

***Programowanie wizualne  
Borland C++ Builder***

***Materiały szkoleniowe***

# 1. Spis treści

|   |           |
|---|-----------|
| <b>1. Spis treści</b> .....                                       | <b>2</b>  |
| <b>2. Wstęp</b> .....   | <b>4</b>  |
| <b>3. Środowisko Borland C++ Builder</b> .....                    | <b>5</b>  |
| <b>Struktura głównego menu</b> .....                              | <b>7</b>  |
| Menu File .....   | 8         |
| Menu Edit .....   | 10        |
| Menu Search .....   | 13        |
| Menu View .....   | 14        |
| Menu Project .....  | 17        |
| Menu Run .....  | 19        |
| Menu Component .....  | 21        |
| Menu Tools .....  | 22        |
| Menu Help .....   | 24        |
| Menu Desktop .....  | 25        |
| <b>Pasek narzędzi – Speed Bar</b> .....                           | <b>26</b> |
| <b>Inspektor obiektów – Object Inspector</b> .....                | <b>26</b> |
| Karta właściwości – Properties .....                              | 26        |
| Karta obsługi zdarzeń – Events .....                              | 27        |
| <b>4. Pierwszy program konsolowy</b> .....                        | <b>28</b> |
| <b>Ćwiczenia</b> .....  | <b>29</b> |
| <b>5. Pierwszy program graficzny</b> .....                        | <b>30</b> |
| <b>Ćwiczenia</b> .....  | <b>30</b> |
| <b>6. Elementy składowe aplikacji w Borland C++ Builder</b> ..... | <b>32</b> |
| <b>Klasa TForm1</b> .....   | <b>33</b> |
| <b>Konstruktor TForm1()</b> .....                                 | <b>34</b> |
| <b>Formularz jako zmienna obiektowa</b> .....                     | <b>34</b> |
| <b>7. Biblioteka Visual Component Library</b> .....               | <b>36</b> |
| <b>Karta Standard</b> .....                                       | <b>37</b> |
| <b>Karta Additional</b> .....                                     | <b>38</b> |
| <b>Karta Win32</b> .....  | <b>38</b> |
| <b>Karta System</b> .....   | <b>39</b> |
| <b>Karta Dialogs</b> .....  | <b>40</b> |
| <b>Karta Win 3.1</b> .....  | <b>40</b> |
| <b>Karta Samples</b> .....  | <b>41</b> |
| <b>Karta ActiveX</b> .....  | <b>42</b> |
| <b>Karta Internet</b> .....                                       | <b>42</b> |
| <b>Karta Servers</b> .....  | <b>43</b> |
| <b>Karta Data Access</b> .....                                    | <b>43</b> |
| <b>Karta Data Controls</b> .....                                  | <b>44</b> |

|   |           |
|---|-----------|
| <b>Ćwiczenia .....</b>  | <b>45</b> |
| <b>8. Podstawowe elementy biblioteki VCL i hierarchia komponentów .....</b> | <b>60</b> |
| <b>Hierarchia komponentów VCL .....</b>                                     | <b>60</b> |
| Klasa TObject.....  | 60        |
| Klasa TPersistent.....  | 61        |
| Klasa TComponent.....   | 61        |
| Klasa TControl .....  | 61        |
| Klasa TGraphicControl .....   | 63        |
| Klasa TWinControl .....   | 64        |
| <b>9. Aplikacje SDI oraz MDI.....</b>                                       | <b>65</b> |
| <b>Aplikacje jednodokumentowe.....</b>                                      | <b>65</b> |
| <b>Aplikacje wielodokumentowe .....</b>                                     | <b>66</b> |
| <b>Literatura .....</b>   | <b>68</b> |
| <b>Zaliczenie .....</b>   | <b>69</b> |

## **2. Wstęp**

We współczesnym świecie informatyki wiodącą rolę odgrywają aplikacje graficzne. Zarówno te działające pod kontrolą systemów Windows jak i te uruchamiane w środowisku graficznym systemów Unix/Linux. Celem zawartego w tym dokumencie materiału jest przedstawienie możliwości tworzenia aplikacji z wykorzystaniem narzędzi wizualnych. Na rynku dostępnych jest obecnie wiele produktów umożliwiających programowanie wizualne. Są to między innymi: Microsoft Visual Studio, Borland Delphi, Borland C++ Builder, Borland J++ Builder, Borland X Builder, IBM Visual Age for Java i wiele innych. Umożliwiają one wykorzystanie języków programowania wysokiego poziomu (np. C++, Java, Pascal) w połączeniu z technikami programowania wizualnego. Każde z tych środowisk programistycznych jest bardzo rozbudowaną aplikacją dlatego też nie jest możliwe nawet przekrojowe zaprezentowanie ich wszystkich. Programowanie wizualne będzie prezentowane na podstawie produktu firmy Borland o nazwie C++ Builder w wersji 5.

Do pełnego zrozumienia omawianych tematów potrzebna jest podstawowa wiedza w zakresie programowania w językach C i C++. Materiału zawartego w tej pracy nie należy traktować jako kompletnego podręcznika wyczerpującego tematykę tworzenia aplikacji w oparciu o środowisko Borland C++ Builder. Autor ma jednak nadzieję, że będzie to dobry punkt wyjścia do dalszej nauki.

### 3. Środowisko Borland C++ Builder

Borland C++ Builder jest zintegrowanym środowiskiem programistycznym (IDE – ang. *Integrated Development Environment*), stanowiącym zbiór niezbędnych narzędzi pomocnych w szybkim tworzeniu aplikacji. Zawiera rozbudowane edytory tekstowe i graficzne, kompilator, linker oraz inne narzędzia pomocnicze. Przy wykorzystaniu C++ Buildera możliwe jest tworzenie aplikacji graficznych, bibliotek dll, kontrolek ActiveX. Z racji tego, że kompilator C++ Builder v. 5.5 jest zgodny z standardami ANSI/ISO języka C++ możliwe jest też budowanie aplikacji konsolowych. Pakiet Borland C++ Builder jest dostępnych w trzech wersjach. Poniżej przedstawiono ich krótką charakterystykę:

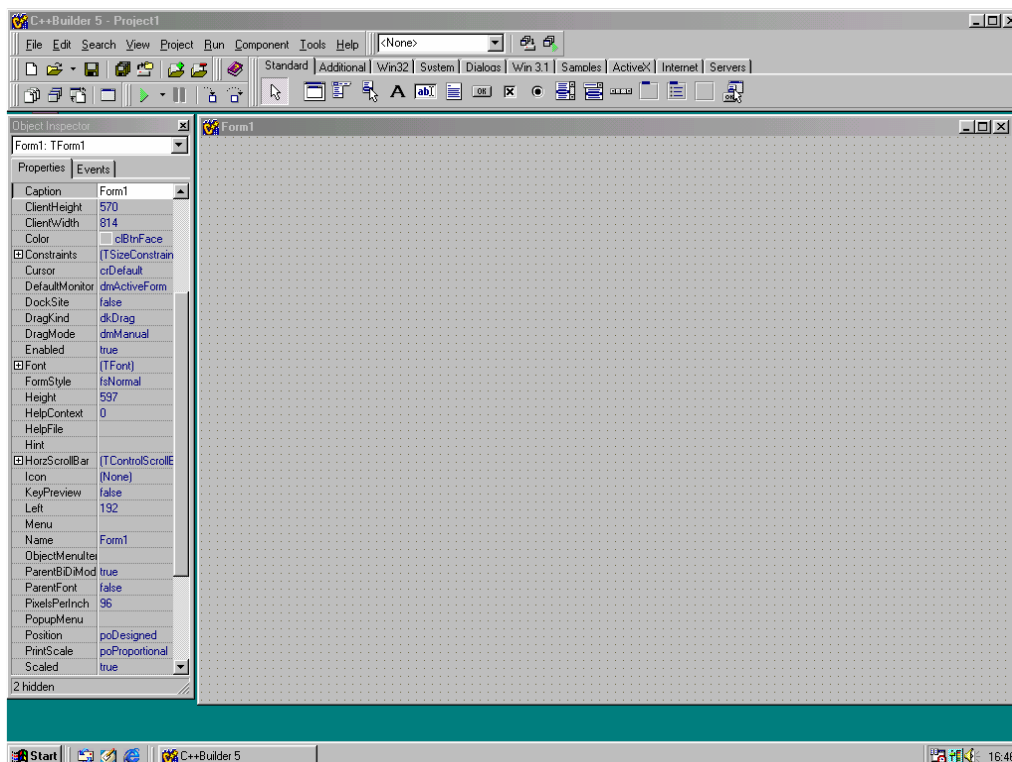
- C++ Builder Enterprise – głównym jego zastosowaniem jest tworzenie aplikacji rozproszonych, internetowych oraz typu klient/serwer. Wbudowane komponenty Internet Express, zawierające kreatory klientów internetowych, bardzo ułatwiają tworzenie w pełni skalowalnych aplikacji, zdolnych dynamicznie przysyłać dane poprzez WWW. Programista ma do dyspozycji języki HTML 4 i XML. Tworzenie aplikacji rozproszonych ułatwiają MIDAS, PageProducer oraz WebBroker. ADOExpress zapewnia bardzo szybki dostęp do danych praktycznie rzecz biorąc z dowolnych źródeł. Tworzone w ten sposób aplikacje będą działać na różnych platformach internetowych. Większa wydajność pracy grup programistów została zapewniona przez TeamSource. Mamy tutaj możliwości grupowania projektów wraz z ich jednoczesną kompilacją.
- C++ Builder Professional – posługując się tą wersją mamy możliwość szybkiego tworzenia aplikacji sieciowych poprzez wbudowane biblioteki elementów internetowych oraz perfekcyjnie zorganizowaną obsługę baz danych. Posługując się technologią CodeGuard można zminimalizować występowanie różnego rodzaju błędów alokacji i dostępu do pamięci. Wykorzystanie komponentów Frame pozwala na efektywne, wizualne tworzenie komponentów biznesowych. Budowanie aplikacji posługującej się relacyjnymi bazami danych ułatwia InterBase Express.
- C++ Builder Standard – jest podstawową dystrybucją C++ Builder. Do dyspozycji programisty jest kilkadziesiąt komponentów wizualnych oferowanych przez biblioteki VCL (ang. *Visual Component Library*). Wersja Standard udostępnia wszystkie niezbędne zasoby interfejsu programi WinAPI (ang. *Windows Application Programming Interface*). Dzięki niej mamy możliwość wykorzystywania zaawansowanych technologii obiektowych, takich jak COM czy ActiveX. Z kolei OLE Automation pozwala na

współpracę z elementami pakietu MS Office, np. Word, Excel, Power Point, Outlook.

Główne elementy graficznego środowiska do budowy aplikacji stanowią:

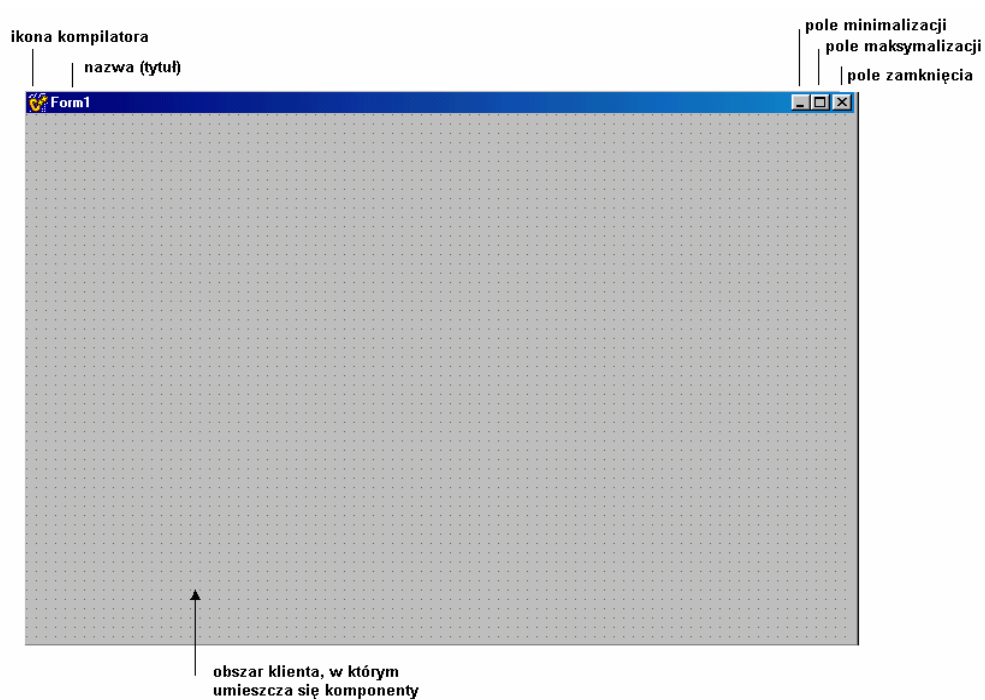
- Główne menu.
- Pasek narzędzi.
- Główny formularz.
- Okno edycji kodu.
- Inspektor obiektów (ang. *Object Inspector*).

Po uruchomieniu programu C++Builder 5 okno monitora powinno wyglądać podobnie jak na poniższym rysunku:



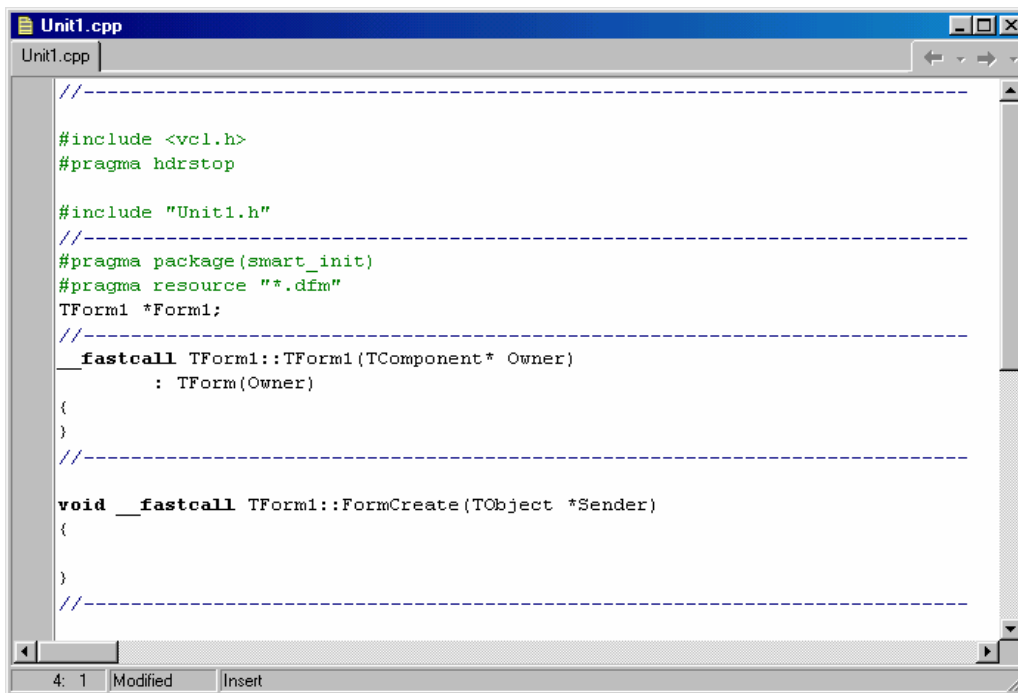
Rys. 1. Główne okno Borland C++ Builder 5.

Jeśli po uruchomieniu programu nie pojawi się formularz o nazwie Form1, a użytkownik chce rozpocząć tworzenie nowej aplikacji, to należy wybrać z menu opcję File->New Application lub File->New->New->Application. Po wybraniu jednej ze wskazanych opcji centralną część ekranu zajmować będzie obszar zwany formularzem lub obszarem klienta, którego nazwa domyślnie przyjmowana jest jako Form1. Formularz posiada wszystkie cechy standardowego okna Windows. Już w tym momencie możemy uruchomić aplikację naciskając klawisz F9 lub wskazując w menu opcję Run->Run. Po uruchomieniu użytkownik otrzymuje puste okienko posiadające wszystkie właściwości standardowego okna Windows:



Rys. 2. Podstawowe elementy formularza w C++ Builder.

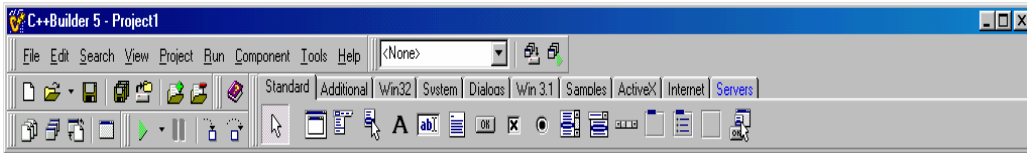
W celu wywołania okna do edycji kodu należy dwukrotnie kliknąć na formularzu lub nacisnąć klawisz F12:



Rys. 3. Okno edycji kodu.

