – przeźroczystego i przeciwnego.

Funkcja *GetSystemMetrics* pozwala programowi uzyskać wymiary ikon oraz kursorów (podane w pikselach).

```
int GetSystemMetrics(int nIndex)
```

– funkcja zwraca informacje na temat wymiarów różnych elementów Windows lub ustawień konfiguracyjnych systemu.
`nIndex` – wskazuje jakie wartości chcemy pobrać.

W przypadku ikon używamy:

- `SM_CXICON` – szerokość standardowej ikony
- `SM_CYICON` – wysokość standardowej ikony
- `SM_CXSMICON` - szerokość małej ikony
- `SM_CYSMICON` – wysokość małej ikony

W programie aby użyć ikonę potrzebujemy jej uchwytu. Uzyskujemy go wywołując funkcję *LoadIcon*.

```
HICON LoadIcon(HINSTANCE hInstance, LPCTSTR lpIconName)
```

– funkcja ładuje ikonę z pliku .exe określonym przy pomocy uchwytu wystąpienia programu. Jeżeli ikona była już wcześniej załadowana funkcja jedynie zwraca uchwyt ikony.
`lpIconName` – nazwa ikony zdefiniowana w skrypcie zasobów.

Zamiast nazwy ikony można użyć 16 bitowych liczb całkowitych nazywanych identyfikatorami ID ikony. Stosowanie tych identyfikatorów ID zmniejsza rozmiar pliku .exe i w niewielkim stopniu przyspiesza działanie funkcji `LoadIcon`.
Jeżeli deklaracja w skrypcie zasobów wygląda następująco:

```
125 ICON pl_ikony.ico
```

to możemy użyć funkcji:

```
LoadIcon(hInstance,MAKEINTRESOURCE(125))
```

lub

```
LoadIcon(hInstance,"#125")
```

W programie można użyć także predefiniowanych ikon systemowych, wtedy parametr hInstance=NULL, a lpIconName przyjmuje jedną z wartości:
IDI_APPLICATION – domyślna ikona aplikacji
IDI_ASTERISK = IDI_INFORMATION – ikona informacji ( 'i' w chmurce)
IDI_ERROR – IDI_HAND- ikona błędu, wstrzymania
IDI_EXCLAMATION = IDI_WARNING – wykrzyknik
IDI_QUESTION - pytajnik
IDI_WINLOGO – logo Windows

Najczęściej ikona pojawia się przy definiowaniu klasy okna. Uchwyt ikony pobierają pola hIcon i hIconSm. Funkcja SetClassLong umożliwia dokonanie zmiany ikony w trakcie działania programu.
DWORD **SetClassLong**(HWND hWnd, int nIndex, LONG dwNewLong)
- funkcja umożliwia zmianę niektórych wartości struktury WNDCLASSEX.
hWnd – uchwyt okna
nIndex – określa wartość do zamiany.
Aby zamienić ikony należy nadać jej wartość : GCL_HICON, GCL_HICONSM
dwNewLong – określa nową wartość.

W trakcie działania programu możesz pobrać uchwyt do ikony przy pomocy LoadIcon lub jeżeli jest to ikona klasy okna przy pomocy funkcji GetClassLong.
DWORD **GetClassLong**(HWND hWnd, int nIndex)
– funkcja zwraca określone wartości struktury WNDCLASSEX.

Przy pomocy funkcji *DrawIcon*, możesz namalować ikonę w określonym kontekście urządzenia (np. w obszarze roboczym okna)
BOOL **DrawIcon**(HDC hdc, int X, int Y, HICON hIcon)
hdc – uchwyt kontekstu urządzenia
X – określa górny lewy róg ikony na osi x
Y - określa górny lewy róg ikony na osi y
hIcon – uchwyt ikony, która ma być namalowana

## Kursory

Obsługa kursorów w programie windowsowym jest bardzo zbliżona do obsługi ikon.
Skrypt zasobów:
```
moj_kursor CURSOR pl_kurs.cur
```

W Edytorze obrazów Developer Studio możemy utworzyć plik kursora i go edytować. W przypadku kursora dodatkowo określamy tzw. HOT SPOT – jest to jeden piksel, który jednoznacznie określa położenie na ekranie całego wskaźnika myszy.

