

Laboratorium Grafiki Komputerowej i Animacji

Ćwiczenie I

Wprowadzenie do programowania w Windows

Sławomir Samolej

Rzeszów, 2012

1. Wstęp

Od kilku lat dominującym systemem operacyjnym na rynku komputerów osobistych jest Microsoft Windows. Wersje systemu Windows 95/98 i Windows NT wykorzystują w pełni 32 bitową architekturę systemów mikroprocesorowych współczesnych komputerów osobistych. Podstawowymi zaletami wyżej wymienionych systemów operacyjnych są jednorodny, niezależny od platformy sprzętowej interfejs użytkownika, wielozadaniowość i wielowątkowość, zintegrowana obsługa wydruku, wbudowany mechanizm przenoszenia obiektów pomiędzy aplikacjami. System Windows stał się również platformą wykorzystywaną do tworzenia oprogramowania stacji graficznych. Dominującymi bibliotekami graficznymi stanowiącymi obecnie integralną część sprzedawanych systemów Windows są DirectX firmy Microsoft oraz OpenGL firmy Silicon Graphics.

Opracowanie w prezentuje podstawy tworzenia oprogramowania zgodnego ze standardem Win32 (32 bitowa wersja Windows). Kolejne rozdziały przedstawiają etapy tworzenia aplikacji Windows w języku C. Omówione zostaną również podstawy obsługi systemu do tworzenia aplikacji w Windows - Microsoft Developer Studio - Visual C++ 5.0. Celem opracowania jest wyjaśnienie sposobu tworzenia aplikacji Windows, oraz idei działania programu w środowisku Windows. Do opracowania dołączona jest dyskietka zawierająca projekt omawianej aplikacji.

2. Podstawy tworzenia aplikacji zgodnej z Win32

Podany poniżej "przepis" na tworzenie aplikacji Windows może być stosowany do tworzenia oprogramowania w języku C przy pomocy dowolnego 32 bitowego kompilatora C/C++. Opracowanie nie będzie omawiało żadnej z gotowych bibliotek wspomagających programowanie w Windows takich jak MFC czy OWL. Szkielet programu w Windows stworzony zostanie w oparciu o podstawowy zestaw funkcji systemu. Do tworzenia oprogramowania wykorzystany będzie pakiet Microsoft Developer Studio Visual C++ 5.0.

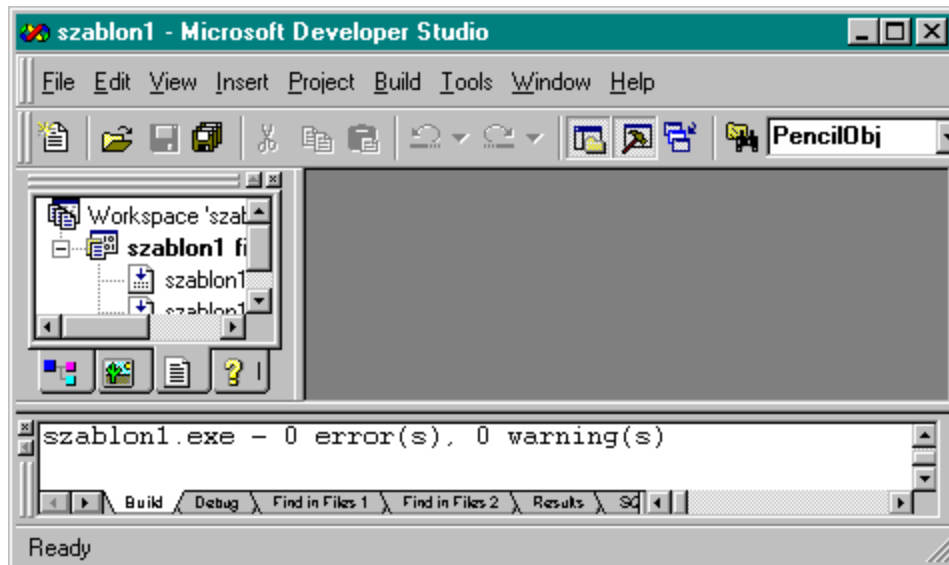
Opracowywanie programu w środowisku Windows można podzielić na następujące etapy:

1. **Utworzenie nowego folderu przeznaczonego na pliki należące do projektu wraz z plikiem projektowym zawierającym dane na temat kompilowania i konsolidowania programu.**
2. **Wykorzystanie środowiska programowania do utworzenia zasobów aplikacji (menu, ikony, okna dialogowe, paski narzędzi, bitmapy, klawisze skrótów, zasoby tekstowe) i dołączenie pliku z zasobami do pliku projektowego.**
3. **Utworzenie plików źródłowych z funkcjami sterującymi aplikacją oraz dołączenie ich do pliku projektowego.**
4. **Kompilacja, konsolidacja i uruchomienie programu.**

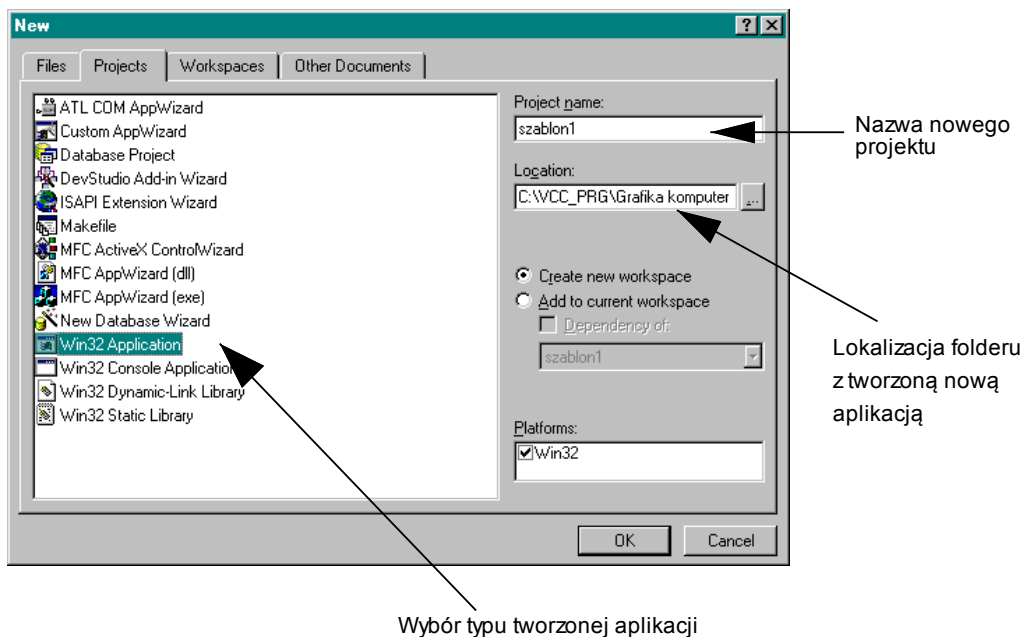
2.1 Utworzenie nowego projektu.