## Struktura głównego menu

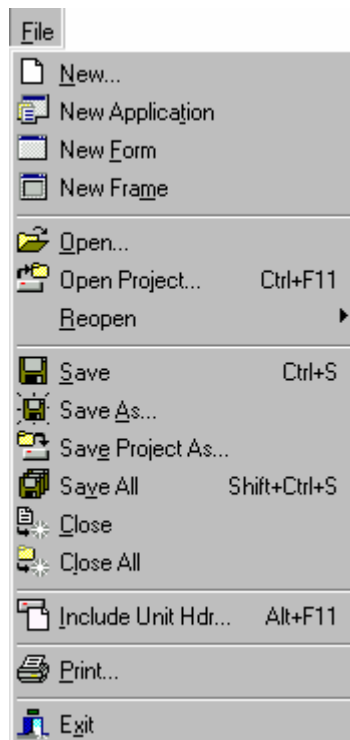
Aby sprawnie tworzyć aplikacje w środowisku C++ Builder należy poznać dostępne opcje programu. W pierwszej kolejności zaprezentowana zostanie struktura menu głównego:



Rys. 4. Menu główne.

## Menu File

W menu File mamy do dyspozycji następujące opcje:



Rys. 5. Menu File

### **New...**

Polecenie umożliwia utworzenie nowego projektu, formularza, okna dialogowego lub otwiera przykładowe projekty aplikacji.

### **New Application**

Polecenie pozwala utworzyć nowy projekt. Nowo powstały projekt składa się z pustego formularza o nazwie Form1 oraz odpowiadającego mu modułu o nazwie *Unit1.cpp*.



### **New Form**

Polecenie tworzy nowy, pusty formularz.

### **New Frame**

Polecenie pozwala utworzyć nową ramkę.

### **Open...**

Polecenie otwarcia modułu, obiektu lub projektu. Katalogiem domyślnym będzie katalog, w którym zainstalowany jest Builder.

### **Open Project...**

Polecenie umożliwia otwarcie zapisanego wcześniej na dysku projektu.

### **Reopen**

Wyświetlenie listy ostatnio używanych projektów, z których każdy można natychmiast otworzyć.

### **Save**

Polecenie pozwala zapisać bieżący moduł na dysku. Domyślnie plik ten będzie miał rozszerzenie *\*.cpp*.

### **Save As...**

Zapisanie wybranego modułu pod nową nazwą. Dobrym zwyczajem jest zapisywanie kolejnych modułów pod innymi nazwami.

### **Save Project As...**

Polecenie umożliwiające zapisanie aktualnie otwartego projektu pod inną nazwą.

### **Save All**

Zapisanie na dysku wszystkich aktualnie otwartych plików C++Buildera.

### **Close**

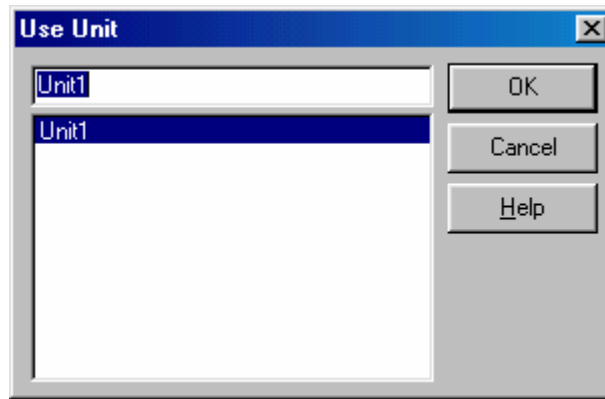
Zamknięcie aktualnie używanego modułu kodu *\*.cpp* wraz z odpowiadającym mu formularzem.

### **Close All**

Zamknięcie wszystkich modułów aktualnie otwartego projektu.

### **Include Unit Hdr...**

Umożliwia dołączenie do aktualnie używanego modułu kodu nowego pliku nagłówkowego. Jeżeli aktualnie pracujemy z formularzem Form2, któremu odpowiada moduł *Unit2.cpp* i zechcemy dołączyć moduł *Unit1.cpp*, wówczas użycie tego polecenia spowoduje wyświetlenie następującego okna:



Rys. 6. Dołączanie nowego modułu

### **Print...**

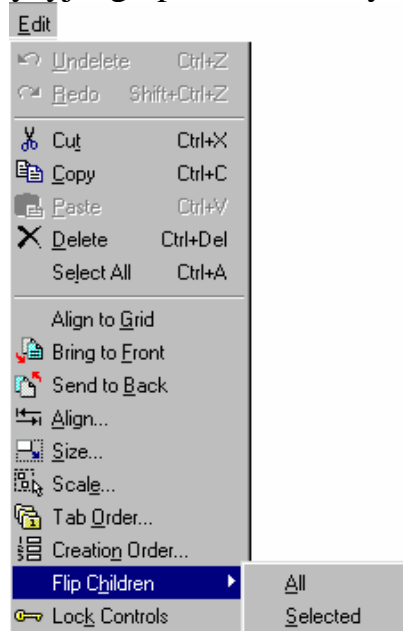
Polecenie drukowania aktualnie używanego elementu projektu. Gdy zechcemy wydrukować zawartość okna edycji kodu pojawi się opcja Print Selection. W przypadku drukowania formularza ujrzymy okienko Print Form.

### **Exit**

Opuszczenie C++Buildera i ewentualne zapisanie wszystkich otwartych elementów aplikacji.

### **Menu Edit**

Pełne rozwinięcie menu edycyjnego pokazano na rysunku 1.7.



Rys. 7. Menu Edit

### **Undelete**

Podobnie jak we wszystkich standardowych aplikacjach Windows, opcja ta pozwala na anulowanie ostatniej operacji, np. jeżeli przez pomyłkę usunięto jakiś komponent z formularza, używając Undelete możemy cofnąć usuwanie.

### ***Redo***

Polecenie odwrotne w stosunku do Undelete.

### ***Cut***

Umieszcza zaznaczony komponent lub tekst w schowku.

### ***Copy***

Polecenie kopiowania zaznaczonego elementu do schowka. W schowku zostanie umieszczona jedynie jego kopia.

### ***Paste***

Wstawianie uprzednio skopiowanego do schowka obiektu (tekstu, komponentu) we wskazane miejsce pola edycji kodu lub formularza.

### ***Delete***

Pozwala na usunięcie zaznaczonego obiektu.

### ***Select All***

W przypadku edycji kodu źródłowego – zaznaczenie całego tekstu. W przypadku formularza – zaznaczenie wszystkich znajdujących się tam komponentów.

### ***Align to Grid***

Przy pomocy tego polecenia można dopasować położenia wszystkich elementów składowych formularza do jego siatki. Operacja ta będzie dawać widoczne efekty pod warunkiem odznaczenia opcji Snap to Grid w menu Tools->Environment Options->Preferences.

### ***Bring to Front***

Zaznaczony element nie będzie ewentualnie przykrywany przez inne, znajdujące się w formularzu. Element taki będzie zawsze całkowicie widoczny.

### ***Send to Back***

Polecenie odwrotne do Bring to Front.

### ***Align...***

Umożliwia dopasowanie i wyrównanie położenia elementu na formularzu.

### ***Size...***

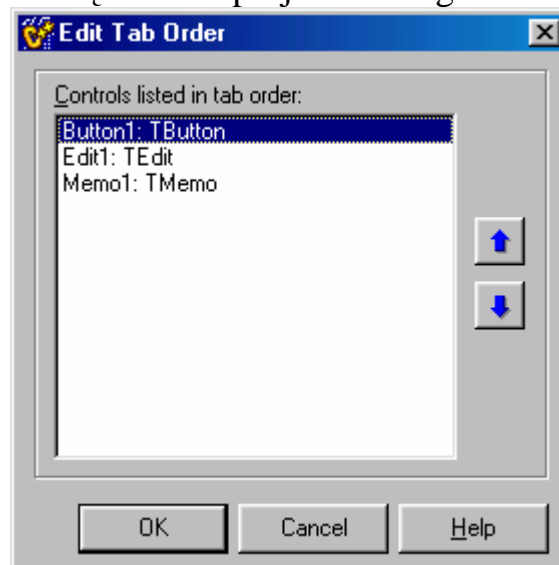
Pozwala ustalić dokładne rozmiary obiektu. Operacja ta może być użyta w stosunku do uprzednio zaznaczonego obiektu lub grupy obiektów.

### **Scale...**

Polecenie przeskalowania formularza jako całości wraz ze wszystkimi elementami wchodzącymi w jego skład.

### **Tab Order...**

Opcja pozwalająca ustalić kolejności przechodzenia pomiędzy składnikami formularza przy użyciu klawisza Tab. Polecenie Tab Order wyświetla okienko dialogowe pokazane na rys. 8. Używając przycisków ze strzałkami można w prosty sposób ustalić kolejność przechodzenia pomiędzy aktualnie dostępnymi elementami, które wchodzi w skład projektowanego formularza.



Rys 8. Okno dialogowe Edit Tab Order.

### **Creation Order...**

Opcja pozwalająca ustalić kolejność tworzenia tzw. komponentów niewidocznych (przestają być widoczne w momencie uruchomienia aplikacji).

### **Flip Children**

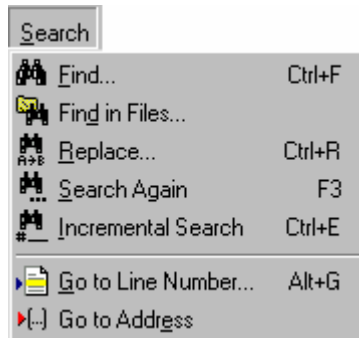
Umożliwia automatyczną zamianę kolejności ułożenia poszczególnych części formularza.

### **Lock Controls**

Wybierając tą opcję zablokujemy możliwość przemieszczania obiektów w obrębie formularza tworzonej aplikacji. Wybranie Lock Controls zapobiega przypadkowej zmianie położenia już wybranego obiektu.

### **Menu Search**

Menu Search zawiera opcję umożliwiającą różnorakie wyszukiwanie:



Rys. 9. Menu Search.

### **Find...**

Pozwala na wyszukanie w kodzie wybranego fragmentu tekstu. Przy pomocy okna dialogowego Find Text określamy żądane parametry wyszukiwania.

### **Find in Files...**

Opcja ta umożliwia przeszukiwanie plików. Przy pomocy zakładki Find in Files określamy żądane parametry wyszukiwania.

### **Replace...**

Umożliwia wyszukanie określonego tekstu lub jego fragmentu i zastąpienie go innym.

### **Search Again**

Wyszukanie kolejnego wystąpienia określonego tekstu lub jego fragmentu.

### **Incremental Search**

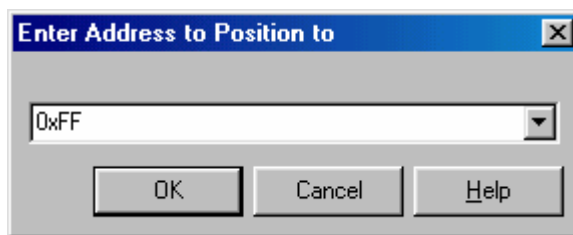
Jest to tzw. opcja niewidoczna. Przed skorzystaniem z jej usług najlepiej jest ustawić kursor na samym początku tekstu kodu. Po wybraniu Search->Incremental Search należy zacząć pisać szukane słowo. Builder odczyta pierwszą literę i natychmiast przeniesie kursor do pierwszego napotkanego w tekście zwrotu zawierającego wpisaną literę.

### **Go to Line Number...**

Przeniesienie kursora do wskazanego wiersza kodu.

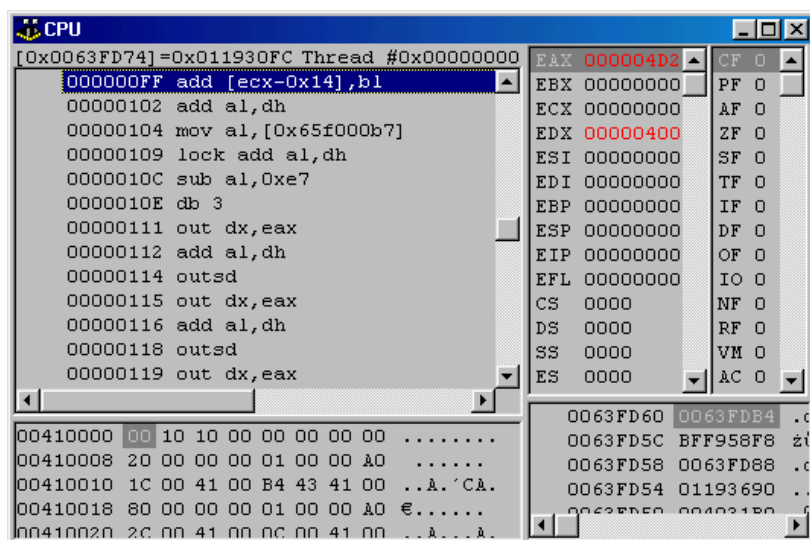
### **Go to Address**

Opcja dostępna w trakcie działania aplikacji. Umożliwia krokowe sprawdzanie wartości zmiennych, rejestrów CPU itp. Po pojawieniu się okienka dialogowego, podobnego do pokazanego na rys. 10, należy wpisać żądaną wartość. Liczby heksadecymalne należy poprzedzić parą znaków 0x.



Rys. 10. Okno dialogowe Enter Address to Position to.

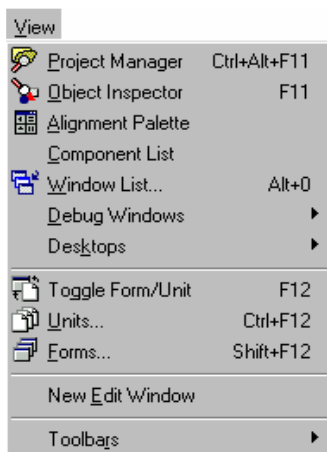
Po potwierdzeniu przyciskiem OK pokazane zostanie okno aktualnego stanu m.in. rejestrów CPU (ang. *Central Processing Unit*). Poruszanie się w oknie CPU możliwe jest dzięki kombinacji klawiszy Ctrl+(prawa/lewa) strzałka:



Rys. 11. Okno dialogowe CPU.

## Menu View

Menu View zawiera opcje odpowiadające za różne aspekty wyświetlania elementów formularza:



Rys. 11. Opcje Menu View.

### **Project Manager**

Polecenie to wywołuje menedżera projektów.

### **Object Inspector**

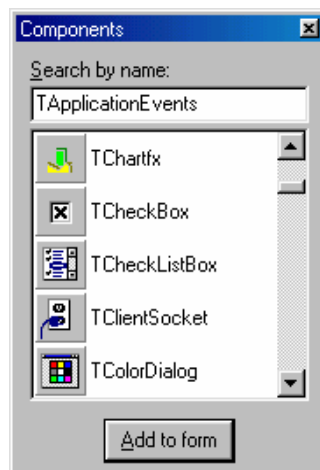
Polecenie odpowiada za wywołanie inspektora obiektów.

### **Alignment Palette**

Opcja umożliwiająca wzajemne ułożenie i dopasowanie komponentów na formularzu. Jest to graficzny odpowiednik opcji Edit->Align.

### **Component List**

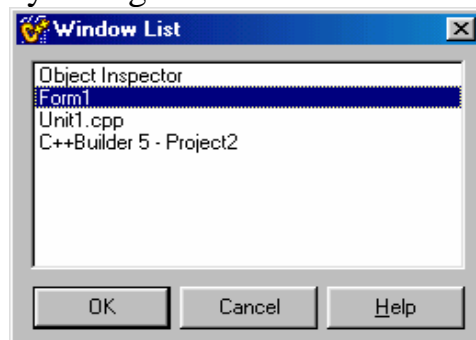
Użycie tego polecenia powoduje uaktywnienie okna zawierającego wszystkie aktualnie dostępne komponenty. Są one ułożone w porządku alfabetycznym. Za pomocą przycisku Add to form dowolny komponent można dodać do formularza:



Rys. 12. Okno Components.

### **Window List...**

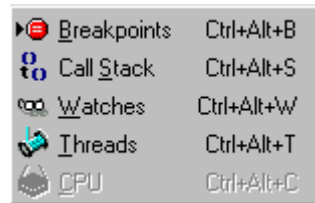
Użycie tego polecenia powoduje uaktywnienie okna dialogowego, w którym pokazana jest lista aktualnie otwartych okien. Zaznaczając odpowiednią pozycję można przenieść się do wybranego okna:



Rys. 13. Okno Window List.

## Debug Windows

W skład Debug Windows wchodzi lista poleceń przedstawiona na poniższym rysunku:

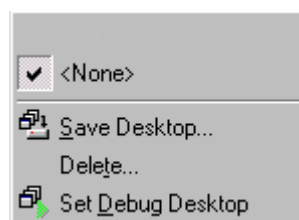


Rys. 14. Opcje Debug Windows

- Breakpoints – wyświetla listę pułapek pomocnych w śledzeniu programu korzystając z debuggera, czyli programu uruchomieniowego. Przy pomocy tego programu mamy możliwość śledzenia krok po kroku oraz możliwość sprawdzania wartości zmiennych i rejestrów procesora.
- Call Stack – opcja ułatwiająca ustalenie kolejności wywoływania funkcji głównego programu podczas działania programu uruchomieniowego.
- Watches – wyświetla okno Watch List, w którym można oglądać aktualne wartości wyrażeń lub zmiennych. Stosowana jest podczas operacji śledzenia wykonywania programu.
- Threads – w okienku Thread Status pojawia się lista aktualnie uruchomionych wątków.
- CPU – wyświetla okienko aktualnego stanu CPU. Opcja ta jest aktywna w czasie działania programu.

## Desktops

Użycie tego polecenia umożliwia skonfigurowanie i zapisanie pod wybraną nazwą wymaganego przez użytkownika wyglądu pulpitu (ang. *Desktop*):



Rys. 15. Opcje Menu Desktops

## Toggle Form/Unit

Daje możliwość przełączenia (ang. *toggle*) pomiędzy edycją formularza a odpowiadającym mu oknem edycji kodu.