Możemy pobrać wymiary standardowego kursora, za pomocą funkcji ***GetSystemMetrics*** podając jako wartość nIndex: SM_CXCURSOR (szerokość kursora myszy) lub SM_CYCURSOR (wyskość)

`HCURSOR` **`LoadCursor`**`(HINSTANCE hInstance, LPCTSTR lpCursorName)`
Podobnie jak w LoadIcon można korzystać z identyfikatorów ID.
Możemy także korzystać z predefiniowanych kursorów standardowych, jeżeli `hInstance = NULL`, a `lpCursorName` będzie miał jedną z wartości:
`IDC_APPSTARTING` (standardowa strzałka i mała klepsydra),
`IDC_ARROW`,
`IDC_CROSS`,
`IDC_HELP`,
`IDC_NO`,
`IDC_IBEAM` (pisanie – np. pionowa kreska),
`IDC_SIZEALL`,
`IDC_SIZENESW`,
`IDC_SIZENS`,
`IDC_SIZENWSE`,
`IDC_SIZEWE`,
`IDC_UPARROW`,
`IDC_WAIT`.

Można pobrać i zmienić kursor w trakcie działania aplikacji za pomocą funkcji `GetClassLong` i `SetClassLong` z parametrem `nIndex` równym `GCL_HCURSOR`.

Zmieniać kursor możemy także przy pomocy funkcji:
`HCURSOR` **`SetCursor`**`(HCURSOR hCursor);`
Jeżeli hCursor jest równe NULL kursor znika z okna.
Funkcja zwraca uchwyt poprzedniego kursora lub `NULL` jeżeli nie było go wcześniej.
Jeżeli kursor okna nie jest określony na `NULL` to Windows przywraca po każdym ruchu myszy kursor klasowy.

`HCURSOR` **`GetCursor`**`(VOID)` - funkcja zwraca uchwyt aktualnego kursora.

`int` **`ShowCursor`**`(BOOL bShow)` – chowa lub wyświetla kursor

## Mapy bitowe

Mapy bitowe są przechowywane w plikach z rozszerzeniem .BMP. Są to rysunki definiowane w pikselach.
Map bitowych używamy w dwóch podstawowych dziedzinach:

- Do kreślenia rysunków na ekranie ( to omówimy na późniejszych zajęciach)
- W Pędzlach – pędzel jest utworzony ze wzorca pikseli, którego Windows używa do wypełniania poszczególnych fragmentów okna.

W Skrypcie zasobów piszemy:
`Moja_mapaB BITMAP rys.bmp`

`HBITMAP` **`LoadBitmap`**`(HINSTANCE hInstance, LPCTSTR lpBitmapName)` – funkcja działa podobnie jak *LoadIcon* i *LoadCursor*.

Uchwyt do mapy bitowej służy do utworzenia pędzla.

HBRUSH CreatePatternBrush(HBITMAP hbmp) – tworzy pędzel na podstawie określonej mapy bitowej.

Jeżeli chcemy pędzla do pokolorowania tła obszaru roboczego, najprostsza droga wiedzie przez definicję pola klasy okna – `hbrBackground`.

Istnieje zasadnicza różnica pomiędzy mapami bitowymi, a innymi zasobami. Mapy bitowe są obiektami GDI. Sztuka programowania wymaga aby usuwać wszystkie obiekty GDI, gdy są już niepotrzebne:

```
DeleteObject((HGDIOBJ) hBrush);
DeleteObject((HGDIOBJ) hbmp);
```

## Zasoby tekstowe

Głównym przeznaczeniem zasobów tekstowych jest ułatwienie dopasowania programu do różnych wersji językowych. Zmieniając na inną wersję nie trzeba szukać tekstów po kodzie programu, ale tłumaczy się jedynie skrypt zasobów.
Definicja zasobów tekstowych w skrypcie zasobów ma następującą postać:

```
STRINGTABLE
{
id1, „pierwszy ciąg znaków”
id2, „drugi ciąg znaków”
[pozostałe ciągi znaków]
}
```