Podstawowym programem wchodzącym w skład pakietu Microsoft Developer Studio 5.0 służącym do tworzenia aplikacji w języku C/C++ dla środowiska Windows jest **Microsoft Visual C++ 5.0**. Po uruchomieniu tego programu na ekranie pojawia się okno aplikacji podobne do wielu popularnych systemów programowania (rys 2.1.1). Aby rozpocząć tworzenie nowego projektu programu należy z menu *File* kliknąć polecenie *New*. Otworzy się okno dialogowe *New*,

pokazane na rysunku 2.1.2. Jeśli trzeba, należy kliknąć w zakładkę *Projects* i z położonej poniżej listy należy wybrać pozycję *Win 32 Application*. W polu *Project name* należy wpisać nazwę nowego projektu. Pole *Location* pozwala zdecydować w jakim miejscu na dysku tworzony będzie nowy projekt. Kliknięcie przycisku *OK* spowoduje automatyczne utworzenie folderu o nazwie podanej w polu *Project name* zlokalizowanego w folderze określonym w polu *Location*. W folderze projektu zostaną również automatycznie utworzone pliki projektowe, z których podstawowe posiadają rozszerzenia **dsp** i **dsw**. Do tak stworzonego projektu można kolejno dołączać elementy aplikacji takie jak zasoby, czy pliki źródłowe. Prezentowany w dalszej części opracowania projekt przykładowej aplikacji w Windows nosi nazwę *Szablon1*.



Rys 2.1.1 Główne okno programu Microsoft Visual C++ 5.0

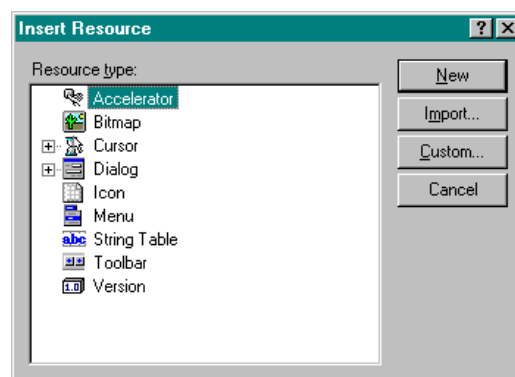


Rys 2.1.2 Okno dialogowe wspomagające tworzenie nowego projektu.

2.2. Generowanie zasobów projektu.


Większość programów przeznaczonych do wykonywania w środowisku Windows wykorzystuje standardowe elementy służące do komunikacji z użytkownikiem lub do identyfikacji programu w systemie takie jak: menu, bitmapy, wskaźniki myszy, okna dialogowe, paski narzędzi i inne. Wszystkie te elementy, jeśli występują jako składowe aplikacji, należą do tak zwanej grupy zasobów aplikacji (ang. resources). Tworzone są one osobno, a następnie włączane do pliku projektowego. Stworzenie zasobów powiązane jest ze zdefiniowaniem unikalnych w aplikacji identyfikatorów liczbowych (stałych liczbowych). Identyfikatory pozwalają na odwoływanie się do zdefiniowanych zasobów podczas pisania kodu źródłowego (np. w języku C). Nowe narzędzia projektowania aplikacji dla środowiska Windows zawierają podprogramy, które umożliwiają automatyczne tworzenie zasobów. Mają one formę graficznych edytorów okien dialogowych, menu, ikon itp.

Podstawowym zasobem większości aplikacji w środowisku Windows jest menu. Poniżej omówiony zostanie szczegółowo sposób tworzenia tego zasobu w środowisku Visual C++ 5.0. Ponieważ pozostałe zasoby nie będą wykorzystywane w ramach przedmiotu Grafika Komputerowa i Animacja zainteresowanych tworzeniem bardziej zaawansowanych zasobów i aplikacji w Windows odsyła się do spisu literatury zamieszczonego na końcu opracowania. Tworzenie nowego menu rozpoczyna się przez wybranie z menu *Insert* opcji *Resources*. Na ekranie pojawi się okno jak na rysunku rys 2.2.1

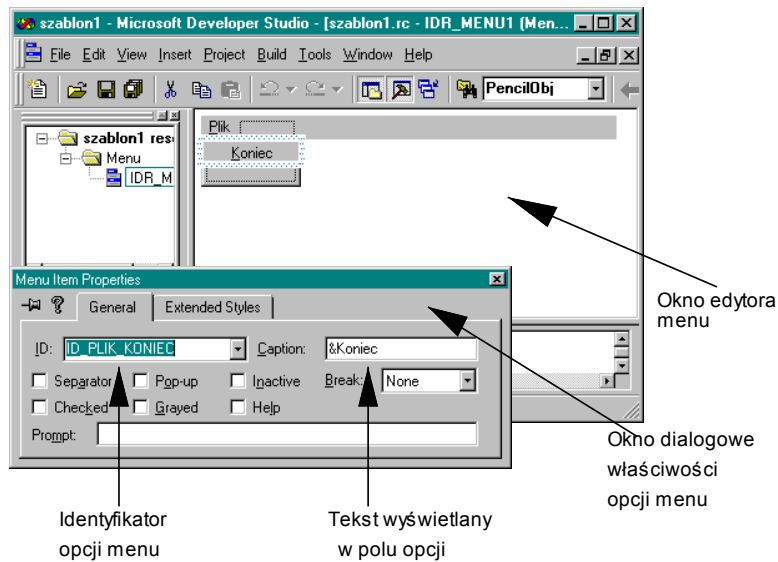


Rys 2.2.1 Okno wyboru nowego zasobu

Aby uruchomić edytor menu należy kliknąć element *Menu* z wyświetlonej w oknie listy. W głównym oknie systemu pojawia się wtedy szara belka z oznaczonymi polami przeznaczonymi do tworzenia poszczególnych opcji menu (rys 2.2.2).