## Units...

Polecenie to podaje listę wszystkich modułów należących do projektu.

## Forms...



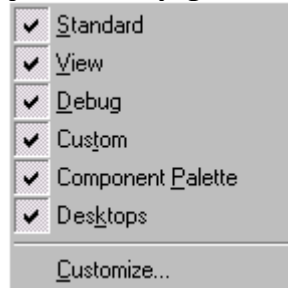
Polecenie umożliwia wyświetlenie listy wszystkich formularzy używanych przez aplikację.

### **New Edit Window**

Polecenie otwarcia kolejnego okna edycji kodu. Dzięki temu możemy pracować z dwoma modułami jednocześnie.

### **Toolbars**

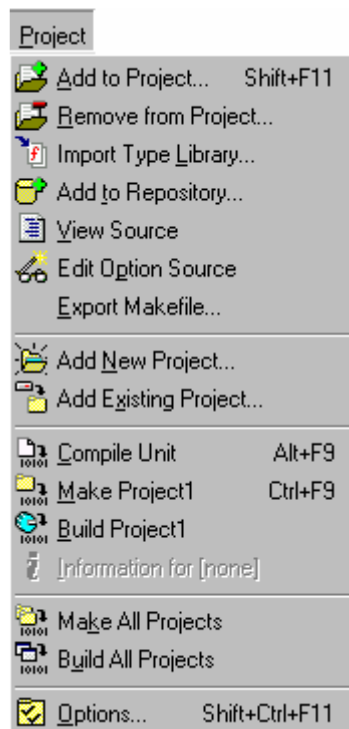
Stwarza możliwość konfiguracji struktury głównego menu:



Rys. 15. Opcje Menu Toolbars.

### **Menu Project**

Opcje dostępne z Menu Project pozwalają na zarządzanie projektem budowanej aplikacji:



Rys. 16. Opcje Menu Project.

### **Add to Project...**

Opcja ta umożliwia włączenie wskazanego modułu do projektu modyfikując automatycznie plik z opisem projektu.

**Remove from Project...**

Usuwa wybrany moduł z projektu modyfikując jednocześnie plik główny projektu.

**Import Type Library...**

Umożliwia zarejestrowanie w środowisku Buildera wybranej biblioteki, która od tej chwili będzie traktowana jak każda składowa biblioteki VCL.

**Add to Repository...**

Powoduje dodanie aktualnie wykorzystywanego formularza do repozytorium.

**View Source**

Polecenie edycji kodu projektu.

**Edit Option Source**

Polecenie edycji wszystkich informacji dotyczących projektu oraz edycji przypisań i odwołań do plików i bibliotek z nim związanych. Wyświetlane są m.in. informacje o środowisku, kompilatorze, standardzie kodu, nazwie pliku wynikowego itp.

**Export Makefile...**

Zapisanie pliku do kompilacji projektu (tzw. pliki *makefile*). Plik taki składa się z ciągu znaków ASCII i zawiera zestaw instrukcji do kompilacji projektu.

**Add New Project...**

Polecenie tworzy nowy projekt w grupie projektów. Opcja ta działa podobnie jak View->Project Manager->New.

**Add Existing Project...**

Przy pomocy tego polecenia można dodać do grupy projektów projekt już istniejący i zapisany wcześniej na dysku.

**Compile Unit**

Umożliwia kompilację modułu projektu.

**Make Project1**

Kompilacja aktualnego projektu w tzw. trybie *Make*. Kompilator kompiluje kody źródłowe wszystkich modułów wchodzących w skład projektu, w których dokonano zmian od czasu ostatniej kompilacji. W wyniku działania tego polecenia na dysku w aktualnym katalogu zostanie utworzony program wykonywalny.

### **Build Project1**

Polecenie kompilacji aktualnego projektu w tzw. trybie *Build*. Kompilowane będą wszystkie moduły niezależnie od tego czy były ostatnio modyfikowane, czy nie. Na dysku w aktualnym katalogu zostanie utworzony plik wykonywalny.

### **Information for (...)**

Podaje informacje na temat ostatnio skompilowanego projektu: liczba linii, rozmiar w bajtach: danych, rozmiar kodu, rozmiar pliku wykonywalnego, itp.

### **Make All Projects**

Kompilacja w trybie Make wszystkich projektów wchodzących w skład grupy projektów.

### **Build All Projects**

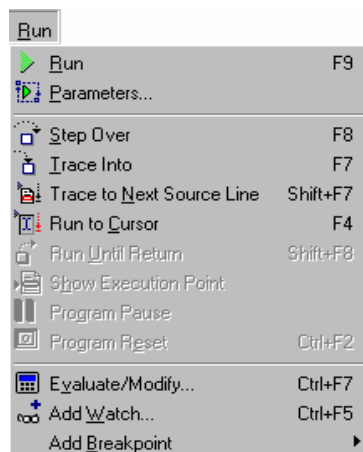
Kompilacja w trybie Build wszystkich projektów wchodzących w skład grupy projektów.

### **Options...**

Polecenie wywołania okna dialogowego Project Options, w którym można ustalić parametry kompilatora i konsolidatora (linkera).

## **Menu Run**

Opcje Menu Run odpowiadają za uruchamianie aplikacji:



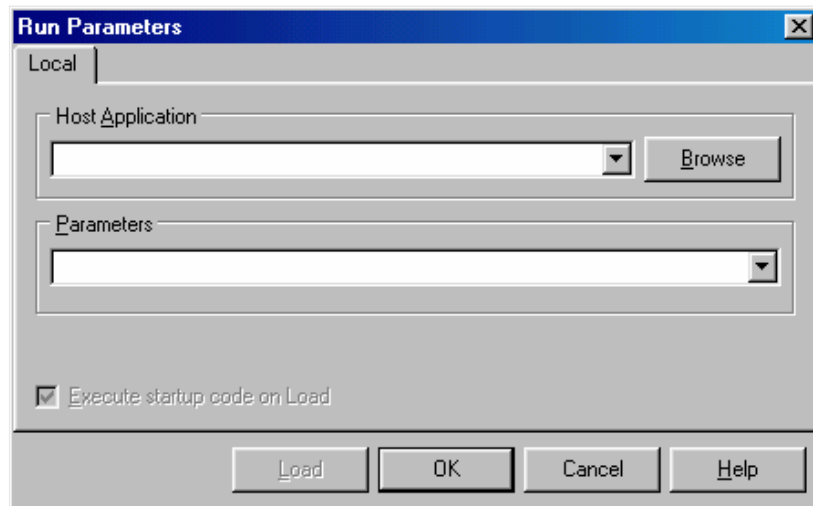
Rys. 17. Opcje Menu Run.

## **Run**

Polecenie umożliwia uruchomienie tworzonej aplikacji. Przed uruchomieniem ma miejsce kompilacja programu.

### **Parameters...**

Polecenie to wyświetla okno dialogowe, w którym można ustalić parametry wywołania aplikacji:



Rys. 18. Okno parametrów uruchomieniowych.

### **Step Over**

Uruchomienie aplikacji w trybie krokowym z możliwością śledzenia jej przebiegu wiersz po wierszu. Wywołania funkcji traktowane będą jako jedna instrukcja bez zaglądania do ich wnętrza.

### **Trace Into**

Uruchomienie aplikacji w trybie krokowym. W momencie wywołania funkcji przenosimy się do jej wnętrza.

### **Trace to Next Source Line**

Uzupełnienie poprzedniej opcji o możliwość zobaczenia kolejnego wiersza kodu, który jest wykonywany.

### **Run to Cursor**

Polecenie wykonania programu do miejsca, w którym ustawiliśmy kursor. Wartość zmiennej można zobaczyć używając polecenia View->Debug Windows->Watches.

### **Run Until Return**

Umożliwia krokowe śledzenie wykonywania programu do momentu uruchomienia aplikacji.

### **Show Execution Point**

Jeżeli w czasie uruchomienia aplikacji w trybie krokowym okno edycji kodu zostało zamknięte, przy pomocy tego polecenia okno zostanie otwarte, zaś kursor znajdować się będzie w wierszu, który jest aktualnie wykonywany.

### **Program Pause**

Tymczasowe wstrzymanie uruchomionego programu.

### **Program Reset**

Polecenie zatrzymania wykonywanego programu z jednoczesnym usunięciem go z pamięci.

### **Evaluate/Modify...**

W czasie działania debuggera istnieje możliwość nie tylko oglądania zmiennych i parametrów, ale również modyfikowania ich wartości. Można też obliczać wyrażenia zawierające te zmienne lub parametry.

### **Add Watch...**

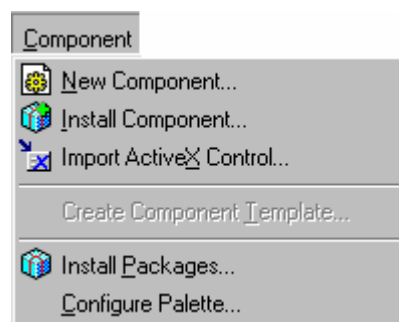
Dodanie nowej zmiennej lub parametru do listy Watches.

### **Add Breakpoint**

Założenie pułapki. Wskazany wiersz kodu zostanie podświetlony.

## **Menu Component**

Opcje dostępne w Menu Component pozwalają na zarządzanie komponentami i pakietami komponentów:



*Rys. 19. Opcje Menu Component.*

### **New Component...**

Wspomaga tworzenie własnego nowego komponentu.

### **Install Component...**

Umożliwia zainstalowanie nowego komponentu.

### **Import ActiveX Control...**

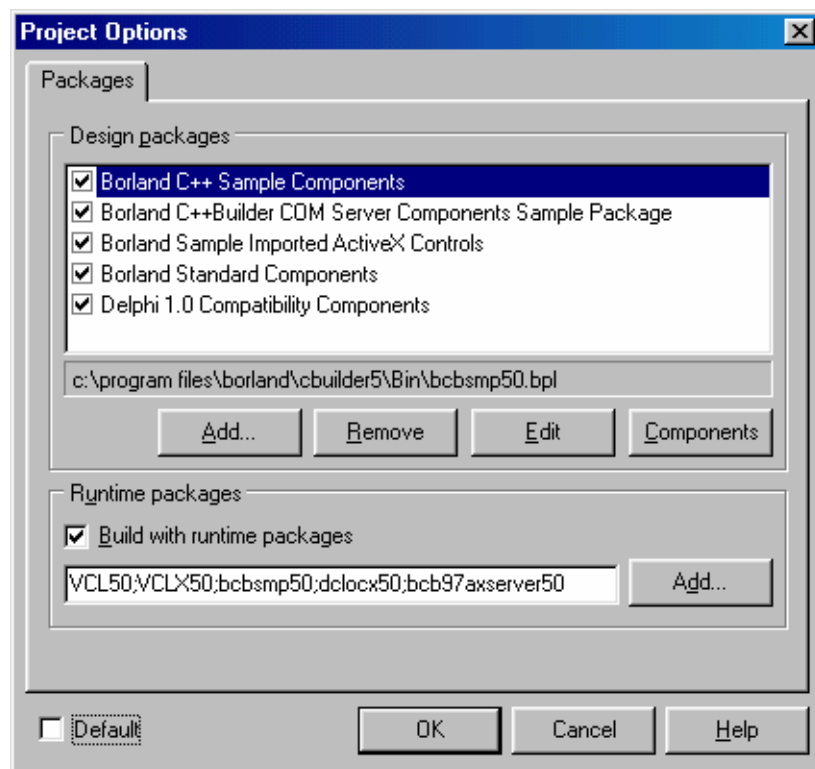
Polecenie dołączenia zarejestrowanego oraz istniejącego obiektu ActiveX do wybranego pakietu VCL.

### **Create Component Template...**

Polecenie tworzy szablon komponentów.

### **Install Packages...**

Opcja umożliwiająca odpowiednie zarządzanie pakietami (ang. *packages*), które stanowią część środowiska i z których zbudowana jest biblioteka VCL. Pakiety takie można dodawać, usuwać i edytować:



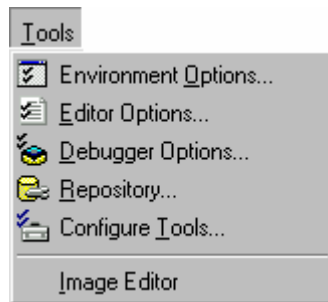
Rys. 20. Okno zarządzania pakietami.

### **Configure Palette...**

Daje możliwość dowolnego skonfigurowania układu palety komponentów poprzez ich dodawanie, usuwanie czy umieszczanie w innych miejscach.

## **Menu Tools**

Menu Tools umożliwia konfigurację różnych opcji programu Borland C++ Builder:



Rys. 21. Opcje Menu Tools.

### **Environment Options...**

Opcja pomocna w określeniu parametrów konfiguracyjnych środowiska (np. ścieżek).

### **Editor Options...**

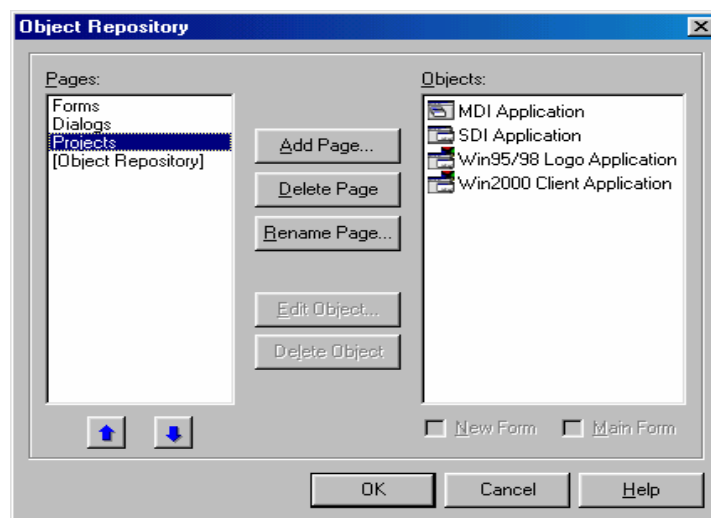
Opcja umożliwiająca na określenie w oknie edycji wielu parametrów konfiguracyjnych, takich jak: rodzaj czcionki, jej kolor, rozmiar okna itp.

### **Debugger Options...**

Ustalenie opcji debuggera.

### **Repository...**

Repozytorium jest centralnym systemem informacji o obiektach tworzących aktualny projekt. Dzięki tej opcji można obiekty takie edytować, dodawać i usuwać:



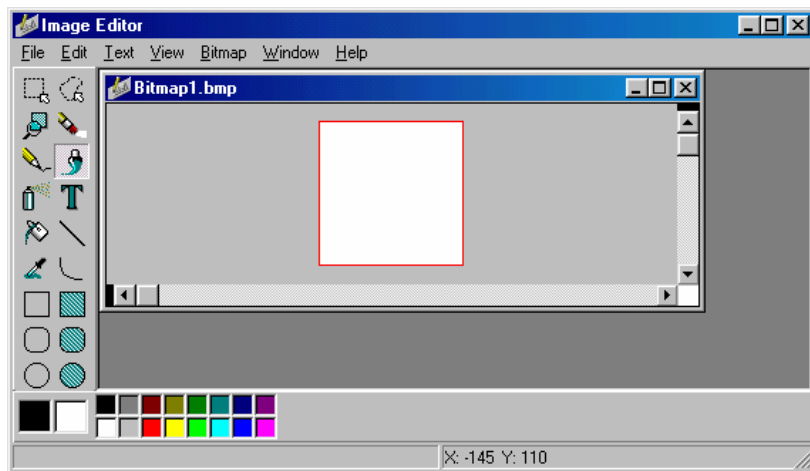
Rys. 22. Okno dialogowe Object Repository.

### **Configure Tools...**

Polecenie umożliwiające odpowiednie skonfigurowanie środowiska.

### **Image Editor**

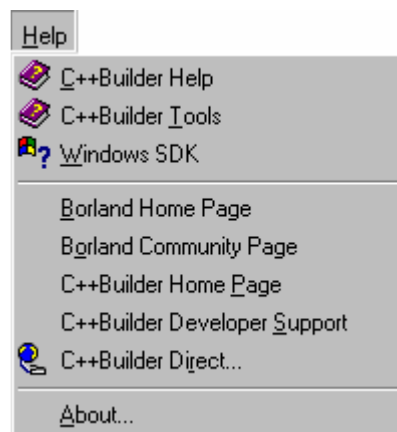
Edytor graficzny służy do samodzielnego projektowania ikon, przycisków, różnego rodzaju rysunków pomocnych w projektowaniu aplikacji. Zasada jego obsługi jest podobna jak w Paint:



*Rys. 23. Wygląd programu narzędziowego Image Editor.*

## **Menu Help**

Opcje Menu Help umożliwiają dostęp pomocy kontekstowej, podręczników C++ Builder i stron internetowych firmy Borland:



*Rys. 24. Opcje Menu Help.*

### **C++Builder Help**

### **C++Builder Tools**

### **Windows SDK**

Zawierają spisy treści oraz pliki pomocy C++ Buildera 5 i WinAPI.

### **Borland Home Page**



## **Borland Community Page**

### **C++Builder Home Page**

### **C++Builder Developer Support**

### **C++Builder Direct...**

Powyższe opcje pozwalają na automatyczne połączenie ze stronami WWW firmy Borland oraz stronami poświęconymi C++ Builder 5.

### **About...**

Zawiera informacje o programie. Można też wyświetlić listę twórców poprzez wpisanie tekstu DEVELOPERS przy wciśniętym lewym klawiszu ALT.