`id1,id2,..` muszą być albo liczbami całkowitymi, albo stałymi zdefiniowanymi przy pomocy `#define`. i zarówno program jaki i skrypt zasobów muszą znać te stałe `(#include)`.
Skrypt zasobów może zawierać tylko jedną tablicę z ciągami znaków, które nie mogą być dłuższe od 255 znaków. W ciągu nie mogą pojawić się znaki kontrolne używane w C, za wyjątkiem znaku tabulacji (\t).
Funkcje `DrawText` oraz `MessageBox` rozpoznają i właściwie interpretują znaki kontrolne zapisane w kodzie ósemkowym:

| | |
|---|---|
| Tabulacja | `\011` |
| Nowy wiersz | `\012` |
| Powrót karetki | `\015` |

Kopiujemy ciąg znaków z zasobów do bufora za pomocą funkcji:

```
int LoadString(HINSTANCE hInstance, UINT uId, LPTSTR lpBuffer,
               int nBufferMax);
```

- `hInstance` – uchwyt wystąpienia
- `uId` – identyfikator liczbowy (liczba) ID, który poprzedza każdy ciąg znaków w skrypcie zasobów.
- `lpBuffer` – wskaźnik na tablicę znaków, w której umieszczamy ciąg
- `nBufferMax` – wielkość tablicy `lpBuffer`

Funkcja zwraca ilość skopiowanych znaków do bufora, nie uwzględniając 0 kończącego ciąg.

Czasami trzeba umieścić w ciągu znaków nazwę pliku lub jakąś wartość liczbową. W tym celu w ciągu definiowanym w skrypcie zasobów należy użyć odpowiedniego znaku formatującego (np. %d) a następnie sformatować tekst używając funkcji *sprintf* lub *wsprintf*.

Zdefiniowane ciągi znaków wyświetlimy przy pomocy funkcji MessageBox.

```
int MessageBox(HWND hWnd, LPCTSTR lpText, LPCTSTR lpCaption,
               UINT uType)
```

– funkcja wyświetla okno komunikatów.

HWND hWnd - uchwyt okna właściciela lub NULL

LPCTSTR lpText - wskaźnik na tekst komunikatu

LPCTSTR lpCaption - wskaźnik na tekst nagłówka okna,

UINT uType - flagi stylu okna komunikatu np. MB_OK. *( dokładniej omówimy ten parametr przy oknach dialogowych)*

funkcja zwraca jedną z wartości odpowiadającą naciśnietemu przyciskowi:
IDABORT, IDCANCEL, IDIGNORE ,IDNO, IDOK, IDRETRY, IDYES, IDHELP.

## Zasoby zdefiniowane przez użytkownika

Dzięki zasobom definiowanym przez użytkownika można włączać różnego rodzaju dane (tekstowe i binarne) do pliku .exe. Często jest to wygodniejsze niż korzystanie z osobnych plików.
W skrypcie zasobów zapisujemy:

```
Tekstpomocy MOJ_ZASOB pomoc.hlp
```

Dwie pierwsze nazwy mogą być dowolne.

Podczas inicjacji programu ( np. w WM_CRATE) można uzyskać uchwyt zdefiniowanego przez użytkownika zasobu za pomocą funkcji:
HGLOBAL **LoadResource**(HMODULE hModule, HRSRC hResInfo)
– funkcja zwraca uchwyt do danych związanych z zasobem. Funkcja nie ładuje zasobu do pamięci.
hModule – uchwyt do modułu związanego z plikiem .exe, który zawiera dane. Podajemy tutaj uchwyt wystąpienia.
hResInfo – uchwyt do zasobu który ma być załadowany. Uchwyt ten zwraca funkcja *FindResource*
HRSRC **FindResource**(HMODULE hModule, LPCTSTR lpName, LPCTSTR lpType)
lpName – nazwa zasobu zdefiniowana w skrypcie zasobów
lpType – nazwa typu zasobów zdefiniowana w skrypcie zasobów.