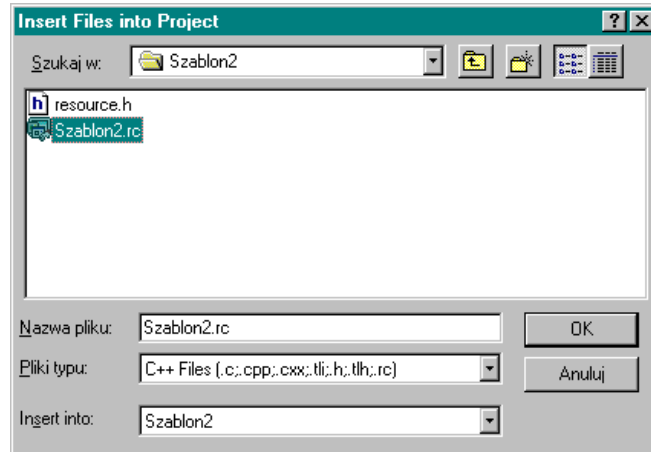
Podwójne kliknięcie na zaznaczonym polu belki powoduje pojawienie się dodatkowego okna dialogowego (rys 2.2.2), w którym można ustalić jaki napis pojawia się w opcji menu i jaki **identyfikator** (automatycznie definiowana zostaje w systemie unikalna stała liczbowo o podanej nazwie) posiadać będzie stworzona opcja. Dodatkowo istnieje możliwość określenia dodatkowych właściwości jakimi mogą się cechować definiowane pola menu. Należy pamiętać, że gdy w programie użytkownik wybierze daną opcję menu, do procedury sterującej obsługą komunikatów przekazany zostanie między innymi identyfikator tej opcji menu. Jeśli chcemy aby dana opcja menu była dostępna przy pomocy kombinacji klawiszy "Alt+litera", co uwidaczniane jest na menu przez podkreślenie danej litery w napisie, należy przed wyróżnioną literą w polu definiowania napisu umieścić znak "&". Zakończenie tworzenia menu polega na wciśnięciu przycisku zamknięcia okna edytora graficznego menu (). Środowisko Developer Studio automatycznie tworzy tak zwany plik skryptu zasobów i domyślnie nadaje mu nazwę

"Script1.rc". Przed zachowaniem pliku na dysku istnieje możliwość dokonania zmiany nazwy pliku skryptu zasobów (W omawianym programie przykładowy plik skryptu zasobów ma nazwę szablony.rc).



Rys 2.2.2 Okno edytora menu systemu Visual C++ 5.0

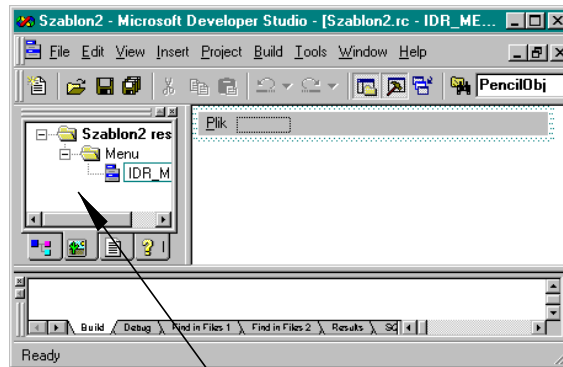
Aby dołączyć do projektu nowo zdefiniowany plik skryptu zasobów należy z menu *Project* wybrać opcję *Add to Project* i podopcję *Files*. Na ekranie pojawi się okno jak na rys. 2.2.3.



Rys 2.2.3 Okno dialogowe dołączania plików do projektu.

Należy następnie wybrać plik skryptu zasobów i kliknąć przycisk *OK*. W podobny sposób jak pliki zasobów można dołączać do projektu dowolne inne pliki. Dotyczy to także plików z kodami źródłowymi funkcji sterujących programem. Samo menu uzyskuje w momencie utworzenia pliku skryptu zasobów swój indywidualny identyfikator zwykle o nazwie `IDR_MENU1`. Równocześnie z kreowaniem pliku z zasobami tworzony i modyfikowany jest tekstowy plik nagłówkowy o nazwie **resource.h**. Zawiera on między innymi definicje stałych tekstowych utworzonych do identyfikacji poszczególnych zasobów w tekście programu. Jeśli tworzony program wykorzystuje zasoby, plik `resource.h` należy przy pomocy dyrektywy `#include` dołączyć do pliku źródłowego z tekstem programu. W dowolnej chwili pracy w systemie istnieje

możliwość modyfikowania zasobów lub dołączania zasobów do utworzonego skryptu zasobów. Najwygodniejszym sposobem przeglądania plików i zasobów włączonych do danego projektu jest posługiwanie się pomocniczym oknem o nazwie "Workspace" (rys 2.2.4). W przypadku gdy do systemu załadowany jest projekt okno pozwala na przyglądnięcie jego zawartości. Dwukrotne kliknięcie dowolnego elementu z listy podanej w oknie powoduje automatyczne uruchomienie odpowiedniego edytora (tekstowego lub graficznego) umożliwiającego modyfikację elementu wchodzącego w skład projektu.



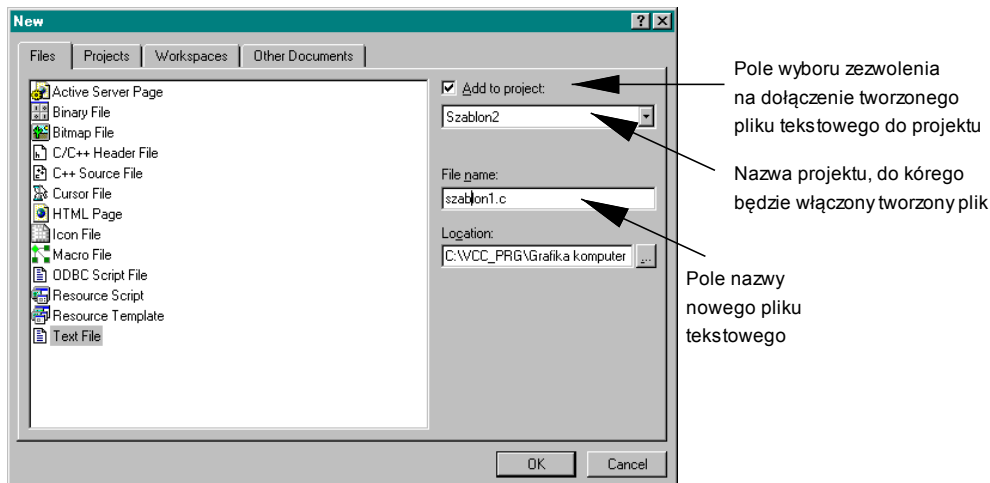
Okno "Workspace"

Rys 2.2.4 Okno "Workspace" pakietu Visual C++ 5.0

2.3 Kod źródłowy aplikacji zgodnej z Win32

Opracowanie prezentuje sposób tworzenia aplikacji w Windows bez wykorzystania narzędzi do automatycznej generacji kodu źródłowego programu. Przygotowany kod źródłowy omawiany w ramach ćwiczenia dostarczony jest na dołączonej do opracowania dyskietce. Ponieważ tworzenie aplikacji Windows, zwłaszcza jeśli posługujemy się językiem C, jest zagadnieniem kłopotliwym omówiony poniżej kod programu należy traktować jako swego rodzaju szablon, dzięki któremu można będzie budować własne aplikacje. Najwygodniej rozpoczynać tworzenie nowego projektu od włączenia do niego omówionego poniżej pliku z kodem źródłowym. W następnym kroku można dokonywać odpowiednich modyfikacji kodu.