## **Menu Desktop**

Przy pomocy tego zestawu opcji możemy zapisać samodzielnie skonfigurowany pulpit środowiska C++Builder 5:



Rys. 25. Menu Desktop.

### **Pick List**

Zawiera listę nazw, pod którymi zapisano wygląd skonfigurowanych pulpitów.

### **Save current desktop**

Przy pomocy tego okienka dialogowego możliwe jest zapisanie aktualnie skonfigurowanego pulpitu. Analogiczną operacją jest View->Desktops->Save Desktop.

### **Set debug desktop**

Przy pomocy tego polecenia można określić wygląd pulpitu podczas uruchamiania aplikacji np. poleceniem Run->Run. Analogiczną operacją jest View->Desktops->Set Debug Desktop.

Wszystkie dane o dokonanej konfiguracji pulpitu zostaną zapisane na dysku w pliku z rozszerzeniem *.dst*.

## **Pasek narzędzi – Speed Bar**

Pasek narzędzi pozwala na szybszy dostęp do najczęściej używanych poleceń IDE Buildera. Standardowo zawiera on 16 przycisków, które są najczęściej używane przez programistów. Przyciski te pogrupowane są w czterech obszarach:

- Standard
- View
- Debug
- Custom

Oczywiście, dostęp do każdego z nich możliwy jest również z poziomu głównego menu:



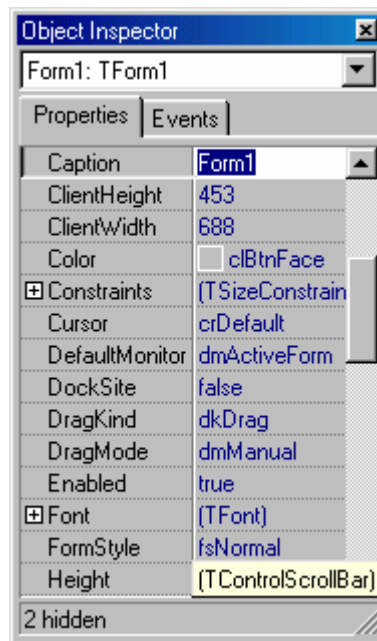
Rys. 26. Pasek narzędzi.

## **Inspektor obiektów – Object Inspector**

Inspektor obiektów jest bardzo ważną częścią IDE. Posługując się nim możemy bardzo szybko ustalać i zmieniać cechy obiektów. Możemy też w wygodny sposób zarządzać i edytować metody stanowiące odpowiedź na określone zdarzenie. Zasadniczą częścią inspektora obiektów są dwie zakładki, czyli karty: karta właściwości, cech (ang. *properties*) oraz karta obsługi zdarzeń (ang. *events*).

### **Karta właściwości – Properties**

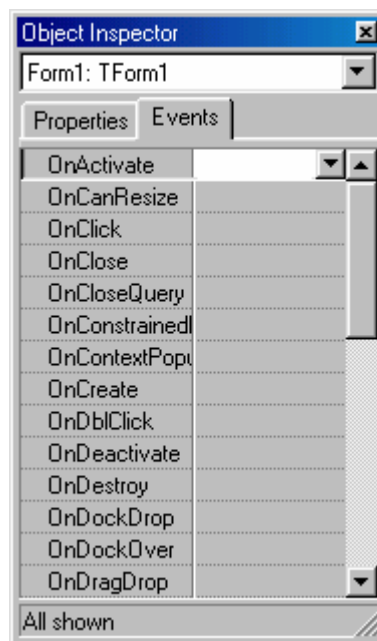
Karta właściwości umożliwia wygodne edytowanie właściwości aktualnie wybranego obiektu. Każdy rodzaj obiektów C++ Builder ma odrębny zestaw własności. Ustawianie ich wartości może odbywać się w różny sposób. Dla niektórych należy wpisać stosowną wartość (np. własność *Caption*), dla innych wybrać jedną z dostępnych opcji (np. własność *Color*). Niektóre właściwości poprzedzone są znaczkiem +. Oznacza to, że zawierają one szereg zagnieżdżonych opcji:



Rys. 27. Zakładka Properties Inspektora Obiektów.

## Karta obsługi zdarzeń – Events

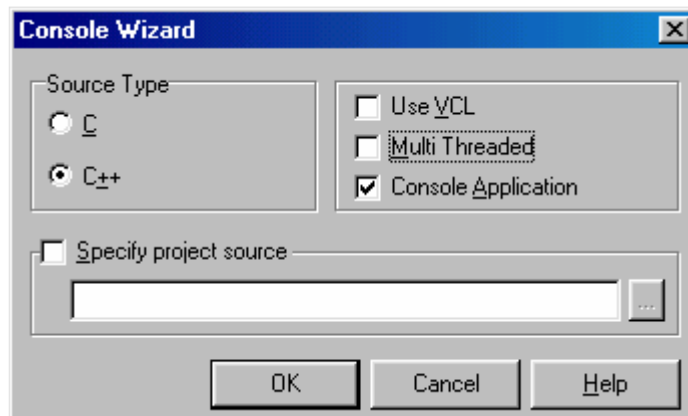
Ta karta stanowi drugą część inspektora obiektów i zawiera listę zdarzeń związanych z danym obiektem. Programowanie w Windows jest związane z obsługą różnego rodzaju zdarzeń (kliknięć myszą, naciśnięć klawiszy, wyboru opcji, itd.). Dla przykładu, jeśli program ma wykonywać jakąś operację w odpowiedzi na kliknięcie na obszar jakiegoś komponentu, to w odpowiedni sposób należy obsłużyć zdarzenie `OnClick`. Tego rodzaju technika programowania nazywana jest programowaniem obiektowo – zdarzeniowym:



Rys. 28. Zakładka Events Inspektora Obiektów.

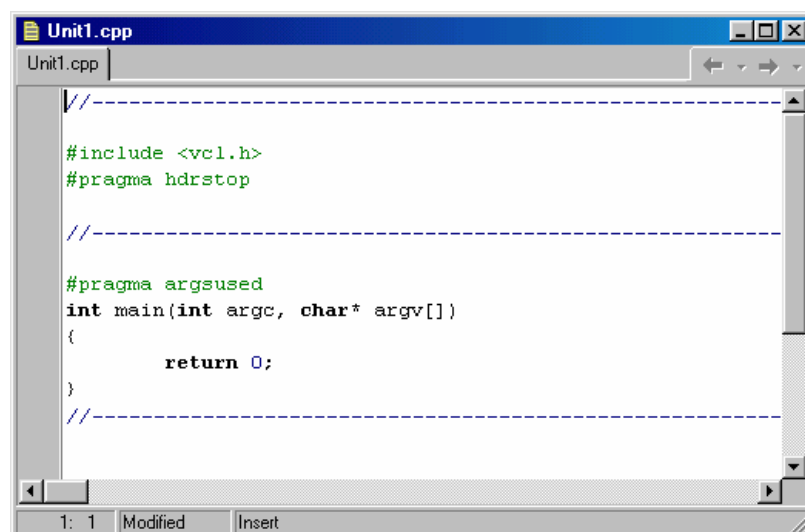
## 4. Pierwszy program konsolowy

Środowisko Borland C++ Builder umożliwia pisanie programów konsoli z wykorzystaniem tradycyjnej postaci języka C++. W celu utworzenie programu konsoli wybieramy z menu opcję File->New->Console Wizard. Na ekranie powinno pojawić się okno wyglądające następująco:



Rys. 29. Okno kreatora aplikacji konsolowej.

W opcji Source Type zaznaczone powinno być C++, zaś w drugim panelu odznaczone Use VCL oraz Multi Threaded, a zaznaczone Console Application. Zaznaczenie tej ostatniej opcji spowoduje, że nasz program będzie traktował główny formularz tak, jakby był normalnym okienkiem tekstowym DOS. Po zatwierdzeniu przyciskiem OK uzyskamy szkielet kodu przyszłego programu:



Rys. 30. Kod modułu Unit1.cpp.

Uzupełnijmy kod programu tak aby wyświetlał na ekranie komunikat:

```
Program konsoli napisany w Borand C++ Builder  
Naciśnij klawisz...
```

Pełny kod tworzonego programu znajduje się poniżej:

```
#include <iostream.h>  
#include <conio.h>  
#pragma hdrstop  
  
int main()  
{  
    cout << "Program konsoli napisany w Borland C++ Builder";  
    cout << endl << "Naciśnij klawisz...";  
    getch();  
    return 0;  
}  
//-----
```

W celu zapisania modułu należy wybrać opcję File->Save As..., zaś w celu zapisania projektu – opcję File->Save Project As... Aby uruchomić program wystarczy nacisnąć klawisz F9 lub wybrać z menu opcję Run->Run.

## **Ćwiczenia**

1. Napisz program konsoli, który prosi o podanie dwóch liczb, a następnie wyświetla ich sumę.

## 5. Pierwszy program graficzny

Aby utworzyć pierwszą aplikację graficzną wystarczy uruchomić Borland C++ Builder lub wybrać File->New Application. W ten sposób uzyskujemy pusty formularz posiadający standardowe właściwości okien Windows. Jeśli chcemy zmienić kolor tła formularza wystarczy ustawić własność Color na clBlack. W ten sposób stworzyliśmy program nie wpisując ani jednej linii kodu. Środowiska IDE do programowania wizualnego wspomagają tworzenie programów graficznych minimalizując potrzebę wpisywania kodu. Jednakże żadna sensowna aplikacja nie obędzie się bez wpisania przez programistę określonej ilości linii kodu. W kolejnym przykładzie dodamy do aplikacji przycisk (ang. *Button*) z zakładki Standard. Wystarczy kliknąć na wybranym komponencie, a następnie umieścić go we wskazanym miejscu na formularzu (w każdej chwili możemy zmienić jego położenie i rozmiar). Następnie zmieniamy własność Caption na &Czerwony (znak & przed wybraną literą spowoduje, że na przycisku będzie ona napisana z podkreśleniem i jednocześnie będzie stanowić klawisz skrótów wyzwalający przycisk). Chcemy aby aplikacja po wciśnięciu przycisku powodowała zmianę tła formularza na czerwony. W tym celu obsłużymy zdarzenie OnClick. Obsługę tego zdarzenia możemy zdefiniować w dwojaki sposób – poprzez dwukrotne kliknięcie na przycisku lub wybranie w Inspektorze Obiektów zakładki Events i dwukrotne kliknięcie na puste pole obok napisu OnClick. Po wykonaniu jednej z wymienionych opcji wygenerowana zostanie procedura obsługi zdarzenia OnClick dla wybranego przycisku. Wystarczy uzupełnić ją poleceniem powodującym zmianę koloru formularza na czerwony:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    Color=clRed;
}
```

Jak widać z poziomu kodu ustawiamy własność Color. Prawie wszystkie własności, które możemy zmieniać z poziomu Inspektora Obiektów możemy też zmieniać programowo.

### Ćwiczenia

1. Napisać program, w którym zostaną umieszczone trzy przyciski z napisami Czerwony, Zielony, Niebieski i spowodować aby ich wciśnięcie powodowało zmianę tła formularza odpowiednio na kolor czerwony, zielony i niebieski.

*Wskazówka:* Posłuż się własnością Color i zdefiniowanymi stałymi clRed, clGreen, clBlue oznaczającymi odpowiednio kolor czerwony, zielony i niebieski.

2. Dodaj do programu kolejny przycisk z napisem Zamknij powodujący zamknięcie aplikacji.

*Wskazówka:* Posłuż się poleceniem Application->Terminate() lub Close().

3. Dodaj do programu następny przycisk z napisem Nagłówek powodujący zmianę tytułu okna na: Przyciski z kolorami.

*Wskazówka:* Wykorzystaj własność Caption.

## 6. Elementy składowe aplikacji w Borland C++ Builder

Wszystkie składniki aplikacji budowanej w Borland C++ Builder przechowywane są w plikach. Poniżej zostały one pokrótce omówione:

- *Projekt01.exe* – binarny plik wykonywalny aplikacji.
- *Projekt01.bpr* – główny plik projektu, zawiera informacje o wszystkich elementach projektu: modułach, bibliotekach itp. Jest to plik tekstowy zapisany w formacie XML.
- *Projekt01.tds* – table debug symbols, czyli tablica symboli debugujących. Jest to plik binarny i podlega automatycznemu utworzeniu w trakcie uruchamiania aplikacji z poziomu Borland C++ Builder.
- *Projekt01.obj* – plik binarny będący efektem kompilacji aplikacji.
- *Projekt01.res* – jest binarnym plikiem zasobów (ang. resources). Zawiera m. in. ikonę.
- *Projekt01.cpp* – główny plik aplikacji. Zawiera definicję funkcji WinMain():

```
#include <vcl.h>
#pragma hdrstop
USERES("Projekt01.res");
USEFORM("Unit01.cpp", Form1);
//-----
----
WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)
{
    try
    {
        Application->Initialize();
        Application->CreateForm(__classid(TForm1), &Form1);
        Application->Run();
    }
    catch (Exception &exception)
    {
        Application->ShowException(&exception);
    }
    return 0;
}
```

Programy pisane w Borland C++Builderze i posługujące się klasą formularza nie zawierają funkcji main(). Wszystkie pisane przez nas aplikacje rozpoczynają działanie od wywołania innej funkcji, mianowicie WinMain(), wywoływanej zgodnie z zasadami WINAPI, co jest wyraźnie zaznaczone w jej definicji. Otrzymuje ona wartość czterech parametrów. Pierwsze dwa, typu HINSTANCE (w wolnym tłumaczeniu określane jako uchwyty przypadku) są niezbędne z prostego powodu, mianowicie Windows w obecnym kształcie jest systemem



wielozadaniowym, w związku z tym w danej chwili może działać jednocześnie wiele egzemplarzy tego samego programu. Parametry przypisane typom HINSTANCE określają aktualnie działające egzemplarze programu. Parametr typu LPSTR jest wskaźnikiem do łańcucha znaków zawierającego argumenty wiersza poleceń, które są określane w trakcie uruchamiania aplikacji. Ostatni parametr typu całkowitego int określa sposób wyświetlania okna formularza po rozpoczęciu działania aplikacji. Proces inicjacji — metoda Initialize(), tworzenia formularza — metoda CreateForm() oraz uruchamiania aplikacji — metoda Run() rozgrywa się pomiędzy klauzulami try...catch (w wolnym tłumaczeniu: próbuj...przechwyć, złap). Jeżeli proces ten nie powiedzie się, na ekranie ujrzymy stosowny komunikat w postaci wygenerowanego przez system tzw. wyjątku (ang. exception), wyświetlanego przy pomocy funkcji ShowException().

- *Unit01.cpp* – zawiera kod źródłowy modułu.
- *Unit01.h* – jest plikiem nagłówkowym modułu i zawiera deklarację klasy formularza.
- *Unit01.dfm* – jest plikiem tekstowym zawierającym definicję obiektu formularza oraz definicje wszystkich używanych komponentów.

## Klasa TForm1

Formularz jest pierwszym obiektem, z którym spotykamy się rozpoczynając pisanie aplikacji. Zawartość przykładowego pliku nagłówkowego moduły formularza została przedstawiona poniżej:

```
#ifndef Unit01H
#define Unit01H
//-----
----
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
//-----
----
class TForm1 : public TForm
{
__published: // IDE-managed Components
private: // User declarations
public: // User declarations
__fastcall TForm1(TComponent* Owner);
};
//-----
----
extern PACKAGE TForm1 *Form1;
//-----
----
#endif
```

Tworzona jest klasa TForm1 dziedzicząca w sposób publiczny po klasie TForm, będącej bazową klasą formularza. Definicja klasy składa się z kilku części. W sekcji `__published` umieszczane będą deklaracje funkcji, czyli deklaracje metod związanych z komponentami pochodzącymi z biblioteki VCL. Sekcja `private` przeznaczona jest dla zmiennych (zwanych tutaj polami) oraz metod widzianych tylko wewnątrz klasy. W sekcji `public` deklarować można pola i metody mogące być udostępniane innym.

## **Konstruktor TForm1()**

Zanim zaczniemy na serio korzystać z obiektu naszego formularza musi on zostać odpowiednio zainicjowany. Dokonuje się to poprzez specjalną funkcję składową, noszącą taką samą nazwę jak klasa, do której należy. Prototyp takiej funkcji (nazywanej konstruktorem) z parametrami wygląda następująco:

```
__fastcall TForm1(TComponent* Owner);
```

Ponieważ konstruktor nie zwraca żadnej wartości, nie określa się jego typu (przez domniemanie jest on typu nieokreślonego, czyli `void`). Konwencja `__fastcall` (szybkie wywołanie) zapewnia, że parametry konstruktora zostaną przekazane poprzez rejestry procesora. Dodatkowo zapis konstruktora z parametrem `Owner` informuje, że właścicielem (ang. `owner`) wszystkich komponentów jest `TComponent` mówi nam, że `TComponent` jest wspólnym przodkiem dla wszystkich komponentów z biblioteki VCL włącznie ze stworzoną klasą `Tform1`. Klasa `TComponent`, wprowadzając wiele metod i właściwości, umożliwia m.in. obsługę komponentów z poziomu inspektora obiektów. Pełny tekst konstruktora klasy `TForm1` zostanie automatycznie umieszczony w module `Unit01.cpp`, tam też zostanie zainicjowany.

## **Formularz jako zmienna obiektowa**

Projekt aplikacji składa się nie tylko z formularza, ale również z modułów i innych zasobów. Wszystkie części składowe aplikacji przechowywane są w odpowiednich plikach, w większości wypadków tworzonych automatycznie przez środowisko programistyczne. Ponieważ C++Builder 5 (podobnie jak C i C++) pozwala na konsolidację oddzielnie skompilowanych modułów dużego programu, musi zatem istnieć jakiś sposób na poinformowanie wszystkich plików wchodzących w skład projektu o występowaniu zmiennych globalnych (widocznych w całej aplikacji), niezbędnych w danym programie. Najlepszym sposobem by to osiągnąć, jest zadeklarowanie zmiennych globalnych tylko w

jednym pliku i wprowadzenie deklaracji przy pomocy specyfikatora extern PACKAGE (ang. zewnętrzny pakiet) w innych plikach.

Formularz jest obiektem (zmienną obiektową), której deklaracja zostanie umieszczona w głównym module formularza Unit01.cpp:

```
#include <vcl.h>
#pragma hdrstop
#include "Unit01.h"
...