Wywołanie może mieć więc postać:

```
static HGLOBAL hResource;
hResource = LoadResource(hInstance,
                FindResource(hInstance, Tekstpomocy”,
                             “MOJ_ZASOB”)
                         );
```

Gdy potrzebujemy zasób wywołujemy funkcję:
`LPVOID LockResource(HGLOBAL hResData)`
– funkcja ładuje zasób do pamięci (o ile nie został załadowany wcześniej) i zwraca wskaźnik do tego zasobu.
Gdy już nie potrzebujemy zasobu możemy zwolnić zarezerwowaną przez niego pamięć funkcją:
`FreeResource(HGLOBAL hResData).`
Zasoby są zwalniane automatycznie w chwili zakończenia działania programu.

## Menu

Włączenie menu do programu polega na zdefiniowaniu w skrypcie zasobów struktury menu, następnie wstawieniu nazwy menu do struktury klasy okna. Gdy użytkownik wybierze pozycję z menu, Windows wysyła do programu komunikat `WM_COMMAND`, przekazując identyfikator wybranej pozycji.

Pasek menu, znajdujący się bezpośrednio poniżej paska tytułu, nazywamy „głównym menu”. Pod pozycjami wymienionymi w menu głównym zazwyczaj kryją się „menu rozwijane”, które możemy dalej zagnieżdżać.

W skrypcie zasobów definiujemy menu w następujący sposób:

```
MojeMenu MENU
{
MENUITEM “Help”,              IDM_HELP, HELP
POPUP "&File"
      {
      MENUITEM "&New",        IDM_NEW
      MENUITEM SEPARATOR
      MENUITEM "&Print",      IDM_PRINT, GRAYED
      }
POPUP "&Edit", INACTIVE
      {
      MENUITEM "&Undo\tCtrl+Z",    IDM_UNDO, CHECKED
      }
}
```

MojeMenu – nazwa menu.

Definicja pozycji menu składa się z czterech elementów.

- Pierwszy – definiuje rodzaj pozycji menu.
    - MENUITEM – definiuje element menu
    - MENUITEM SEPARATOR – kreśli linę poziomą w menu rozwijanym
    - POPUP – definiuje podmenu rozwijane

- Drugi – jest widoczny na ekranie.

Tekst każdego polecenia menu musi być ujęty w cudzysłów;
znak & powoduje, że występująca po nim litera pojawi się na ekranie z podkreśleniem – to tej litery szuka Windows, gdy wybieramy pozycję menu za pomocą klawisza `[Alt]`.
Znak `'\t'` spowoduje umieszczenie tekstu po nim w nowej kolumnie.

Znak `'\a'` powoduje dosunięcie do prawej krawędzi menu części tekstu występującej po tym znaku.
Znak '\a' i '\t' używamy tylko w przypadku menu rozwijanych.

- Trzeci :

Dla MENUITEM jest to liczbowy identyfikator ID, który Windows przekazuje do programu razem z komunikatem `WM_COMMAND`
Dla `POPUP` menu rozwijane, które pojawia się na ekranie po wybraniu pozycji menu.

- Czwarty: (wyświetlany w linii wywołania rodzaju pozycji menu) opcje danej pozycji menu. Są to:
    - `CHECKED` – z lewej strony tekstu wystąpi znacznik ( tylko dla menu rozwijanego)
    - `GRAYED` – polecenie nieaktywne i nie generuje komunikatu `WM_COMMAND`. Kolor tekstu jest szary. Nie używany razem z `INACTIVE`.
    - `INACTIVE` - polecenie nieaktywne i nie generuje komunikatu `WM_COMMAND`. Tekst jest wyświetlany normalnie. Nie może być używany razem z `GRAYED`
    - `MANUBREAK` - następne polecenia ( z tym włącznie) pojawiają się w nowym wierszu paska menu dla menu głównego i w nowej kolumnie menu dla menu rozwijanego.
    - `MENUBARBREAK` - następne polecenia ( z tym włącznie) pojawiają się w nowej kolumnie oddzielonej linią pionową. Nie można użyć razem z `MENUBREAK`. (tylko dla menu rozwijanego)
    - `HELP` –następne polecenia ( z tym włącznie) będą wyrównywane do prawego marginesu. ( tylko dla menu głównego)