Do tworzenia plików tekstowych z kodami źródłowymi funkcji wchodzących w skład projektowanej aplikacji można się posłużyć edytorem tekstowym systemu Microsoft Developer Studio. W tym celu można wybrać z opcji menu *File* podopcję *New*. Otworzy się okno dialogowe *New*, pokazane na rysunku 2.3.1. Jeśli trzeba należy kliknąć w zakładkę *Files* i z położonej poniżej listy można wybrać pozycję *Text File*. W polu *File name* należy wpisać nazwę nowego pliku tekstowego (np. szablon1.c). Zaznaczenie w oknie dialogowym pola wyboru *Add to project* spowoduje automatyczne włączenie pliku jako składnika aktywnego projektu. Jeśli chcemy wprowadzić do projektu napisany wcześniej plik tekstowy, dany plik powinno się przekopiować do folderu ze zdefiniowanym nowym projektem, a następnie klikając w menu *Project* opcję *Add to Project* i wybierając podopcję *Files* należy podobnie jak było to omówione w przypadku włączania pliku skryptu zasobów włączyć dany plik do projektu.



Rys 2.3.1 Okno dialogowe wspomagające tworzenie nowego pliku tekstowego.

Poniżej podany zostanie pełny kod źródłowy programu szablon1.c. W kolejnych podrozdziałach omówione zostaną poszczególne fragmenty kodu.

```

////////////////////////////////////
//                                                                    //
// PROGRAM: SZABLON1 - WZÓR APLIKACJI ZGODNEJ Z WIN32                //
//                                                                    //
//                                                                    //
////////////////////////////////////

#define STRICT // żądanie ścisłego przestrzegania
                // zgodności typów w programie
#include <windows.h> // plik nagłówkowy funkcji API
                // Windows
#include "resource.h"

LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);
                // Prototyp głównej funkcji
                // obsługi głównego okna programu

// Główna funkcja programu:
int WINAPI WinMain( HINSTANCE hInstance, // uchwyt instancji
                  //
programu
                  HINSTANCE hPrevInstance, // 0
                  PSTR szCmdLine,        // linia komend
                  int iCmdShow)           // początkowy
stan
//
programu
{
    static char szAppName[]="Win1";
    HWND hwnd; // uchwyt głównego okna
    MSG msg;    // struktura przechowująca messages

```

```

WNDCLASS wndclass;    // struktura do konstruowania klasy okna

// Tworzenie klasy głównego okna:
wndclass.style=CS_HREDRAW|CS_VREDRAW; // styl klasy okna
wndclass.lpfWndProc=WndProc;          // nazwa głównej
procedury                               // sterującej
okna
wndclass.cbClsExtra=0;                 // dodatkowy obszar
pamięci
wndclass.cbWndExtra=0;                 // dodatkowy obszar
pamięci
wndclass.hInstance=hInstance;         // instancja programu
wndclass.hIcon=LoadIcon(NULL,IDI_APPLICATION); // ikona klasy
okien
wndclass.hCursor=LoadCursor(NULL, IDC_ARROW); // kształt kursora
wndclass.hbrBackground=(HBRUSH)GetStockObject(WHITE_BRUSH);

// kolor tła
wndclass.lpszMenuName=MAKEINTRESOURCE(IDR_MENU1); // przyłączone
MENU
wndclass.lpszClassName=szAppName;     //
indywidualna nazwa

// klasy okna

RegisterClass(&wndclass); // zarejestrowanie klasy

hwnd=CreateWindow(
    szAppName, // Nazwa klasy okna
    "Szablon1", // Napis w pasku tytułu
    WS_OVERLAPPEDWINDOW, // Styl okna
    CW_USEDEFAULT, // położenie okna x
    CW_USEDEFAULT, // położenie okna y
    CW_USEDEFAULT, // wysokość okna
    CW_USEDEFAULT, // szerokość okna
    NULL, // uchwyt do okna rodzica
    NULL, // uchwyt do menu lub identy-
           // fikator okna dziecka
    hInstance, // uchwyt do instancji programu
    NULL // uchwyt do danych tworzących
okno
);
ShowWindow(hwnd, iCmdShow); // przygotuj okno do wyświetlenia
UpdateWindow(hwnd); // wyślij pierwszy komunikat WM_PAINT

////////////////////////////////////
////
// Obsługa pętli komunikatów : //
////////////////////////////////////
////
while (GetMessage (&msg, NULL, 0, 0))

```



```

    {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }

    return msg.wParam;
}
// Główna procedura obsługi komunikatów okna:
LRESULT CALLBACK WndProc(HWND hwnd,
                        UINT iMsg,
                        WPARAM wParam,
                        LPARAM lParam)
{
    HDC HDCPaint;           // uchwyt obszaru kontekstu
                          // urządzenia klienta
    PAINTSTRUCT PaintStruct; // informacja o malowaniu
    RECT rect;             // współrzędne obszaru prostokątnego

    switch(iMsg)
    {
        case WM_COMMAND: // Komunikat przychodzący po wybraniu
                        // opcji menu itp
            switch(LOWORD(wParam))
            {
                // Wybrano opcję menu Plik->Koniec:
                case ID_PLIK_KONIEC:
                    DestroyWindow(hwnd);
                    return 0;
            }
            return 0;

        case WM_DESTROY: // Komunikat przychodzący gdy okno
                        // ma być zniszczone
            PostQuitMessage(0);
            return 0;

        case WM_PAINT: // Komunikat przychodzi w przypadku
konieczności
                        // "odmalowania" lub namalowania okna
                        // Wymaż okno, uzyskaj kontekst urządzenia:
                        HDCPaint=BeginPaint(hwnd,&PaintStruct);
                        // Pobierz rozmiary obszaru roboczego okna
                        GetClientRect(hwnd,&rect);
                        // Umieść centralnie w oknie tekst:
                        DrawText(HDCPaint,"Pozdrowienia",-1,&rect,
                                DT_SINGLELINE|DT_CENTER|DT_VCENTER);
                        // Zakończ rysowanie
                        EndPaint(hwnd, &PaintStruct);
                        return 0;
    }
}

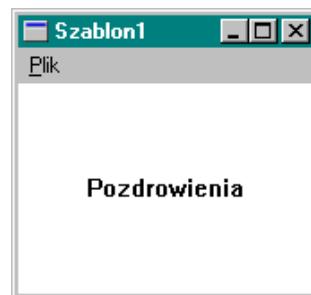
```

```

default:
    return DefWindowProc (hwnd, iMsg, wParam, lParam);
}
}

```

Program szablon1 wyświetla na ekranie okno z umieszczonym centralnie napisem "Pozdrowienia". Okno wyposażone jest w menu umożliwiające zamknięcie programu. Istnieje możliwość zmiany rozmiaru okna. Okno programu szablon1 zaprezentowane jest na rysunku 2.3.2.