TForm1 *Form1;
...
```

Widzimy więc, że nazwa klasy stała się nowym specyfikatorem typu danych.

## **7. Biblioteka Visual Component Library**


Najważniejszym elementem środowisk programistycznych dla Windows, takich jak Delphi czy Builder, jest biblioteka wizualnych komponentów. Korzystając z Borland C++ Buildera możemy posługiwać się dziewiętnastoma paletami takich komponentów:

- Standard components
- Additional components
- Win32 components
- System components
- Data Access components
- Data Controls components
- ADO components
- InterBase components
- MIDAS components
- InternetExpress components
- Internet components
- FastNet components
- Decision Cube components
- QReport components
- Dialogs components
- Win 3.1 components
- Samples components
- ActiveX components
- Servers components

W wersji Standard mamy do dyspozycji dziesięć kart zawierających najczęściej używane komponenty. Nie jest oczywiście możliwe, aby w opracowaniu o niewielkich rozmiarach szczegółowo opisać każdy komponent z uwzględnieniem jego cech, metod i zdarzeń, nawet jeżeli pracujemy w standardowej wersji C++ Buildera. Ważnym uzupełnieniem są pliki pomocy Buildera. Sprowadzając jakiś komponent do obszaru formularza zawsze możemy posłużyć się klawiszem F1, aby otrzymać wyczerpującą informację na temat klasy, do jakiej należy wybrany komponent, jego właściwości, itp. Poruszając się po niezwykle bogatych w treści plikach pomocy znajdziemy tam również bardzo wiele użytecznych przykładów praktycznego posługiwania się określonymi obiektami. Obecnie zapoznamy się z kilkoma najczęściej używanymi kartami.






## Karta Standard

Korzystając z zasobów tej karty mamy do dyspozycji wszystkie najczęściej wykorzystywane komponenty reprezentujące sobą wszystkie podstawowe elementy sterujące Windows.

| Ikona   | Typ          | Znaczenie   |
|---|--------------|---|
|    | TFrames      | „Ramki” nie są w ścisłym tego słowa znaczeniu typowymi komponentami, tzn. nie można ich bezpośrednio w prosty sposób umieszczać na formularzu. Jeżeli zdecydujemy się na włączenie ramki w skład naszego projektu, najpierw należy ją stworzyć, najlepiej poleceniem menu <b>File-&gt;New Frame</b> . Właściwości ramki do złudzenia przypominają właściwości formularza. |
|    | TMainMenu    | Komponent pomocny w procesie projektowania i tworzenia głównego menu aplikacji; jest niewidoczny w trakcie działania aplikacji.   |
|    | TPopupMenu   | Ten komponent generuje tzw. menu kontekstowe, którym można się posługiwać po naciśnięciu prawego klawisza myszki. Należy do grupy komponentów niewidocznych.  |
|    | TLabel       | W polu tej etykiety możemy wyświetlać tekst.  |
|  | TEdit        | Komponent edycyjny, nazywany polem edycji, w którym możemy wyświetlić jeden wiersz tekstu.  |
|  | TMemo        | Ten komponent pozwala na edycję większej porcji tekstu.   |
|  | TButton      | Przycisk.   |
|  | TCheckBox    | Komponent reprezentujący pole wyboru. Posiada właściwość <b>Checked</b> , która może reprezentować dwa stany: włączony — <b>TRUE</b> lub wyłączony — <b>FALSE</b> .   |
|  | TRadioButton | Umożliwia dokonanie wyboru tylko jednej spośród wielu opcji. Komponent ten powinien występować w grupie podobnych komponentów reprezentujących pewne opcje aplikacji, z których możemy wybrać tylko jedną.  |
|  | TListBox     | Komponent pomocny w tworzeniu listy elementów, które następnie możemy dowolnie zaznaczać i wybierać.  |
|  | TComboBox    | Ten komponent także wykorzystywany jest do tworzenia listy elementów, jednak posiadając pewne cechy <b>TEdit</b> umożliwia nam również wpisywanie tekstu.   |
|  | TScrollBar   | Ten komponent reprezentuje pasek przewijania (choć nie jest typowym suwakiem). Dodajemy go z reguły do innych, które nie posiadają w sobie opcji przewijania, np. do tekstu.  |
|  | TGroupBox    | W obszarze tego komponentu możemy pogrupować inne elementy, np. <b>TRadioButton</b> czy <b>TCheckBox</b> . Posiada ciekawą własność w postaci linii tytułowej, w której możemy wpisać np. nazwę danego obszaru formularza.  |
|  | TRadioGroup  | Komponent grupujący elementy typu <b>TRadioButton</b> . Również posiada własną linię tytułową.  |
|  | TPanel       | Reprezentuje panel, na którym możemy umieszczać inne komponenty. Posiadając rozbudowane własności „estetyczne” doskonale nadaje się do roli paska narzędzi lub linii statusu.   |
|  | TActionList  | Komponent ten potocznie nazywany jest „organizatorem pisania oraz działania aplikacji”. W wygodny sposób udostępnia nam zestawy akcji, pozwalające na wywoływanie funkcji obsługi zdarzeń w określonej sekwencji. Umożliwia też (wspólnie z <b>TImageList</b> znajdującym się na karcie Win32) bardzo estetyczne zaprojektowanie menu aplikacji.                          |

## Karta Additional

Karta Additional jest rozszerzeniem karty Standard. Zawiera szereg komponentów, które okazują się bardzo przydatne w projektowaniu aplikacji.

| Ikona   | Typ                | Znaczenie   |
|---|--------------------|---|
|    | TBitBtn            | Przycisk, na którym można umieszczać rysunek.   |
|    | TSpeedButton       | Przycisk umieszczany zwykle na pasku zadań. Na nim również możemy umieszczać rysunki.   |
|    | TMaskEdit          | Komponent służący do maskowania i filtrowania danych wpisywanych zwykle z klawiatury.   |
|    | TStringGrid        | Element, który pozwala na umieszczenie na formularzu typowego arkusza składającego się z komórek edycyjnych rozmieszczonych w wierszach i kolumnach.  |
|    | TDrawGrid          | Element, który umożliwia graficzne przedstawienie danych nie będących tekstem.  |
|    | TImage             | Komponent graficzny. Umożliwia wyświetlenie na formularzu np. mapy bitowej.   |
|    | TShape             | Ten element umieszcza na formularzu wybraną figurę geometryczną. Posiada cechę <code>Shape</code> , przy pomocy której możemy wybrać rodzaj figury.   |
|  | TBevel             | Składnik, który umieszcza na formularzu obszar prostokątny, posiadający cechy trójwymiarowości. Dzięki cechom <code>Shape</code> i <code>Style</code> możemy określić sposób jego wyświetlania. |
|  | TScrollBar         | Komponent zawierający paski przewijania. Może pełnić rolę przewijanego okienka.   |
|  | TCheckListBox      | Element stanowiący połączenie listy i pola wyboru. Posiada cechę <code>Items</code> umożliwiającą edytowanie tekstu.  |
|  | TSplitter          | Ten komponent dzieli formularz lub okno na kilka części, których obszar możemy zmieniać. Dopiero użycie co najmniej dwóch takich komponentów może dać pożądany efekt.                           |
|  | TStaticText        | Składnik, który jest odpowiednikiem <code>TLabel</code> , uzupełniony jednak o szereg właściwości, umożliwia bardziej estetyczne wyświetlenie tekstu.   |
|  | TControlBar        | Komponent, który umożliwia eleganckie i wygodne rozmieszczenie różnych komponentów na pasku zadań.  |
|  | TApplicationEvents | Komponent umożliwiający przechwytywanie zdarzeń generowanych przez aplikację, w tym również wyjątków.   |
|  | TChart             | Ten składnik umożliwia graficzną wizualizację danych w postaci różnego rodzaju wykresów.  |















## Karta Win32

Karta zawiera wszystkie elementy sterujące reprezentowane w aplikacjach Windows.

Tabela 6.3. Komponenty karty Win32



| Ikona | Typ | Znaczenie |
|-------|-----|-----------|
|-------|-----|-----------|







## Programowanie wizualne – Borland C++ Builder

|   |                        |  |
|---|------------------------|--|
|    | <b>TTabControl</b>     | Korzystając z tego komponentu mamy możliwość tworzenia zakładek.   |
|    | <b>TPageControl</b>    | Komponent składający się z większej ilości kart. Aby stworzyć nową kartę w najprostszym przypadku należy nacisnąć prawy klawisz myszki i wybrać opcję <a href="#">New Page</a> .                           |
|    | <b>TImageList</b>      | Ten składnik umożliwia utworzenie listy elementów graficznych. Każdemu z obrazków automatycznie jest przypisywany odpowiedni indeks. Komponent niewidzialny.   |
|    | <b>TTrackBar</b>       | Suwak. Posiada cechę <b>Position</b> , dzięki której można regulować i odczytywać aktualną pozycję wskaźnika przesuwania.  |
|    | <b>TProgressBar</b>    | Komponent będący wskaźnikiem postępu. Również posiada cechę <b>Position</b> , dzięki której możemy śledzić postęp wykonywanych operacji.   |
|    | <b>TUpDown</b>         | Komponent umożliwiający zwiększanie bądź zmniejszanie jakiejś wartości. Z reguły nie występuje samodzielnie. Wartości należy wyświetlać w komponentach edycyjnych. Również posiada cechę <b>Position</b> . |
|    | <b>THotKey</b>         | Element umożliwiający utworzenie klawisza szybkiego dostępu.   |
|    | <b>TAnimate</b>        | Komponent, który umożliwia wyświetlanie sekwencji obrazów.   |
|    | <b>TDateTimePicker</b> | Komponent będący w istocie pewnego rodzaju kalendarzem. Umożliwia odczytanie i wybranie odpowiedniej daty. Posiada rozwijany obszar podobny do <b>TListBox</b> .   |
|    | <b>TMonthCalendar</b>  | Komponent bardzo podobny do poprzedniego, z tą różnicą, że wyświetla od razu datę bieżącego miesiąca.  |
|   | <b>TTreeView</b>       | Składnik powodujący hierarchiczne wyświetlanie elementów.  |
|  | <b>TListView</b>       | Lista widoków wyświetla pozycje składające się z ikon i etykiet.   |
|  | <b>THeaderControl</b>  | Komponent tworzący listę nagłówkową mogącą składać się z wielu sekcji.   |
|  | <b>TStatusBar</b>      | Linia statusu formularza. Aby umieścić odpowiedni tekst w linii statusu formularza, wystarczy nacisnąć prawy klawisz myszki i dostać się do <a href="#">Panels Editor...</a>                               |
|  | <b>TToolBar</b>        | Komponent, który tworzy paski narzędzi.  |
|  | <b>TCoolBar</b>        | Komponent będący pewną odmianą panelu, z tą różnicą, że pozwala na zmianę jego rozmiaru.   |
|  | <b>TPageScroller</b>   | Ten składnik może zawierać inne obiekty z możliwością przewijania ich zarówno w pionie jak i poziomie.   |

## Karta System











Karta System zawiera szereg komponentów wykorzystywanych w różnych operacjach na poziomie systemu Windows.

| <b>Ikona</b>  | <b>Typ</b>       | <b>Znaczenie</b>  |
|---|------------------|---|
|  | <b>TTimer</b>    | Jest komponentem niewidzialnym. Służy do generowania zdarzeń w równych odstępach czasu. |
|  | <b>TPaintBox</b> | Komponent wykorzystywany do wykonywania różnych operacji graficznych.                   |

|   |                       |  |
|---|-----------------------|--|
|  | <b>TMediaPlayer</b>   | Komponent, który umożliwia wykorzystywanie w aplikacji technik multimedialnych.                      |
|  | <b>TOleContainer</b>  | Jest komponentem niewidocznym. Służy do generowania na formularzu obszaru klienta OLE.               |
|  | <b>TDDEClientConv</b> | Komponent niewidzialny. Umożliwia połączenie z serwerem DDE.   |
|  | <b>TDDEClientItem</b> | Komponent niewidzialny. Określa dane wysyłane przez klienta podczas konwersacji DDE.                 |
|  | <b>TDDEServerConv</b> | Niewidzialny komponent umożliwiający nawiązanie dialogu z klientem DDE.                              |
|  | <b>TDDEServerItem</b> | Komponent niewidzialny. Umożliwia określenie danych wysyłanych do klienta w trakcie konwersacji DDE. |

## Karta Dialogs

Komponenty Karty Dialogs reprezentują standardowe okna dialogowe Windows. Są to np. okna do zapisu pliku, odczytu, drukowania, wyboru rodzaju czcionki czy palety kolorów. Wszystkie są komponentami niewidzialnymi.












| <b>Ikona</b>  | <b>Typ</b>                 | <b>Znaczenie</b>  |
|---|----------------------------|---|
|  | <b>TOpenDialog</b>         | Komponent tworzący okienko dialogowe służące do wyboru i otwarcia pliku.          |
|  | <b>TSaveDialog</b>         | Komponent tworzący okienko dialogowe służące do zapisu danych do pliku.           |
|  | <b>TOpenPictureDialog</b>  | Składnik umożliwiający dokonanie wyboru plików, w tym również plików graficznych. |
|  | <b>TSavePictureDialog</b>  | Komponent tworzący okienko dialogowe służące do zapisu pliku graficznego.         |
|  | <b>TFontDialog</b>         | Komponent, który umożliwia dokonanie wyboru czcionki.                             |
|  | <b>TColorDialog</b>        | Okienko dialogowe służące do wyboru palety kolorów.                               |
|  | <b>TPrintDialog</b>        | Okienko dialogowe służące do drukowania.  |
|  | <b>TPrinterSetupDialog</b> | Komponent określający ustawienia drukarki.  |
|  | <b>TFindDialog</b>         | Komponent służący do podglądu i wyszukiwania tekstu.                              |
|  | <b>TReplaceDialog</b>      | Okienko, które umożliwia wyszukanie fragmentu tekstu i zastąpienie go innym.      |

## Karta Win 3.1

Karta Win 3.1 udostępnia listę komponentów stosowanych w starszych, 16-bitowych wersjach C++ Buildera. Nie jest zalecane używanie komponentów




posiadających swoje odpowiedniki np. w obecnych kartach Win32 czy Data Controls.









| Ikona   | Typ               | Znaczenie   |
|---|-------------------|---|
|    | TTabSet           | Odpowiada komponentowi TTabControl z karty Win32.   |
|    | TOutline          | Odpowiada komponentowi TTreeView z karty Win32.   |
|    | TTabbedNoteBook   | Odpowiednik TPageControl z karty Win32.   |
|    | TNoteBook         | Odpowiednik TPageControl.   |
|    | THeader           | Odpowiada komponentowi THeaderControl z karty Win32.  |
|    | TFileListBox      | Komponent dający możliwość wyświetlenia listy plików wskazanego katalogu.   |
|    | TDirectoryListBox | Element udostępniający listę katalogów wybranego napędu.  |
|    | TDriveComboBox    | Komponent pozwalający dokonać wyboru napędu (stacji dysków).  |
|   | TFilterComboBox   | Komponent, który udostępnia listę plików wyświetlanych z zastosowaniem odpowiedniego filtru. Celowi temu służy właściwość Mask. |
|  | DBLookupList      | Odpowiada komponentowi TDBLookupListBox z karty Data Controls dostępnej w wersji Enterprise C++Buildera 5.                      |
|  | DBLookupCombo     | Odpowiada komponentowi TDBLookupComboBox z karty Data Controls dostępnej w wersji Enterprise C++Buildera 5.                     |

Komponenty karty Win 3.1 mimo, iż pochodzą ze starszych wersji C++Buildera, są w dalszym ciągu często używane. Chodzi głównie o komponenty ułatwiające bardzo szybko wczytanie wybranego pliku. Obiekty obsługujące te zdarzenia mają jeszcze jedną poważną zaletę, mianowicie wczytywany plik można natychmiast poddać edycji.

## Karta Samples





Karta Samples zawiera 9 przykładowych komponentów. Ich kody źródłowe znajdują się w katalogu instalacyjnym Buildera \EXAMPLES\CONTROLS\SOURCE. W momencie włączenia tych komponentów do formularza, ich pliki nagłówkowe zostaną dołączone dyrektywą #pragma link, która informuje konsolidator o potrzebie dołączenia danego zbioru do pliku wykonawczego programu.

| Ikona   | Typ  | Znaczenie   |
|---|------|---|
|  | TPie | Element służący do przedstawiania okręgu lub wycinka okręgu. Właściwość <code>Angles</code> uruchamia Pie Angles Editor. Kod źródłowy komponentu można znaleźć w plikach <code>piereg.*</code> oraz <code>pies.*</code> . |

|   |                    |   |
|---|--------------------|---|
|  | TTrayIcon          | Komponent, który umożliwia m.in. wykonanie zamiany ikon, w tym ich prostej animacji. Kod źródłowy komponentu można znaleźć w plikach <i>Trayicon.*</i> .  |
|  | TPerformanceGraph  | Element służący do przedstawienia grafiki. Kod źródłowy komponentu znajduje się w plikach <i>PERFGRAP.*</i> .   |
|  | TCSpinButton       | Komponent umożliwiający płynne zmniejszanie i zwiększanie zawartości liczbowej wybranego pola edycji. Jego kod źródłowy znajduje się w plikach <i>CSPIN.*</i> .   |
|  | TCSpinEdit         | Element stanowiący połączenie TCSpinButton oraz TEdit. Kod źródłowy można znaleźć w plikach <i>CSPIN.*</i> .  |
|  | TCColorGrid        | Komponent umożliwiający dokonanie wyboru koloru. Jego kod źródłowy znajduje się w plikach <i>CGRID.*</i> .  |
|  | TCGauge            | Komponent przedstawiający wskaźnik postępu. Dzięki właściwości <i>Kind</i> można go przedstawić w postaci paska, liczby, koła lub wycinka koła. Jego kod źródłowy znajduje się w plikach <i>CGAUGES.*</i> . |
|  | TCDirectoryOutLine | Wyświetla drzewo katalogów znajdujących się na dysku. Kod źródłowy komponentu znajduje się w plikach <i>cdiroutl.*</i> .  |
|  | TCCalendar         | Komponent wyświetlający aktualną datę w postaci uproszczonego kalendarza. Jego kod źródłowy znajduje się w pliku <i>CCALENDR.*</i> .  |



## Karta ActiveX






Komponenty karty ActiveX nie wchodzi w skład biblioteki VCL. Są to przykładowe obiekty ActiveX, zaprojektowane w ten sposób, by można było natychmiast skorzystać z ich usług.

| Ikona   | Typ      | Znaczenie   |
|---|----------|---|
|  | TChartfx | Obiekt ActiveX służący do tworzenia wykresów.   |
|  | TVSSpell | Visual Speller Control Properties. Komponent pełniący rolę tzw. <i>spell-chackera</i> . |
|  | TF1Book  | Obiekt posiadający cechy arkusza kalkulacyjnego.  |
|  | TVtChart | Komponent służący to tworzenia wykresów.  |

## Karta Internet

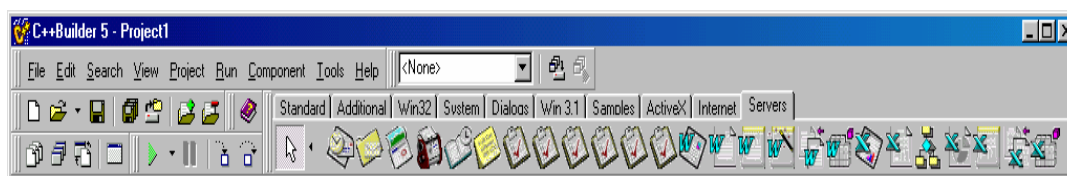
Wykorzystując komponenty karty Internet można w aplikacjach umieszczać opcje pozwalające na korzystanie z sieci Internet oraz protokołu TCP/IP.

| Ikona   | Typ           | Znaczenie   |
|---|---------------|---|
|  | TClientSocket | Komponent ułatwiający połączenie z innym komputerem w sieci.  |
|  | TServerSocket | Komponent odpowiadający na żądania innych komputerów w sieci. |



|   |                              |   |
|---|------------------------------|---|
|  | <b>TCppWebBrowser</b>        | Komponent wyświetlający stronę HTML w postaci Web. Warunkiem jest posiadanie przeglądarki Internet Explorer wersji 4 lub wyższej. |
|  | <b>TWebDispatcher</b>        | Komponent, przy pomocy którego następuje konwersja zwykłego modułu danych na postać Web.  |
|  | <b>TPageProducer</b>         | Komponent konwertujący szablon HTML na kod, który może być następnie przeglądany.   |
|  | <b>TQueryTableProducer</b>   | Komponent tworzący tablice HTML na podstawie rekordów obiektu typu TQuery.  |
|  | <b>TDataSetTableProducer</b> | Komponent tworzący tablice HTML na podstawie rekordów obiektu typu TDataSet.  |

## Karta Servers

Karta Servers zawiera 30 komponentów będących swego rodzaju wizualizacją aktualnie dostępnych serwerów COM dokonaną na potrzeby biblioteki VCL. Wszystkie wywodzą się z obiektu TOleServer. Przy ich pomocy możemy automatycznie połączyć się z wybranym serwerem COM.



Dokładne omówienie wszystkich komponentów karty Servers wraz z ich właściwościami i metodami, z których korzystają, a tym samym budowy serwerów COM, znacznie wykracza poza ramy naszych ćwiczeń. Niemniej jednak możemy chociażby jakościowo zapoznać się z podstawowymi własnościami wybranych obiektów. Prawdopodobnie nie ma wśród nas nikogo, kto by nie miał do czynienia z narzędziami pakietu Office. Do najbardziej podstawowych aplikacji tego pakietu należy oczywiście zaliczyć Worda, Excela oraz Power Pointa. Spróbujmy zatem, korzystając z bardzo prostych funkcji połączyć się z wymienionymi aplikacjami.

| Ikona   | Typ                           | Znaczenie   |
|---|-------------------------------|---|
|  | <b>TPowerPointApplication</b> | Umożliwia połączenie z aplikacjami Power Point. Komponent niewidzialny. Jego kod źródłowy znajduje się w plikach <i>PowerPoint_97_SRVR.*</i> znajdujących się w katalogach <code>\Include\VCL\</code> oraz <code>\Source\Comservers\Office97\</code> <sup>1</sup> |
|  | <b>TWordApplication</b>       | Umożliwia połączenie z aplikacjami Worda. Komponent niewidzialny. Jego kod źródłowy znajduje się w plikach <i>Word_97_SRVR.*</i> znajdujących się w katalogach <code>\Include\VCL\</code> oraz <code>\Source\Comservers\Office97\</code>                          |

<sup>1</sup> Nazwa ostatniego katalogu będzie pochodzić od nazwy katalogu, w którym zainstalowana jest wybrana wersja pakietu Office. Wersję pakietu Office, z którą chcemy współpracować należy podać w trakcie instalacji Borland C++Buildera 5.



TExcelApplication

Umożliwia połączenie z aplikacjami Excela. Komponent niewidzialny. Jego kod źródłowy znajduje się w plikach *Excel\_97\_SRVR.\** znajdujących się w katalogach `\Include\VCL\` oraz `\Source\Comservers\Office97\`

## Karta Data Access

Komponenty Karty Data Access umożliwiają na dostęp z poziomu Borland C++ Builder 5 do danych zapisanych w systemach baz danych.















| Ikona | Typ         | Znaczenie   |
|-------|-------------|---|
|       | TDataSource | Komponent zapewnia połączenie pomiędzy zbiorem danych a kontrolką je przetwarzającą.  |
|       | TTable      | Komponent umożliwia pobieranie danych z tabel baz danych Paradox, dBASE, Access, FoxPro, InterBase, Oracle, Sybase, MSSQL Server, Informix, DB2.          |
|       | TQuery      | Składnik umożliwiający wykonanie w bazie danych polecenia typu SELECT.  |
|       | TStoredProc | Komponent umożliwia wywołanie kodu składowanego (procedury, funkcji) z bazy danych. Pozwala na przekazywanie parametrów wejściowych i odczytywanie wyniku |
|       | TDatabase   | Komponent definiujący bazę danych, z którą zamierzamy pracować.   |
|       | TSession    | Komponent odpowiadający za połączenie (sesję) z bazą danych.  |
|       | TBatchMove  | Komponent umożliwia wykonywanie operacji na grupie wierszy lub na całej tabeli.   |
|       | TUpdateSQL  | Komponent umożliwia wykonanie poleceń typu INSERT, UPDATE, DELETE.  |

## Karta Data Controls

Komponenty Karty Data Controls umożliwiają wizualizację danych z bazy danych.

| Ikona | Typ | Znaczenie |
|-------|-----|-----------|
|-------|-----|-----------|

## Programowanie wizualne – Borland C++ Builder

|   |                   |  |
|---|-------------------|--|
|    | TDBGrid           | Komponent umożliwia wizualizację danych w postaci tabelarycznej.   |
|    | TDBNavigator      | Komponent będący paskiem narzędziowym umożliwiającym wykonywanie różnych operacji na danych (przechodzenie pomiędzy wierszami, dodawanie, usuwanie, zatwierdzanie itd.). |
|    | TDBText           | Składnik umożliwiający wyświetlenie wartości z bazy w postaci pola tekstowego.   |
|    | TDBEdit           | Komponent wyświetlenia i edytowania wartości z bazy danych.  |
|    | TDBMemo           | Komponent umożliwiający wyświetlanie i edycję wieloliniowego tekstu z bazy danych.   |
|    | TDBImage          | Umożliwia wyświetlenie obrazu zapisanego w tabelach bazy danych.   |
|    | TDBListBox        | Element listowy umożliwiający prezentację i zmianę danych pochodzących z bazy danych.  |
|    | TDBComboBox       | Komponent typu combo (lista wyboru) umożliwiający wyświetlanie i pobieranie danych z bazy.   |
|  | TDBCheckBox       | Komponent typu opcja (check box) zaznaczony/niezaznaczony odpowiadający stosownym wartościom z bazy.   |
|  | TDBRadioGroup     | Grupa przycisków radiowych odpowiadająca wybranym wartościom z bazy danych   |
|  | TDBLookupListBox  | Element typu "podgląd" do wartości z innego zbioru danych. Ma postać listy.  |
|  | TDBLookupComboBox | Element typu lista wyboru. Umożliwia wyświetlenie wartości z innego źródła danych.   |
|  | TDBRichEdit       | Rozbudowany komponent edycyjny umożliwiający wyświetlanie i modyfikacje tekstu z bazy danych.  |
|  | TDBChart          | Komponent umożliwiający generowanie na formularzu wykresów na podstawie zbioru danych pobranego z bazy.  |

## Ćwiczenia

1. Utwórz nową aplikację, w której umieścisz komponent TLabel. W Inspektorze Obiektów ustaw dla niego własności: Caption="Przykład użycia komponentow z karty standard", Font->Size=20. Umieść na formularzu komponent TGroupBox, a następnie zmień jego własności: Caption="Styl czcionki", Font->Size=10. Na komponencie TGroupBox umieść cztery komponenty TCheckBox. Zmień ich własności Caption odpowiednio na: Pogrubiona, Pochylona, Podkreślona, Przekreślona. Aplikacja będzie miała za zadanie zmieniać styl czcionki użytej w komponencie TLabel w zależności od zaznaczenia stosownej opcji.

*Wskazówka:* Wykorzystaj zdarzenie OnClick dla komponentu TCheckBox i jego własności State oraz Checked. Posłuż się własnością Font->Style komponentu TLabel. Ustaw odpowiednio styl na fsBold (pogrubiony), fsItalic (pochylony), fsUnderline (podkreślony), fsStrikeOut (przekreślony). Przykład pochylania tekstu:

```
if (CheckBox2->Checked)
    Label1->Font->Style=TFontStyles()<<fsItalic;
```

2. Zmodyfikuj aplikację tak aby możliwe było "kumulowanie" stylów czcionki, czyli żeby jednocześnie mogła ona być zarówno pogrubiona, pochylona, podkreślona i przekreślona.

*Wskazówka:* Przy ustawianiu stylu wykorzystaj informacje już zawarte we własności Font->Style:

```
if (CheckBox2->Checked)
    Label1->Font->Style=Label1->Font->Style+
    TFontStyles()<<fsItalic;
```

3. Utworzona do tej pory aplikacja umożliwia zmianę stylu czcionki etykiety. Zaznaczenie odpowiedniej opcji stosownie zmienia styl etykiety. Jednakże odznaczenie checkboxa nie powoduje żadnej zmiany, a powinno odbierać dany atrybut stylu. Należy tak zmodyfikować program aby zaznaczenie opcji powodowało dodanie danego stylu czcionki, a odznaczenie zabranie.

*Wskazówka:* Wykorzystaj kod:

```
if (CheckBox2->Checked)
    Label1->Font->Style=Label1->Font->Style+
    TFontStyles()<<fsItalic;
else
    Label1->Font->Style=Label1->Font->Style+
    TFontStyles()>>fsItalic;
```

4. Dodaj do utworzonego formularza komponent TRadioGroup, a następnie zmień jego własności: Caption="Kolor czcionki", Font->Size=10. Rozwiń własność Items i w oknie dialogowym wpisz w osobnych liniach: Czerwony, Zielony, Niebieski. W ten sposób na formularzu uzyskamy trzy przyciski

radiowe powodujące stosowną zmianę koloru czcionki w etykiecie. Należy spowodować aby zaznaczenie opcji “Czerwony” zmieniało kolor czcionki elementu Label na clRed, “Zielony” - clGreen, “Niebieski” - clBlue.

*Wskazówka:* Należy wykorzystać zdarzenie OnClick i własność ItemIndex komponentu TRadioGroup. Informuje ona o tym, jaki jest indeks aktualnie zaznaczonej opcji. ItemIndex jest numerowany od 0:

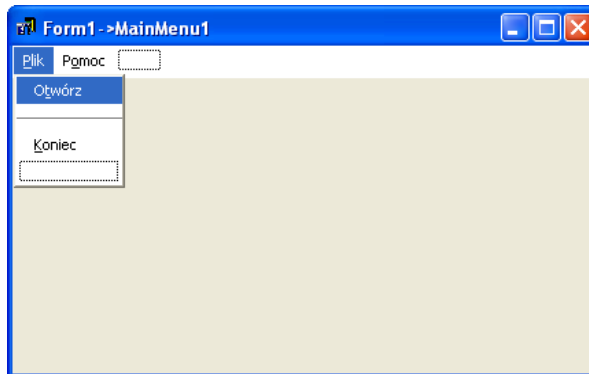
```
if (RadioGroup1->ItemIndex==0)
    Label1->Font->Color=clRed;
```

5. Umieść na formularzu komponent TEdit. Usuń dla niego napis z własności Text. Następnie umieść przycisk (TButton) i ustaw jego własność Caption na “Zmien napis”. Zmodyfikuj działanie aplikacji tak aby naciśnięcie przycisku powodowało przepisanie tekstu z pola edycyjnego do etykiety.

*Wskazówka:* Wykorzystaj zdarzenie OnClick dla przycisku:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    Label1->Caption=Edit1->Text;
}
```

6. Utwórz nową aplikację, w której umieścisz komponent TMainMenu umożliwiające zbudowanie menu głównego. Dwukrotnie klikając na komponencie TMainMenu uruchom edytor menu i zaprojektuj menu aplikacji tak jak na rysunku poniżej:



W celu uzyskania separatora ustaw własność Caption na -. Aby uzyskać znak podkreślenia umieść we właściwości Caption przed stosowną literą znak &. Dla opcji Otwórz ustaw własność ShortCut na Ctrl+T. Spowoduje to zdefiniowanie skrótu do tej opcji w postaci kombinacji klawiszy Ctrl+T. Spowoduj aby po naciśnięciu opcji Otwórz pojawił się komunikat “Otwórz”. W tym celu wykorzystaj funkcję:

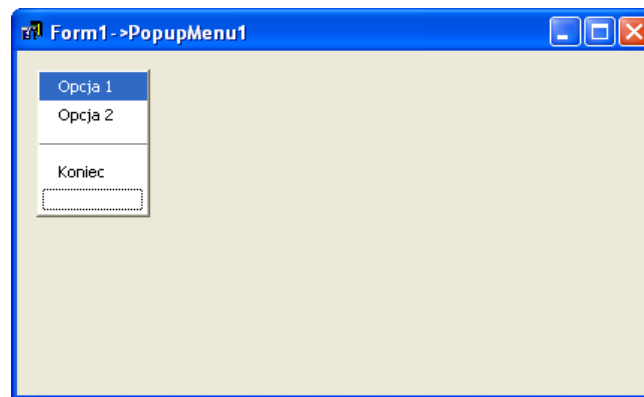
```
ShowMessage("Otwórz");
```

W analogiczny sposób zdefiniuj obsługę opcji Pomoc. Tak aby po jej wybraniu pojawiło się okno z komunikatem “Pomoc”.

Po wybraniu opcji Koniec wyświetl pytanie umożliwiające podjęcie decyzji o zamknięciu programu albo rezygnacji z wyjścia. Wykorzystaj poniższy kod:

```
if (Application->MessageBox("Czy zamknąć program?",  
    NULL, MB_OKCANCEL) == IDOK)  
    Close();
```

7. Dodaj do aplikacji z poprzedniego punktu komponent TPopupMenu umożliwiający zdefiniowanie menu podręcznego o wyglądzie:



Spowoduj aby po wybraniu dwóch pierwszych opcji pojawiał się stosowny komunikat, zaś po wybraniu Koniec następowało zamknięcie aplikacji. Po zaprojektowaniu menu ustaw własność formularza PopupMenu na nazwę zdefiniowanego komponentu menu. Uruchom i przetestuj aplikację.