Można określić jakie menu chcemy wyświetlić w oknie na kilka sposobów:

- przypisując nazwę menu polu `lpszMenuName` struktury `WNDCLASSEX`.
- ładując menu do pamięci za pomocą funkcji `LoadMenu` i przekazując uchwyt menu jako dziewiąty parametr funkcji `CreateWindow`. To menu zastąpi menu określone w klasie okna ( jeżeli jakieś zostało dla niej zdefiniowane)

  `HMENU` **`LoadMenu`**`(HINSTANCE hInstance, LPCTSTR lpMenuName)`
  - funkcja działa podobnie jak `LoadIcon, LoadCursor`.

- przypisując menu do okna, kiedy okno już istnieje za pomocą funkcji SetMenu.

`BOOL` **`SetMenu`**`(HWND hWnd, HMENU hMenu)`
– funkcja zastępuje poprzednie menu, ale go nie niszczy. Aby usunąć menu nie przypisane do żadnego okna należy wywołać funkcję *`DestroyMenu`*. Menu przypisane do okna jest automatycznie niszczone kiedy aplikacja kończy działanie.
`BOOL` **`DestroyMenu`**`(HMENU hMenu)`

**WM_INITMENUPOPUP** – komunikat wysyłany do procedury okna, kiedy Windows jest gotowy do wyświetlenia menu rozwijanego. Wykorzystujemy ten komunikat jeżeli należy udostępnić lub zabronić dostępu do poszczególnych poleceń menu, zanim menu zostanie rozwinięte na ekranie.
`wParm` – uchwyt menu rozwijanego
`LOWORD(lParam)` –indeks menu rozwijanego
`HIWORD(lParam)` –1 dla menu sterowania (menu rozwijane po naciśnięciu na przycisk z lewej strony paska tytułu), 0 dla pozostałych menu.

**WM_COMMAND** – komunikat przesyłany gdy użytkownik wybrał polecenie menu.
`LOWORD (wParam)` - identyfikator ID polecenia menu

W trakcie działania programu możemy zmieniać opcje poleceń menu.
Aby zaznaczyć polecenie menu musimy pobrać uchwyt menu:
```
HMENU GetMenu (HWND hWnd);
```
Który wykorzystujemy w funkcji:
```
DWORD CheckMenuItem(HMENU hmenu, UINT uIDCheckItem,
                    UINT uCheck)
```
`hmenu` – uchwyt menu
`uIDCheckItem` – identyfikator ID polecenia menu
`uCheck`:
`MF_CHECKED` – zaznacza polecenie menu,
`MF_UNCHECKED` – odznacza polecenie menu
Funkcja zwraca informację o poprzednim stanie polecenia:
`MF_CHECKED, MF_UNCHECKED.`

Możemy także uaktywniać i dezaktywować polecenia menu za pomocą funkcji:
```
BOOL EnableMenuItem(HMENU hMenu, UINT uIDEnableItem,
                    UINT uEnable)
uEnable –MF_DISABLED, MF_GRAYED, MF_ENABLED
```

**Menu kontekstowe:**
Możemy stosować menu bez paska menu głównego, tzw. menu kontekstowe, które pojawia się po naciśnięciu prawego klawisza myszy na obszarze roboczym.
Menu kontekstowe definiujemy w skrypcie zasobów w następujący sposób:
```
MojeMenu MENU
{
  POPUP ""
  {
     [polecenia menu]
  }
}
```

Następnie pobieramy uchwyt do menu rozwijanego.
`hMenu=LoadMenu(hInst, „MojeMenu”);`
`hMenu =GetSubMenu(hMenu,0);` - funkcja zwraca uchwyt do podmenu, gdzie `iPosition` jest indeksem (licząc od 0) menu rozwijanego w menu głównym wskazanym prze `hMenu`.