Rys 2.3.2 Okno programu szablon1

Idea programowania w Windows w języku C polega na odpowiednim zonglowaniu pewnym zdefiniowanym zestawem gotowych funkcji (nagłówki ich znajdują się między innymi w plikach **windows.h** **windowsx.h** dołączanych na początku programu), które odpowiadają funkcjom systemu. Podstawową nowością w architekturze programu przeznaczanego do wykonywania w środowisku Windows jest fakt, że szereg funkcji zdefiniowanych w programie wywoływanych jest nie przez główną procedurę sterującą programem, ale bezpośrednio przez system operacyjny. Wprowadzone zostały predefiniowane stałe liczbowe identyfikujące komunikaty od systemu, a także predefiniowano nowe typy danych sugerujące rodzaj otrzymywanych danych przez poszczególne części programu. Zdefiniowanie stałej **STRICT** na początku pliku z kodem źródłowym powoduje, że kompilator ściśle przestrzega sprawdzanie typów danych, co pomaga w usuwaniu niektórych błędów i sprawia, że kod tworzonej aplikacji będzie przenośny na następne wersje systemu Windows.

2.3.1 Funkcje sterujące przebiegiem programu

Kiedy program jest uruchamiany pierwszą funkcją, która otrzymuje sterowanie od systemu operacyjnego jest `WinMain()`. Odpowiada ona znanej z C funkcji `main()`. Przekazywane są do niej cztery parametry odbierane z systemu operacyjnego:

```

int WINAPI WinMain(    HINSTANCE hInstance,
                      HINSTANCE hPrevInstance,
                      PSTR szCmdLine,
                      int iCmdShow)

```

Parametr `hInstance` zawiera uchwyt do bieżącej instancji programu. Instancja jest liczbą, która jednoznacznie identyfikuje program działający w środowisku Windows. Uchwyt instancji programu wykorzystywany jest przez niektóre funkcje wywoływane w programie. Parametr `szCmdLine` jest wskaźnikiem do tablicy tekstowej zawierającej linię zleceń programu jeśli takie

zostały w momencie jego wywołania podane (tablica nie zawiera nazwy programu, a jedynie flagi, nazwy plików lub tekst zawarty w linii zleceń programu). Pozostałe parametry wywołania funkcji WinMain() pozostają jedynie dla kompatybilności tej funkcji z wcześniejszymi wersjami Windows. Parametr hPrevInstance ustawiany jest zawsze na 0, a we wcześniejszych wersjach systemu zawierał uchwyt instancji programu, który wywołany był przed uruchomieniem naszego programu. Parametr iCmdShow może zawierać wartość określającą początkowy stan okna programu w momencie jego uruchomienia (np. można zarządać aby program uruchamiany poprzez skrót do niego był automatycznie minimalizowany). W Windows 95/98/NT nie trzeba przechwytywać tego parametru, ponieważ rolę tą przejąć może przesłanie odpowiednich flag do funkcji ShowWindow(), która omówiona zostanie w dalszej części opracowania.

Następnie funkcja WinMain() dokonuje utworzenia głównego okna programu. Dwuczściowy proces tworzenia okna omówiony zostanie później. Ostatnie linie kodu funkcji WinMain() mają postać:

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

    DispatchMessage ( &msg );
}
```

Od momentu uruchomienia wyżej pokazanej pętli *while* program reaguje tylko na tak zwane komunikaty systemowe przeznaczone dla niego (Ujawnia się tu podstawowa różnica pomiędzy typowymi programami w C a programami pisanymi dla systemu Windows. W typowych programach programista samodzielnie tworzył interfejs pomiędzy swoim programem a użytkownikiem, czy systemem operacyjnym. W systemie Windows program jest tak skonstruowany aby reagował jedynie na sygnały wysyłane do niego przez system). Windows dla każdego działającego programu windowsowego zakłada i obsługuje "kolejkę komunikatów". Jeśli kiedykolwiek następuje ważne dla programu zdarzenie (okno ma zostać narysowane na ekranie, okno ma być zniszczone, pojawił się sygnał od użytkownika dla programu) Windows umieszcza strukturę danych znaną jako komunikat (ang. message) w kolejce komunikatów programu. Umieszczenie komunikatu w kolejce określane jest jako wysłanie komunikatu do okna. Kiedy program wywołuje funkcję GetMessage() znajdującą się na początku pętli obsługi komunikatów, pierwszy komunikat jaki znajduje się w kolejce jest z niej pobierany i odpowiednie jego części rozmieszczane są w strukturze typu MSG przeznaczonej do przechowywania aktualnie obsługiwanego komunikatu. Jeśli w kolejce nie ma żadnego komunikatu funkcja GetMessage() oczekuje na pojawienie się następnego. Funkcja TranslateMessage() pełni rolę wspomagającą konwersję pewnego typu komunikatów przekazywanych do okna (np. komunikaty obsługi klawiatury WM_CHAR). Funkcja DispatchMessage() przesyła pobrany komunikat do odpowiedniej procedury okna programu. Przesłanie komunikatu do procedury oka odbywa się za pośrednictwem systemu operacyjnego. Zadaniem procedury okna jest interpretowanie otrzymywanych komunikatów i wywoływanie odpowiednich reakcji programu na nie. W programie szablon1 główna procedura sterująca oknem nosi nazwę WndProc(). Podstawowa struktura funkcji obsługi komunikatów systemowych może mieć następującą postać:

```
LRESULT CALLBACK WndProc ( WND hwnd,
                          UINT iMsg,
                          WPARAM wParam,
                          LPARAM lParam)
```

```

{
    switch (iMsg)
    {
        case WM_CREATE:
            // obsłuż komunikat WM_CREATE i zwróć sterowanie

        case WM_SIZE:
            // obsłuż komunikat WM_SIZE i zwróć sterowanie

        case WM_COMMAND:
            // obsłuż komunikat WM_COMMAND i zwróć sterowanie

        case WM_DESTROY:
            // obsłuż komunikat WM_DESTROY i zwróć sterowanie

        case WM_PAINT:
            // obsłuż komunikat WM_PAINT i zwróć sterowanie

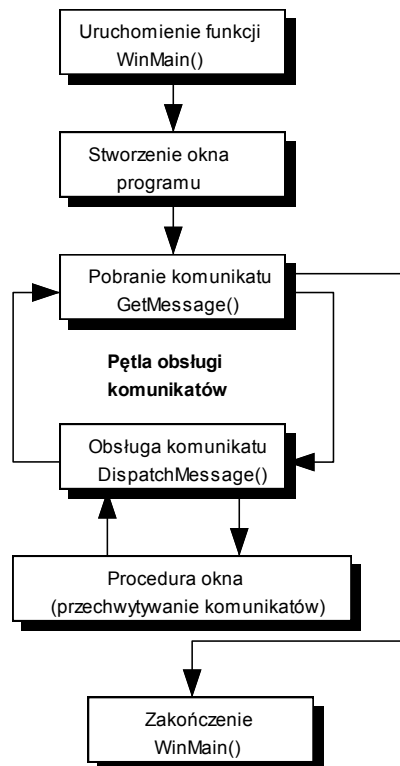
        case WM_CHAR:
            // obsłuż komunikat WM_CHAR i zwróć sterowanie

        default:
            return DefWindowProc (hwnd, iMsg, wParam, lParam);
    }
}

```

Parametr `iMsg` zawiera wartość, która identyfikuje typ komunikatu. Komunikat wysyłany przez system kiedy okno programu potrzebuje narysowania lub ponownego narysowania ma identyfikator `WM_PAINT`. Komunikat `WM_COMMAND` przesyłany jest gdy użytkownik wybiera jedną z opcji menu programu. Komunikat `WM_DESTROY` przesyłany jest gdy okno programu ma zostać zniszczone. Komunikat `WM_SIZE` pojawia się w chwili, gdy użytkownik zmienił rozmiary okna programu. Komunikat `WM_CREATE` przesyłany jest do programu tuż przed pierwszym narysowaniem okna programu. Komunikat `WM_CHAR` jest jednym ze sposobów przesłania informacji przez system, że naciśnięty został klawisz na klawiaturze komputera. Procedura `WndProc()` dokonuje identyfikacji komunikatów i kieruje programem w taki sposób aby dany komunikat został obsłużony. W typowym przypadku procedura `WndProc()` przechwytuje tylko kilka istotnych dla aplikacji komunikatów, pozostałe zaś przekazuje funkcji `DefWindowProc()`, która w sposób domyślny dla systemu przetwarza pozostałe komunikaty przekazywane do okna. W programie szablon1 obsługiwane są komunikaty `WM_COMMAND`, `WM_PAINT` i `WM_DESTROY`. Obsługa tych komunikatów zostanie omówiona poniżej.

Kiedy procedura okna zwraca sterowanie (wywoływana jest instrukcja `return`), sterowanie przekazywane jest ponownie do pętli komunikatów i program wywołuje ponownie funkcję `GetMessage()` w celu pobrania następnego komunikatu. W przypadku, gdy program ma być zatrzymany i usunięty z pamięci otrzymuje on specjalny komunikat (`WM_QUIT`) co spowoduje, że funkcja `GetMessage()` zwraca wartość 0 i pętla obsługi komunikatów jest zakończona. Po zakończeniu działania pętli obsługi komunikatów funkcja `WinMain()` zwraca określoną wartość do systemu i kończy działanie. Jeśli w działaniu programu występuje błąd zanim przejdzie on do pętli wykonywania komunikatów powinien on zwrócić do systemu wartość 0. W przypadku normalnego zakończenia działania programu funkcja `WinMain()` zwraca odpowiedni kod wyjściowy do systemu. Rysunek 2.3.1 prezentuje procesy sterujące pracą aplikacji w Windows omawiane w tym podrozdziale.



Rys. 2.3.1.1. Przepływ sterowania w programie szablon1

2.3.2 Tworzenie okna programu

Tworzenie okna składa się z dwu etapów. Po pierwsze funkcja WinMain() wywołuje funkcję RegisterClass() w celu zarejestrowania w systemie tak zwanej klasy okna. Następnie wywoływana jest funkcja CreateWindow() tworząca bieżące okno. Klasa okna zawiera ogólny opis okna programu. Kiedy program dokona rejestracji klasy okna, może on na jej podstawie stworzyć jedno lub więcej okien należących do tej klasy. Każde ze stworzonych okien przejmuje cechy charakterystyczne dla zdefiniowanej klasy oraz pewne dodatkowe atrybuty podawane w momencie tworzenia okna. (Uwaga: Pojęcie klasy okna nie ma nic wspólnego z pojęciem klasy w języku C++.)

Klasę okna definiuje się poprzez wypełnienie pól specjalnej struktury typu WNDCLASS:

```

WNDCLASS wndclass;    // struktura do konstruowania klasy okna

// Tworzenie klasy głównego okna:
wndclass.style=CS_HREDRAW|CS_VREDRAW; // styl klasy okna
wndclass.lpfnWndProc=WndProc;         // nazwa głównej
procedury                               // sterującej okna
wndclass.cbClsExtra=0;                 // dodatkowy obszar
pamięci
wndclass.cbWndExtra=0;                 // dodatkowy obszar
pamięci
wndclass.hInstance=hInstance;         // instancja programu
wndclass.hIcon=LoadIcon(NULL, IDI_APPLICATION); // ikona klasy
okien
  
```



```

        WS_OVERLAPPEDWINDOW, // Styl okna
        CW_USEDEFAULT, // położenie okna x
        CW_USEDEFAULT, // położenie okna y
        CW_USEDEFAULT, // wysokość okna
        CW_USEDEFAULT, // szerokość okna
        NULL, // uchwyt do okna rodzica
        NULL, // uchwyt do menu lub identy-
                // fikator okna dziecka
        hInstance, // uchwyt do instancji programu
        NULL // uchwyt do danych tworzących
okno
    );

```

Pierwszy parametr wywołania funkcji jest nazwą zdefiniowanej klasy okna, do której należy ma tworzone okno. W rezultacie stworzone okno będzie posiadało wszystkie cechy zdefiniowane w zarejestrowanej klasie. Drugi parametr ("Szablon1") jest napisem, jaki pojawi się w pasku tytułu okna programu. Trzeci parametr definiuje styl okna. Podobnie jak w przypadku definiowania stylu klasy, styl okna można ustalić podając zestawienie stałych o nagłówku `WS_` połączonych logicznym operatorem `lub`. W programie `szablon1` styl okna (`WS_OVERLAPPEDWINDOW`) ustala, że tworzone okno posiadać będzie pasek tytułu, menu systemowe, krawędź pozwalającą na zmianę jego wielkości, przycisk minimalizacji, przycisk maksymalizacji. Cztery następne parametry pozwalają na zdefiniowanie początkowego rozmiaru okna oraz jego początkowego położenia na ekranie (Postawienie stałych `CW_DEFAULT` pozwala systemowi na ustalenie tych parametrów automatycznie i zapewnia, że okno na pewno pojawi się na ekranie.). Parametry wywołania funkcji przechowujące uchwyt do okna rodzica i uchwyt do menu w programie `szablon1` ustawiono na `NULL`, ponieważ okno programu jest głównym oknem i nie posiada żadnego okna nadrzędnego oraz ponieważ menu dla okna zdefiniowane zostało w klasie okna. Przedostatnim parametrem wywołania funkcji jest uchwyt do instancji programu przechwycony z systemu przez główną funkcję programu. Ostatni parametr wywołania funkcji jest wskaźnikiem na obszar danych, którymi można się posługiwać w czasie działania programu. W programie `szablon1` ustawiono go na `NULL`. Funkcja `CreateWindow()` zwraca uchwyt do nowo utworzonego okna (typ danych `HWND`).

Koniec działania funkcji `CreateWindow()` oznacza, że Windows utworzył okno. Jednakże okno nie pojawi się jeszcze na ekranie. Potrzebne są dalsze dwa wywołania. Pierwsze z nich to:

```
ShowWindow(hwnd, iCmdShow); // przygotuj okno do wyświetlenia
```

Pierwszym parametrem funkcji jest uchwyt utworzonego okna (`hwnd`). Drugi zaś (`iCmdShow`) w programie `szablon1` jest jednym z parametrów wywołania funkcji `WinMain()`. Parametr ten określa w jakiej formie okno po raz pierwszy pojawi się na ekranie. Jeśli `iCmdShow` ma wartość `SW_SHOWNORMAL` (wartość domyślna), to okno zostanie wyświetlone normalnie. Natomiast gdy `iCmdShow` jest równe `SW_SHOWMINNOACTIVE`, okno nie zostanie wyświetlone na ekranie, a jedynie jego nazwa i ikona pojawią się na pasku zadań. Funkcja `ShowWindow()` umieszcza okno na ekranie. Jeśli wartość drugiego parametru funkcji wynosi `SW_SHOWNORMAL`, obszar roboczy okna będzie wyczyszczony pędzlem określonym przez klasę okna. Wywołanie funkcji:

```
UpdateWindow(hwnd); // wyślij pierwszy komunikat WM_PAINT
```

powoduje, że obszar roboczy okna zostanie wypełniony przez program. Jest to związane z wysłaniem do procedury okna (funkcji WndProc()) komunikatu WM_PAINT (obsługa poszczególnych komunikatów występujących w przykładowym programie zostanie omówiona poniżej).

2.3.3 Rysowanie okna

Główne okno programu otrzymuje komunikat o identyfikatorze WM_PAINT jeśli kiedykolwiek zachodzi w systemie taka sytuacja, że okno programu wymaga narysowania lub ponownego narysowania (np. gdy tworzone jest nowe okno lub gdy fragment okna uprzednio zasłonięty przez inne okno ma być ponownie narysowany). W programowaniu w Windows istnieje reguła, że jeśli to jest możliwe, wszystkie instrukcje dotyczące wypełniania obszaru roboczego okna (rysowania w obszarze roboczym okna) powinny znajdować się w obsłudze komunikatu WM_PAINT. W programie szablon1 obsługa komunikatu wygląda następująco:

```
case WM_PAINT: // Komunikat przychodzi w przypadku
konieczności
                // "odmalowania" lub namalowania okna
                // Wymaż okno, uzyskaj kontekst urządzenia:
                HDCPaint=BeginPaint(hwnd,&PaintStruct);
                // Pobierz rozmiary obszaru roboczego okna
                GetClientRect(hwnd,&rect);
                // Umieść centralnie w oknie tekst:
                DrawText(HDCPaint,"Pozdrowienia",-1,&rect,
                        DT_SINGLELINE|DT_CENTER|DT_VCENTER);
                // Zakończ rysowanie
                EndPaint(hwnd, &PaintStruct);
                return 0;
```

Odmalowywanie okna rozpoczyna się zwykle od wywołania funkcji BeginPaint(), a kończy wywołaniem funkcji EndPaint(). W obydwu przypadkach pierwszym parametrem funkcji jest uchwyt okna programu, drugi zaś wskaźnikiem do struktury typu PAINTSTRUCT. Struktura PAINTSTRUCT zawiera pewne informacje przydatne przy kreśleniu w obszarze roboczym okna. Funkcja BeginPaint() czyści obszar roboczy okna przy użyciu pędzla zdefiniowanego podczas tworzenia klasy okna.

Funkcja zwraca także uchwyt do kontekstu urządzenia (ang. device context, typ danych: HDC), który umożliwia programowi rysowanie w obszarze roboczym okna. W systemie Windows program nigdy nie wyświetla grafiki czy tekstu bezpośrednio na danym urządzeniu (ekranie, drukarce, ploterze). W zamian za to obiekty graficzne rysowane są na abstrakcyjnej powierzchni zwanej kontekstem urządzenia. Każdy kontekst urządzenia powiązany jest z fizycznym urządzeniem i rysowanie w kontekście w rezultacie powoduje rysowanie na konkretnym urządzeniu. Sam kontekst urządzenia dodatkowo przechowuje szereg właściwości dotyczących sposobu rysowania obiektów na urządzeniu z którym jest związany. W przypadku wywoływania jakiegokolwiek funkcji rysującej tekst lub grafikę na ekranie wymaga ona przekazania do niej uchwyty kontekstu urządzenia, na którym chcemy uzyskać obraz. Kolejną funkcją wywoływaną w obsłudze komunikatu jest GetClientRect(). Funkcja przechwytyje rozmiary obszaru roboczego okna i umieszcza je w strukturze typu RECT. Zawartość struktury

jest następnie przekazywana do kolejnej funkcji DrawText(). DrawText() rysuje na środku obszaru roboczego napis "Pozdrowienia". Należy ona do grupy funkcji służących do wyprowadzania tekstów wewnątrz okien.

Na zakończenie obsługi komunikatu wywoływana jest funkcja EndPaint(), która zwalnia przechwycony przez funkcję BeginPaint() kontekst urządzenia (W dobrym stylu programowania w systemie Windows jest zwracanie systemowi przechwyconego uchwytu kontekstu urządzenia. Wynika to z faktu, że system dysponuje ograniczoną liczbą uchwytów.) i przerywa proces rysowania w oknie. Następnie procedura obsługi komunikatów okna zwraca systemowi wartość 0 co oznacza, że zakończona została obsługa komunikatu.


2.3.4 Obsługa poleceń menu

Jesli kiedykolwiek użytkownik programu używa komend menu, główne okno programu otrzymuje komunikat WM_COMMAND. Dodatkowo parametr wParam przekazywany do procedury okna zawiera **identyfikator opcji menu** (Jest to ten sam identyfikator, który został przyłączony do opcji menu w momencie tworzenia menu przy pomocy edytora graficznego i pamiętany jest w pliku skryptu zasobów oraz w pliku **resources.h** włączonym do pliku szablon1.c). Obsługa komunikatu WM_COMMAND rozpoczyna się od identyfikacji opcji menu, która została wybrana przez użytkownika. Ponieważ identyfikator opcji menu zawarty jest w młodszych 16 bitach słowa wParam numer identyfikatora uzyskuje się przy pomocy makrorozwinięcia LOWORD():