Po otrzymaniu komunikatu WM_RBUTTONDOWN wywołujemy funkcję, która wyświetla menu kontekstowe i śledzi zaznaczanie pozycji na tym menu.

```
BOOL TrackPopupMenu ( HMENU hMenu, UINT uFlags, int x, int y,
                      int nReserved, HWND hWnd,
                      CONST RECT *prcRect)
```

hMenu – uchwyt menu kontekstowego

uFlags – opcje wyświetlania menu, oraz czy ma być wysłany komunikat WM_COMMAND. Aby użyć domyślnych ustawień przypisujemy temu parametrowi 0.

x – określa położenie menu na osi x względem lewego górnego rogu ekranu (**ClientToScreen**(hwnd, &point))

y- określa położenie menu na osi y względem lewego górnego rogu ekranu (**ClientToScreen**(hwnd, &point))

nReserved – zarezerwowane; musi być 0

hWnd – okno które otrzymuje komunikaty od menu kontekstowego

prcRect – ignorowane.

Możemy dokonać zmian menu w trakcie działania programu:

BOOL **AppendMenu**(HMENU hMenu, UINT uFlags, UINT_PTR uIDNewItem, LPCTSTR lpNewItem)

– dopisuje nową pozycję na końcu menu.

hMenu – uchwyt menu które ma zostać zmienione

nFlags – specyfikuje opcje które określają wygląd i zachowanie polecenia menu:

- MF_BITMAP — Używa bitmapy jako polecenia menu. LpNewItem zawiera uchwyt
- MF_CHECKED — Zaznacza polecenie.
- MF_DISABLED — Dezaktywuje polecenie, ale wyświetla go normalnie
- MF_ENABLED — Uaktywnia polecenie
- MF_GRAYED — Dezaktywuje polecenie i wyświetla go na szaro.
- MF_MENUBARBREAK — Polecenie menu rozwijanego pojawia się w nowej kolumnie oddzielonej linią pionową
- MF_MENUBREAK — Polecenie pojawiają się w nowym wierszu paska menu dla menu głównego lub w nowej kolumnie menu dla menu rozwijanego
- MF_POPUP — Polecenie otwiera menu rozwijane. *uIDNewItem* określa uchwyt menu rozwijanego.
- MF_SEPARATOR — Kreśli linę poziomą w menu rozwijanym. *lpNewItem* i *uIDNewItem* są ignorowane.
- MF_STRING — Wskazuje że polecenie jest ciągiem znaków. *lpNewItem* jest wskaźnikiem na tablicę znakową.
- MF_UNCHECKED — Nie zaznacza polecenia

uIDNewItem – identyfikator ID polecenia menu lub uchwyt menu rozwijanego

lpNewItem – określa zawartość nowej pozycji menu, jej rodzaj zależy od *uFlags*

BOOL **DeleteMenu**(HMENU hMenu, UINT uPosition, UINT uFlags)
- usuwa pozycję z menu i usuwa menu rozwijane z systemu
uFlags
MF_BYCOMMAND – określa, że uPosition specyfikuje identyfikator ID pozycji menu,
MF_BYPOSITION – określa że uPosition identyfikuje indeks pozycji menu.

BOOL **RemoveMenu**(HMENU hMenu, UINT uPosition, UINT uFlags)
– usuwa pozycję menu, ale nie usuwa menu rozwijane z systemu.

BOOL **InsertMenu**(HMENU hMenu, UINT uPosition, UINT uFlags,
UINT_PTR uIDNewItem, LPCTSTR lpNewItem)
– wstawia nową pozycję do menu
uPosition – określa pozycje menu przed który nowa pozycja ma być dodana.
Pozostałe parametry jak w *AppendMenu* i *DeleteMenu*

BOOL **ModifyMenu(**HMENU hMenu, UINT uPosition, UINT uFlags,
UINT_PTR uIDNewItem, LPCTSTR lpNewItem)
- zmienia istniejącą pozycję menu
uPosition – określa pozycję menu do modyfikacji.
Pozostałe parametry jak w *InsertMenu*

**Przydatne polecenia menu:**

BOOL **DrawMenuBar**(HWND hWnd)
– funkcja wymusza na Windows natychmiastowe odświeżenie poleceń w menu głównym.

int **GetMenuItemCount**(HMENU hMenu)
– funkcja zwraca liczbę poleceń menu

UINT **GetMenuItemID**(HMENU hMenu, UINT iPosition)
– funkcja zwraca identyfikator ID polecenia menu rozwijanego, określonego przez indeks (licząc od 0) menu.

UINT **GetMenuState**(HMENU hMenu, UINT uId, UINT uFlags)
– zwraca jakie flagi są ustawione dla danej pozycji (*uId*) menu(*hMenu*).
uFlags – określa, czy uId to identyfikator ID pozycji czy indeks
Zwraca kombinację flag ( zobacz *AppendMenu*)

## Klawisze skrótu

Klawisze skrótu (keyboard accelerators) są takimi kombinacjami naciśniętych klawiszy, które generują komunikat `WM_COMMAND`.

Tablicę klawiszy skrótu definiujemy w skrypcie zasobów:

```
MojeKlawisze ACCELERATORS
{
"^Z",      IDM_UNDO                      \\[Ctrl+Z]
"^C",      IDM_COPY                      \\[Ctrl+C]
VK_INSERT,IDM_PASTE, VIRTKEY, SHIFT      \\[Shift+Insert]
VK_F1,     IDM_HELP, VIRTKEY             \\[F1]
"Z"        IDM_Z                         \\[Z]
}
```

W jednym skrypcie można zamieścić kilka tablic.

- Pierwszy parametr to znak, który ma być naciśnięty.

„^" +znak jest równoznaczny z zadeklarowaniem wciśnięcia znaku razem z [Ctrl]. Można także deklarować wirtualne kody klawisza wtedy trzeci parametr musi zawierać słowo kluczowe VIRTKEY.

- Drugi parametr to identyfikator ID klawisza skrótu. Jeżeli chcemy by klawisz skrótu zastępował wybranie polecenia z menu, ich identyfikatory muszą być identyczne.
- W trzecim parametrze po przecinku można umieścić : SHIFT, CONTROL, ALT – definiujące klawisze, które mają być naciśnięte razem ze znakiem, oraz słowa kluczowe VIRTKEY lub ASCII.

Tablicę wprowadzamy do programu za pomocą funkcji:

```
HACCEL LoadAccelerators(HINSTANCE hInstance,
                        LPCTSTR lpTableName)
```

– ( wywołanie jak *LoadIcon)* – zwraca uchwyt tablicy klawiszy skrótu

Aby obsłużyć komunikaty klawiszy skrótu musimy zmodyfikować pętlę komunikatów w `WinMain`:

```
while (GetMessage(&msg, NULL, 0, 0))
{
   if(!TranslateAccelerator(hwnd,hAccel,&msg))
   {
     TranslateMessage(&msg);
     DispatchMessage(&msg);
    }
}
```

`int` **`TranslateAccelerator`**`(HWND hwnd ,HACCEL hAccel, LPMSG msg)`
- Funkcja sprawdza czy komunikat zawarty w strukturze msg, jest komunikatem klawiatury. Jeśli tak funkcja przegląda tablicę skrótu, której uchwyt równa się *hAccel.* Jeżeli stwierdzi, że komunikat pochodzi od klawisza skrótu, wywołuje procedurę okna o uchwycie *hwnd*. Wysyła komunikat WM_COMMAND lub WM_SYSCOMMAND. Jeżeli komunikat nie odpowiada jednemu z klawiszy skrótu funkcja zwraca 0, w przeciwnym razie zwraca wartość różną od zera.

Parametry **WM_COMMAND**, w zależności kto wysyła ten komunikat:

| | ***Klawisz skrótu*** | ***Menu*** | ***Kontrolka*** |
|---|---|---|---|
| LOWORD (*wParam*) | ID klawisza skrótu | ID menu | ID kontrolki |
| HIWORD (*wParam*) | 1 | 0 | Kod powiadomienia |
| *lParam* | 0 | 0 | Uchwyt okna potomnego |