```
case WM_COMMAND: // Komunikat przychodzący po wybraniu
                  // opcji menu
    switch(LOWORD(wParam))
    {
        // Wybrano opcję menu Plik->Koniec:
        case ID_PLIK_KONIEC:
            DestroyWindow(hwnd);
            return 0;
    }
    return 0;
```

Wybranie opcji Plik → Koniec w programie szablon1 powoduje zamknięcie okna. Jednym ze sposobów wydania polecenia zamknięcia okna jest wywołanie funkcji DestroyWindow(), której parametrem jest uchwyt okna, które ma być zamknięte.

2.3.5 Zakończenie pracy programu

Kiedy użytkownik klika przycisk zamknięcia okna (), lub wybiera opcję Zamknij w menu systemowym okna, okno główne otrzymuje komunikat WM_CLOSE. Procedura okna WndProc() nie zawiera własnej obsługi tego komunikatu. Obsługa przekazywana jest do funkcji DefWindowProc(). Funkcja DefWindowProc() komunikat WM_CLOSE obsługuje w taki sposób, że wywołuje automatycznie funkcję DestroyWindow() (Identyczna funkcja wywoływana jest przy obsłudze opcji Koniec z menu Plik programu szablon1- rozdział 2.3.4). Kiedy okno ma być zniszczone otrzymuje ono komunikat WM_DESTROY. Obsługa tego komunikatu w programie szablon1 wygląda następująco:

```
case WM_DESTROY: // Komunikat przychodzący gdy okno
```

```

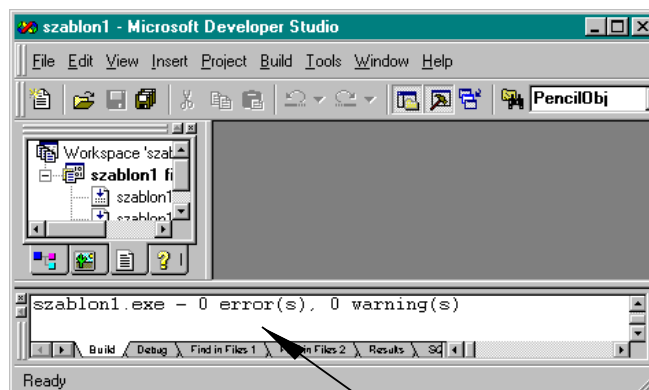
// ma być zniszczone
PostQuitMessage(0);
return 0;

```

Obsługę komunikatu WM_DESTROY można wykorzystać do między innymi do zwolnienia dynamicznie zarezerwowanych obszarów pamięci, a także do zwolnienia uchwytów urządzeń wykorzystywanych w programie. W programie szablon1 w obsłudze komunikatu WM_DESTROY wywoływana jest jedynie funkcja PostQuitMessage(). Funkcja ta wysyła do głównego okna programu komunikat WM_QUIT, który jak wcześniej wyjaśniono powoduje wyjście programu z pętli obsługi komunikatów i zakończenie działania programu. Jeśli program nie zawiera obsługi komunikatu WM_DESTROY, lub nie wywołuje funkcji PostQuitMessage(), okno programu zostaje zniszczone, natomiast sam program egzystuje w pamięci i zajmuje zasoby systemu.

2.4 Kompilacja, konsolidacja i uruchamianie programu

Jeśli w systemie Visual C++ 5.0 znajduje się załadowany i ukończony projekt, to kompilację projektu uruchamia się klikając Opcję *Build nazwa projektu* z menu *Build* (klawisz skrótów - F7). Raport z kompilacji wypisywany jest w automatycznie pojawiającym się oknie "Output" (rys 2.4.1). Kliknięcie w linię raportu informującą o ewentualnych błędach w kompilacji lub konsolidacji powoduje wywołanie odpowiedniego dokumentu z projektu i wskazanie linii, w której kompilator wykrył nieprawidłowość. Jeśli kompilacja i konsolidacja projektu przebiega bez zgłoszenia błędów, można uruchomić program klikając opcję *Execute nazwa projektu* z menu *Build* (klawisz skrótów Ctrl+F5).



Okno "Output" z raportem z kompilacji i konsolidacji

Rys 2.4.1 Okno "Output" pakietu Visual C++ 5.0

3. Modyfikowanie, tworzenie projektów

Środowisko Visual C++ 5.0 dostosowane jest do pracy z projektami programów. Nie jest możliwe załadowanie do pakietu pojedynczego pliku tekstowego, skompilowanie go i uzyskanie gotowego programu. Zawsze stworzenie nowej aplikacji poprzedzone musi być zdefiniowaniem projektu w sposób podobny, jak było to omówione w punkcie 2.1 opracowania.

Pakiet Visual C++ 5.0 umożliwia kompilowanie i uruchamianie aplikacji napisanych w języku ANSI C, czy C++ (Programy nie wykorzystujące formalizmu zgodnego z projektowaniem

aplikacji dla środowiska Windows). W celu skompilowania takiego programu należy stworzyć specjalny rodzaj projektu. Nowy projekt tworzy się wybierając z menu *File* opcję *New*. Na ekranie pojawi się okno dialogowe *New*. Po wybraniu zakładki *Projects* należy zamiast elementu *Win 32 Application* wybrać *Win 32 Console Application*. Do tak utworzonego projektu należy włączyć plik źródłowy w języku ANSI C lub C++ w sposób pokazany w punkcie 2.2 opracowania. Kompilowanie, konsolidowanie i uruchamianie aplikacji przebiega tak samo jak opisano to w punkcie 2.4.

Stworzone uprzednio projekty aplikacji można w całości wczytywać do systemu programowania. Służy do tego opcja *Open Workspace...* z menu *File*. Uruchomienie tej opcji powoduje pojawienie się typowego okna "Otwórz" systemu Windows. Plikami, które wystarczy otworzyć, aby do pakietu załadowany został cały projekt aplikacji są pliki z rozszerzeniami **dsw**. Opcja *SaveWorkspace* z menu *File* umożliwia zachowanie wszystkich składników projektu. Natomiast opcja *CloseWorkspace* z menu *File* zamyka projekt otwarty w systemie.

Bibliografia:

- [1] *Introduction to Graphics Programming for Windows 95*, M.J. Young, 1996 AP PROFESSIONAL.
- [2] *Programowanie Windows 95*, C. Petzolt, 1997 Oficyna Wydawnicza READ ME
- [3] *Visual C++*, S. Holzner, 1998 Wydawnictwo HELION

Przebieg ćwiczenia:

1. Uruchomić program *szablon1.exe* i zapoznać się z jego obsługą.
2. Stworzyć własną aplikację Windows na podstawie programu *szablon*:
 - a. Utworzyć nowy projekt,
 - b. Stworzyć menu dla programu,
 - c. Dołączyć plik zasobów do projektu,
 - d. Przekopiować plik źródłowy „szablon1.c” do folderu z plikami nowego projektu
 - e. Dołączyć plik *szablon1.c* do projektu,
 - f. Jeśli zachodzi konieczność, zmodyfikować stałe symboliczne przywiązane do opcji menu w taki sposób, żeby były one identyczne w pliku *szablon1.c* i w edytorze zasobów (chodzi o stałe symboliczne *ID_PLIK_KONIEC* i *IDR_MENU1*),
 - g. Skompilować i uruchomić program.
3. Zmodyfikować aplikację Windows w taki sposób, że stworzone zostaną dwie dodatkowe opcje menu i pod wpływem ich wybrania zmieniać się treść napisu umieszczonego w oknie.