8. Zmodyfikuj tworzoną aplikację dodając do niej pasek narzędzi. W tym celu z zakładki Win32 wybierz komponent TToolBar i umieść go na formularzu. Następnie wykorzystując menu kontekstowe (prawego klawisza myszy) utwórz na pasku narzędziowym dwa nowe przyciski oddzielone separatorem. Następnie umieść na formularzu komponent TImageList pochodzący z zakładki Win32. Kliknij dwukrotnie na tym komponencie i dodaj do niego dwie ikony lub bitmapy z rysunkami, które mają się pojawić na przyciskach z paska narzędziowego. Wykorzystaj przykładowe pliki z %BORLAND\_HOME%\Borland Shared\Images\Buttons. Ustaw Własność Images komponentu TToolBar na zdefiniowaną listę zdjęć. Powinno to spowodować umieszczenie ikon na przyciskach. W razie konieczności zamiany ikon posłuż się własnościami ImageIndex przycisków. Zdefiniuj zdarzenia obsługujące naciśnięcie przycisków tak aby wybranie pierwszego przycisku powodowało tą samą akcją co wybranie opcji Plik->Otwórz, zaś drugiego Plik->Koniec. W tym celu ustaw ich zdarzenia OnClick na już zdefiniowane wcześniej zdarzenia obsługujące wybór stosownych opcji.



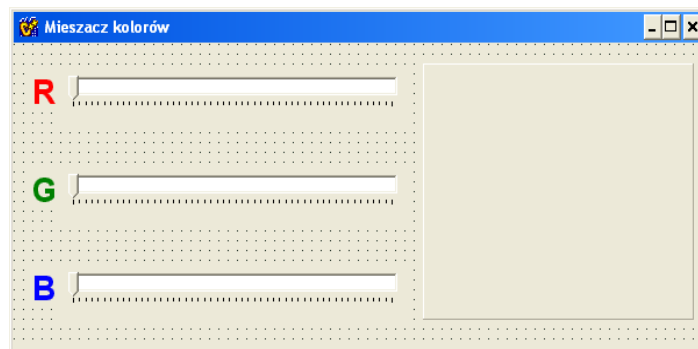
9. Umieść na tworzonej aplikacji komponent RichEdit znajdujący się na zakładce Win32. Ustaw dla niego własność Align na alClient, a następnie usuń tekst z własności Lines. W kolejnym kroku zmodyfikuj obsługę opcji Plik->Otwórz tak aby jej wybranie otwierało plik tekstowy znajdujący się w bieżącym katalogu. Wykorzystaj kod:

```
RichEdit1->Lines->LoadFromFile(".\\plik.txt");
```

10. Pobierz z zakładki Win32 komponent TStatusBar i umieść go na formularzu aplikacji. Dodaj w obsłudze zdarzenia Otwórz kod wyświetlający na pasku stanu informację o otwarciu pliku:

```
StatusBar1->SimpleText="Plik został otwarty";
```

11. Utwórz nową aplikację. Zaprojektuj formularz o przedstawionym poniżej wyglądzie:



Na formularzu umieść komponent TPanel z zakładki Standard. Usuń dla niego wartość własności Caption. Następnie umieść trzy komponenty TTrackBar z zakładki Win32. Dla każdego z suwaków ustaw własności: Min=0, Max=255, Frequency=5. W kolejnym kroku umieść trzy etykiety (komponent TLabel z zakładki Standard) opisujące za pomocą liter R, G, B kolory jakie reprezentują suwaki. Dla każdej z etykiet ustaw własności: Font=Arial, Size=20, Style=fsBold, a następnie zmodyfikuj stosownie ich kolor tak aby R było napisane kolorem czerwonym, G – zielonym, a B – niebieskim. Dla suwaka reprezentującego kolor czerwony obsłuż zdarzenie OnChange wywoływane w momencie zmiany położenia wskaźnika suwaka. Wykorzystaj funkcję RGB zwracającą na podstawie trzech podanych wartości kolorów barwę wynikową. Do pobrania aktualnej pozycji wskaźników suwaka wykorzystaj własność Position. Aby wygenerować podgląd uzyskanej barwy ustaw własność kolor panela. Posłuż się kodem:

```
Panel1->Color=(TColor) RGB(TrackBar1->Position,
```

```
TrackBar2->Position, TrackBar3->Position);
```

Dla pozostałych dwóch suwak zdefiniuj tak zdarzenie OnChange aby wywoływała się ta sama funkcja co dla suwaka obsługującego kolor czerwony. Ustaw tytuł okna na “Mieszacz kolorów”. Uruchom i przetestuj aplikację.

12. Spowoduj aby nastawy suwaków uwidaczniały się wewnątrz barwnego panela. Niech się tam pokaże tekst “(100,100,100)” opisujący wartości trzech barw składowych. Wykorzystaj kod:

```
Panel1->Caption=" (" +AnsiString(TrackBar1->Position)+", "+  
AnsiString(TrackBar2->Position)+", "+  
AnsiString(TrackBar3->Position)+")";
```

13. Dodaj do formularza checkbox z opisem “Heksadecymalnie” decydujący czy współrzędne mają się pokazać w systemie szesnastkowym. Posłuż się funkcją IntToHex zamieniającą liczbę dziesiętną na napis w systemie heksadecymalnym. Ponieważ ten sam kod byłby wykorzystywany zarówno w przy zmianie stanu checkboxa jak i przy zmianie położenia suwaka dlatego też zdefiniuj funkcję “Napis”. W tym celu otwórz plik nagłówkowy (naciśnij prawy klawisz myszy na zakładce z nazwą otwartego pliku cpp i wybierz Open Source/Header File). W części prywatnej klasy formularza zadeklaruj funkcję:

```
void Napis();
```

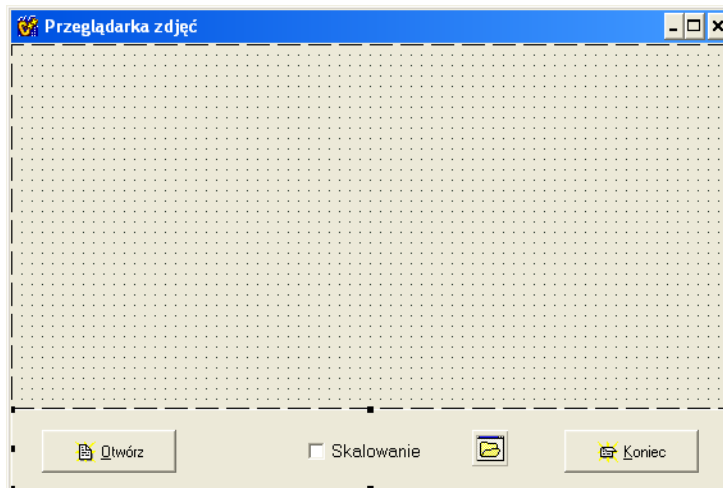
Wróć do pliku cpp i zdefiniuj ciało funkcji:

```
void TForm1::Napis()  
{  
    if(CheckBox1->Checked)  
        Panel1->Caption=" (" +IntToHex(TrackBar1->Position,2)+  
", "+IntToHex(TrackBar2->Position,2)+", "+  
    IntToHex(TrackBar3->Position,2)+")";  
    else  
        Panel1->Caption=" (" +AnsiString(TrackBar1->Position)+", "+  
    AnsiString(TrackBar2->Position)+", "+  
    AnsiString(TrackBar3->Position)+")";  
}
```

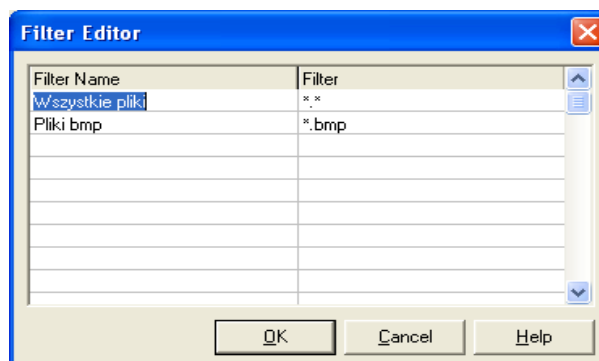
Następnie wywołaj ją w funkcji obsługi zdarzenia OnChange dla suwaków oraz w funkcji obsługującej zdarzenie OnClick dla checkboxu.

---

14. Utwórz nową aplikację o wyglądzie przedstawionym poniżej:



Umieść komponent TPanel z zakładki Standard i ustaw jego własność Align na alBottom. Spowoduje to ustawienie panela w dolnej części formularza. Na Panelu umieść dwa przyciski typu TBitBtn z zakładki Additdional. Ustaw dla nich stosowne napisy “Otwórz” i “Koniec”. Dla każdego z nich wybierz stosowną ikonę (własność Glyph). Następnie umieść na panelu chechbox x opisem “Skalowanie”. Ustaw tytuł okna na “Przeglądarka zdjęć”. W kolejnych krokach umieść komponent TImage z zakładki Additdional oraz komponent TOpenDialog z zakładki Dialogs. Dla komponentu reprezentującego zdjęcie ustaw własność Align na alClient. Dla komponentu TopenDialog ustaw własność Filter tak jak pokazano na poniższym przykładzie:



Spowoduje to rozciągnięcie komponentu na cały wolny obszar roboczy formularza. Obsłuż zamykanie formularza za pomocą przycisku “Koniec”. Wygeneruj zdarzenie OnClick dla przycisku “Otwórz” tak aby powodowało pokazanie okna dialogowego z wyborem plików a następnie wyświetlało wskazany plik. W zależności od stanu checkboxa dopasuj zdjęcie do rozmiarów komponentu lub pokaż je w oryginalnym rozmiarze (steruj własnością Stretch zdjęcia). Wykorzystaj kod:

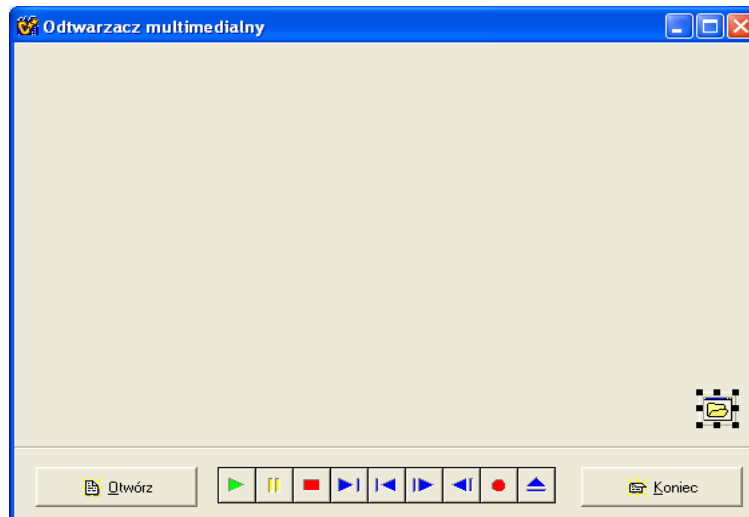
```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
```

```
if(OpenDialog1->Execute())
    Image1->Picture->LoadFromFile(OpenDialog1->FileName);
if(CheckBox1->Checked)
    Image1->Stretch=true;
}
//-----
void __fastcall TForm1::CheckBox1Click(TObject *Sender)
{
    if(CheckBox1->Checked)
        Image1->Stretch=true;
    else
        Image1->Stretch=false;
}
```

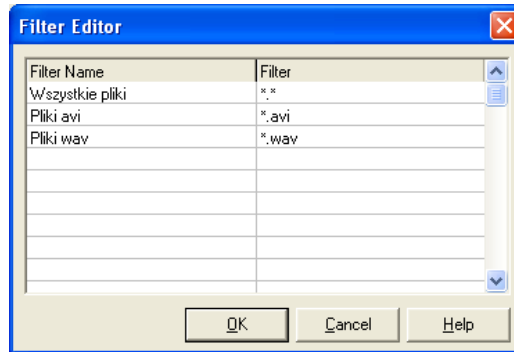
Uruchom i przetestuj aplikację.

---

15. Utwórz nową aplikację o wyglądzie jak poniżej:



W tym celu umieść na formularzu dwa panele. Dla jednego z nich ustaw własność `Align` na `alBottom`, zaś dla drugiego `Align=alClient`. Na dolnym panelu podobnie jak w poprzednim programie umieść dwa przyciski “Otwórz” i “Koniec”. Z zakładki `System` pobierz komponent `TMediaPlayer` i również umieść go na dolnym panelu. Dla komponentu multimedialnego ustaw własność `Display` na nazwę górnego panela. Wykorzystaj też komponent `TOpenDialog` i ustaw dla niego odpowiednio własności `filter`:

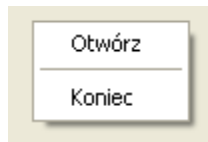


Posłuż się kodem:

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
    if(OpenDialog1->Execute())
    {
        MediaPlayer1->FileName=OpenDialog1->FileName;
        MediaPlayer1->Open();
        MediaPlayer1->DisplayRect=Panel2->ClientRect;
        MediaPlayer1->Play();
    }
}
```

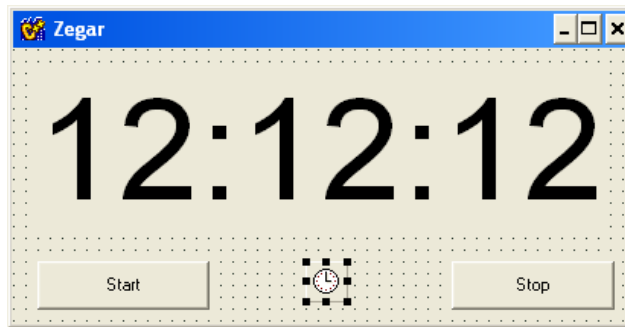
Uruchom i przetestuj program.

- Do poprzednio utworzonego programu dodaj menu kontekstowe (podręczne, prawego klawisza myszy) umożliwiające otwieranie pliku i zamykanie programu:



Wykorzystaj komponent TPopupMenu, przypisz go do własności PopupMenu komponentu TImage i wykorzystaj wcześniej zdefiniowane funkcje obsługi przycisków.

- 
- Utwórz nową aplikację, która będzie obsługiwać zegar. Jej wygląd przedstawia się następująco:



Umieść na formularzy dwa przyciski z napisami “Start” i “Stop”. Umieść etykietę i ustaw jej czcionkę na Arial, rozmiar 72. W tytule okna napisz “Zegar”. W celu zbudowania działającego zegara wykorzystaj komponent TTimer z zakładki System. Ustaw dla niego Interval na 1000 (1 sekunda) i Enabled na false. Chcemy aby po naciśnięciu przycisku “Start” zegar zaczął pokazywać co sekunda właściwy czas. W tym celu w obsłudze OnClick dla przycisku “Start” wywołamy pokazanie aktualnego czasu i włączenie timera, który zadba o właściwe odświeżanie:

```
Timer1->Enabled=true;  
Label1->Caption=TimeToStr(Time());
```

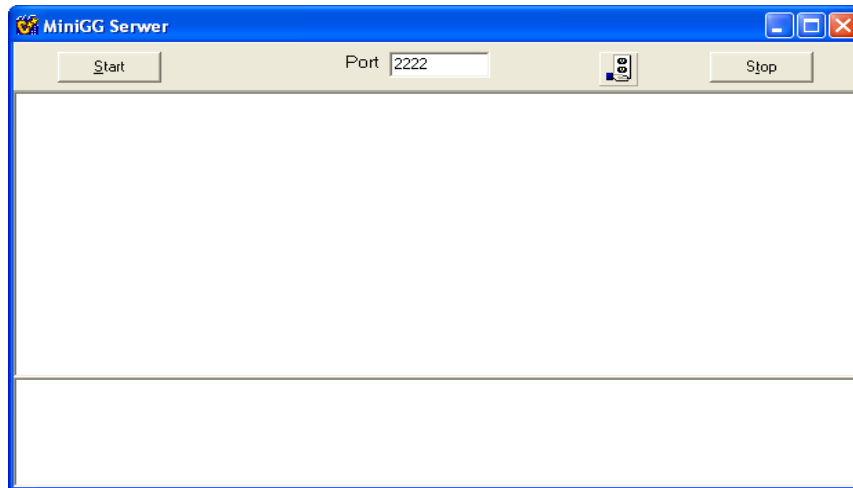
W obsłudze wyzwalającej timer co zadany interwał czasu pozostanie tylko odświeżać napis:

```
Label1->Caption=TimeToStr(Time());
```

Zaś w zdarzeniu OnClick dla przycisku “Stop” wyłączyć timer:

```
Timer1->Enabled=false;
```

- 
18. Utwórz nową aplikację, która będzie obsługiwać serwer rozmów internetowych (typu chat, gadu-gadu, itp.). W tym celu zbuduj formularz o wyglądzie przedstawionym poniżej:



Zainicjuj nowy projekt, wybierając opcję File->New Application z menu głównego IDE. Umieść na formularzu komponent TMemo i nazwij go MemoIn. Umieść na formularzu panel (TPanel) i wyczyść jego tytuł (Caption). Umieść na formularzu komponent TMemo i nazwij go MemoOut. Ustaw wyrównanie (Align) komponentów: MemoOut na alBottom, Panel1 na alTop i MemoIn na alClient. Ustaw na true właściwość ReadOnly komponentu MemoIn – jego zadaniem jest wyświetlanie komunikatów przychodzących i historii rozmowy (dane te nie powinny być modyfikowane). Bezpośrednio po uruchomieniu programu serwer nie będzie uruchomiony, nie będzie też przyłączony żaden klient, należy więc ukryć komponent MemoOut, ustawiając na false jego właściwość Enabled. Uniemożliwi to wpisywanie komunikatów jeśli serwer jest nieaktywny i nie ma połączenia z klientem. Dla obydwu komponentów typu Tmemo usuń tekst z własności Lines. Umieść na panelu (Panel1) dwa przyciski, nadając im nazwy StartButton i StopButton oraz tytuły (odpowiednio) Start i Stop. Dla przycisku StopButton ustaw własność Enabled na false. Umieść na panelu komponent TEdit, nadając mu nazwę PortEdit i poprzedzając go etykietą (TLabel) o treści (Caption) „Port”. Umieść na panelu komponent TServerSocket (ze strony Internet) i nadać mu nazwę MyServer. Obsłuż naciśnięcie przycisku StartButton tak aby uruchomiony został nasłuch na określonym w polu edycyjnym porcie. Posłuż się kodem:

```
void __fastcall TForm1::StartButtonClick(TObject *Sender)
{
    MyServer->Port = StrToInt(PortEdit->Text);
    Caption = "MiniGG Serwer ["+IntToStr(MyServer->Port)+"]";
    MyServer->Active = true;
    StopButton->Enabled = true;
    StartButton->Enabled = false;
    MemoIn->Lines->Add("Serwer: Start Serwera");
    PortEdit->Enabled = false;
}
```

Następnie obsłuż naciśnięcie przycisku StopButton tak aby powodowało ono wstrzymanie nasłuchu:

```
void __fastcall TForm1::StopButtonClick(TObject *Sender)
{
    MyServer->Active = false;
    StartButton->Enabled = true;
    StopButton->Enabled = false;
    PortEdit->Enabled = true;
    MemoOut->Enabled = false;
    MemoIn->Lines->Add("Serwer: Stop Serwera");
    Caption = "MiniGG Serwer";
}
```

Obsłuż zdarzenie połączenia się klienta (OnClientConnect) dla komponentu MyServer (TserverSocket):

```
void __fastcall TForm1::MyServerClientConnect(TObject *Sender,
    TCustomWinSocket *Socket)
{
    MemoIn->SetFocus();
    MemoIn->Lines->Add("Serwer: Połączono z "+
    Socket->RemoteAddress+": "+IntToStr(Socket->RemotePort));
    MemoOut->Enabled = true;
    MemoOut->SetFocus();
}
```

Obsłuż zdarzenie odłączenia się klienta (OnClientDisconnect):

```
void __fastcall TForm1::MyServerClientDisconnect(TObject *Sender,
    TCustomWinSocket *Socket)
{
    MemoIn->SetFocus();
    MemoIn->Lines->Add("Serwer: Rołączono z "+
    Socket->RemoteAddress+": "+IntToStr(Socket->RemotePort));
    MemoOut->Enabled = false;
}
```

Obsłuż zdarzenie odczytania komunikatu przesłanego przez klienta:

```
void __fastcall TForm1::MyServerClientRead(TObject *Sender,
    TCustomWinSocket *Socket)
{
    MemoIn->SetFocus();
    MemoIn->Lines->Add("Klient: "+Socket->ReceiveText());
    MemoOut->SetFocus();
}
```

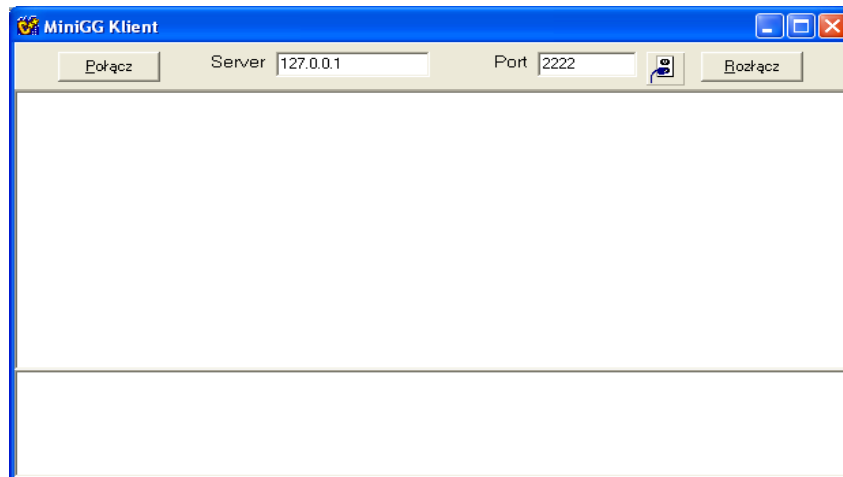
Dla komponentu MemoOut obsłuż zdarzenie OnKeyPress tak aby powodowało po naciśnięciu klawisza ENTER wysłanie komunikatu do podłączonego klienta:

```
void __fastcall TForm1::MemoOutKeyPress(TObject *Sender, char
&Key)
{
    if(Key == VK_RETURN)
    {
        MemoIn->Lines->Add("Serwer: "+MemoOut->Text);
        MyServer->Socket->Connections[0]->SendText(MemoOut->Text);
        MemoOut->Lines->Clear();
    }
}
```



```
    Key = 0;  
  }  
}
```

19. Utwórz nową aplikację, która będzie obsługiwać klienta rozmów internetowych (typu chat, gadu-gadu, itp.). W tym celu zbuduj formularz o wyglądzie przedstawionym poniżej:



Zainicjuj nowy projekt, wybierając opcję File->New Application z menu głównego IDE. Umieść na formularzu komponent TMemo i nazwij go MemoIn. Umieść na formularzu panel (TPanel) i wyczyść jego tytuł (Caption). Umieść na formularzu komponent TMemo i nazwij go MemoOut. Ustaw wyrównanie (Align) komponentów: MemoOut na alBottom, Panel1 na alTop i MemoIn na alClient. Ustaw na true właściwość ReadOnly komponentu MemoIn – jego zadaniem jest wyświetlanie komunikatów przychodzących i historii rozmowy (dane te nie powinny być modyfikowane). Bezpośrednio po uruchomieniu programu serwer nie będzie uruchomiony, nie będzie też przyłączony żaden klient, należy więc ukryć komponent MemoOut, ustawiając na false jego właściwość Enabled. Uniemożliwi to wpisywanie komunikatów jeśli aplikacja klienta nie będzie połączona do serwera. Dla obydwu komponentów typu Tmemo usuń tekst z własności Lines. Umieść na panelu przycisk TButton, nadając mu nazwę ConnectButton i tytułując „Połącz”. Umieść na panelu przycisk TButton, nadając mu nazwę DisconnectButton i tytułując „Rozłącz”. Umieść na panelu dwie kontrolki TEdit, nazywając je kolejno PortEdit i ServerEdit, odpowiednio ustaw ich zawartość początkową na 127.0.0.1 i 2222. Poprzedź kontrolki edycyjne etykietami (TLabel) o treści (kolejno) „Port” i „Serwer”. Umieść na panelu komponent TClientSocket (ze strony Internet) i nadać mu nazwę MySocket. Obsłuż naciśnięcie przycisku ConnectButton tak aby zainicjalizować połączenie z adresem serwera wskazanym w polach edycyjnych. Posłuż się kodem:

```
void __fastcall TForm1::ConnectButtonClick(TObject *Sender)
```

```
{
    MySocket->Address = ServerEdit->Text;
    MySocket->Port = StrToInt(PortEdit->Text);
    MySocket->Active = true;

    ConnectButton->Enabled = false;
    DisconnectButton->Enabled = true;
    MemoOut->Enabled = true;
    ServerEdit->Enabled = false;
    PortEdit->Enabled = false;
}
```

Następnie obsłuż naciśnięcie przycisku DisconnectButton tak aby powodowało ono odłączenie od serwera:

```
void __fastcall TForm1::DisconnectButtonClick(TObject *Sender)
{
    MySocket->Active = false;

    ConnectButton->Enabled = true;
    DisconnectButton->Enabled = false;
    MemoOut->Enabled = false;
    ServerEdit->Enabled = true;
    PortEdit->Enabled = true;
}
```

Obsłuż zdarzenie podłączenia do serwera (OnConnect) dla komponentu MySocket (TClientSocket):

```
void __fastcall TForm1::MySocketConnect(TObject *Sender,
    TCustomWinSocket *Socket)
{
    MemoIn->SetFocus();
    MemoIn->Lines->Add("Klient: Połączono z "+
    MySocket->Address+": "+IntToStr(MySocket->Port));
    MemoOut->SetFocus();
}
```

Obsłuż zdarzenie odłączenia od serwera (OnDisconnect):

```
void __fastcall TForm1::MySocketDisconnect(TObject *Sender,
    TCustomWinSocket *Socket)
{
    MemoIn->SetFocus();
    MemoIn->Lines->Add("Klient: Rołączono z "+
    MySocket->Address+": "+IntToStr(MySocket->Port));
    MemoOut->SetFocus();
}
```

Obsłuż zdarzenie odczytania komunikatu przesłanego przez serwer:

```
void __fastcall TForm1::MySocketRead(TObject *Sender,
    TCustomWinSocket *Socket)
{
    MemoIn->SetFocus();
    MemoIn->Lines->Add("Serwer: "+Socket->ReceiveText());
}
```

```
    MemoOut->SetFocus();  
}
```

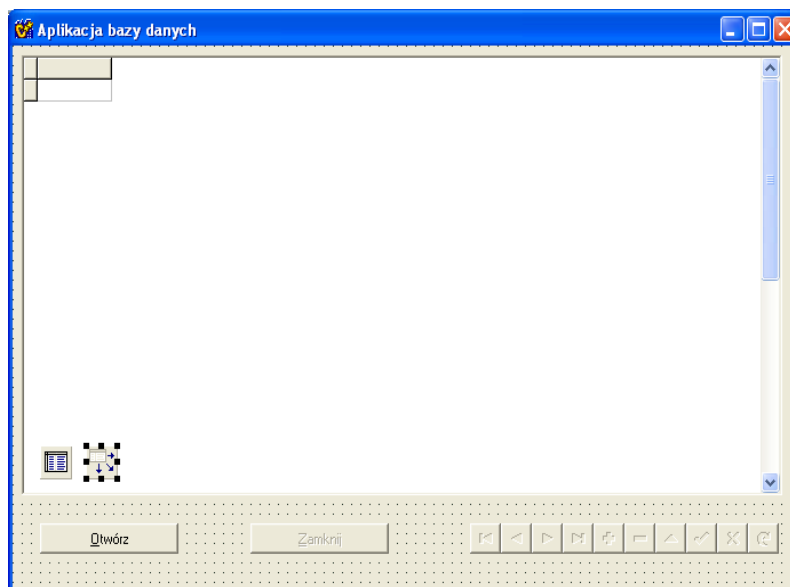
Dla komponentu MemoOut obsłuż zdarzenie OnKeyPress tak aby powodowało po naciśnięciu klawisza ENTER wysłanie komunikatu do serwera:

```
void __fastcall TForm1::MemoOutKeyPress(TObject *Sender, char  
&Key)  
{  
    if(Key == VK_RETURN)  
    {  
        MemoIn->Lines->Add("Klient: "+MemoOut->Text);  
        MySocket->Socket->SendText(MemoOut->Text);  
        MemoOut->Lines->Clear();  
        Key = 0;  
    }  
}
```

Uruchom i przetestuj obydwie programy.

---

20. Utwórz nową aplikację o wyglądzie przedstawionym poniżej:



W tym celu umieść na formularzu komponent TDBGrid z zakładki Data Controls. Ustaw jego rozmiary tak aby pokrywał większą część okna. Następnie z zakładki Data Access pobierz komponent TTable i umieść go w dowolnym miejscu formularza. Dla tego komponentu ustaw własności: DatabaseName na BCDEMOS, TableName na country.db (obydwe wartości należy wybrać z listy). W kolejnym kroku umieść w dowolnym miejscu komponent TDataSource z zakładki Data Access. Ustaw dla niego własność DataSet na Table1. Następnie ustaw własność DataSource komponentu DBGrid1 na DataSource1. Dodaj do formularza komponent TDBNavigator (z zakładki Data Controls) i umieść go poniżej komponentu DBGrid1. Dla paska nawigatora ustaw własność

DataSource na DataSource1. Umieść na formularzu dwa przyciski i zatytułuj je odpowiednio Otwórz i Zamknij (przycisk Zamknij ma być domyślnie niedostępny, ustaw dla niego własność Enabled na false). Wygeneruj zdarzenie Onclick dla przycisku Otwórz i umieść kod powodujący wyświetlenie danych z tabeli bazy danych (ustaw własność Active komponentu Table1 na true):

```
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
    Table1->Active=true;
    BitBtn2->Enabled=true;
    BitBtn1->Enabled=false;
}
```

Spowoduj aby naciśnięcie przycisku Zamknij powodowało ukrycie wyświetlanych danych:

```
void __fastcall TForm1::BitBtn2Click(TObject *Sender)
{
    Table1->Active=false;
    BitBtn1->Enabled=true;
    BitBtn2->Enabled=false;
}
```

Uruchom i przetestuj program.

## 8. Podstawowe elementy biblioteki VCL i hierarchia komponentów

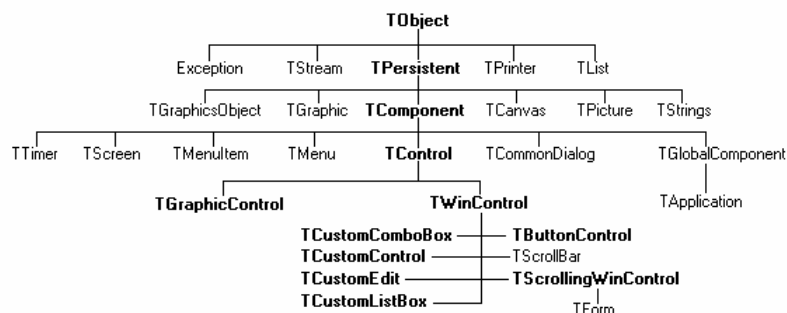
Komponenty VCL są podstawowymi elementami, z których budujemy aplikację. W rozdziale tym omówione zostaną podstawowe elementy VCL oraz przedstawiona zostanie ich struktura hierarchiczna.

### Hierarchia komponentów VCL

W ogólnym przypadku rozróżniamy cztery podstawowe rodzaje komponentów:

- Komponenty standardowe. Są one najczęściej używane przez programistów, dlatego większość z nich umieszczona jest na pierwszej karcie palety komponentów — karcie Standard.
- Komponenty sterujące. Nie są one dostępne w bibliotece standardowej.
- Komponenty graficzne. Służą do wypisywania tekstu bezpośrednio na formularzu oraz do wyświetlania grafiki.
- Komponenty niewidoczne. Stają się niewidoczne po uruchomieniu programu. Wszystkie komponenty z karty Dialogs oraz niektóre z kart System i Servers są obiektami, które przestajemy widzieć w działającej aplikacji.

Poniżej na rysunku przedstawiono fragment drzewa hierarchicznego obiektów z biblioteki VCL:



### Klasa TObject

Klasa TObject jest przodkiem wszystkich typów obiektowych Borland C++ Builder 5. Najczęściej nie korzysta się bezpośrednio z właściwości i metod, które nam udostępnia.

## ***Klasa TPersistent***

Wszystkie typy obiektowe, mające zdolność posługiwania się strumieniami, pochodzą właśnie od tej klasy. Klasa ta w rzeczywistości nie definiuje nowych właściwości ani pól, definiuje natomiast destruktor `~TPersistent()` oraz sześć metod:

`Assign()` — metoda przypisania obiektowi właściwości i atrybutów innego obiektu.

`AssignTo()` — metoda odwrotna do poprzedniej. Przypisuje danemu obiektowi kopię własnych właściwości i atrybutów.

`DefineProperties()` — ta metoda definiuje sposób przypisania strumieniowi pewnych dodatkowych właściwości komponentu.

`GetNamePath()` — umożliwia odczytanie nazwy obiektu oraz jego ustalonych właściwości w inspektorze obiektów.

`GetOwner()` — podaje właściciela obiektu.

`TPersistent()` — tworzy nowy obiekt.

## ***Klasa TComponent***

Z klasy tej pochodzi każdy komponent C++ Buildera 5. Wprowadzone przez nią właściwości i metody pozwalają na obsługę komponentów poprzez inspektora obiektów.

## ***Klasa TControl***

Komponenty wizualne reprezentowane w tej klasie są widoczne w czasie działania programu, chociaż istnieją sposoby by je ukryć lub uczynić niewidocznymi w trakcie działania programu. Obiekty tej klasy posiadają szereg właściwości. Poniżej zaprezentowano niektóre z nich.

### ***Właściwości klasy TControl***

`Align` — określa w jaki sposób komponent ma być ustawiony na formularzu (obszarze klienta). Jeżeli np. wybierzemy w inspektorze obiektów `alClient`, wówczas komponent ten pokryje cały dostępny obszar formularza. Właściwość tego typu aktywna jest np. dla komponentów typu `TPanel`, `TGroupBox` czy `TRadioGroup` z karty `Standard`.

**Anchors** — określa położenie komponentu w stosunku do jednego z rogów formularza.

**Caption** — opisuje komponent.

**ClientHeight** oraz **ClientWidth** — określa wymiary komponentu (wysokość i długość) w obszarze klienta.

**Color** — ustala kolor wypełnienia (wnętrza) komponentu.

**Cursor** — określa postać kursora, który będzie widoczny w obszarze danego komponentu.

**Enabled** — określa, czy komponent będzie dostępny. Jeżeli posługując się np. przyciskiem typu `TButton` napiszemy:

```
Button1->Enabled = FALSE;
```

przycisk będzie widoczny, ale nie będzie aktywny. Powrót do normalnej sytuacji możliwy jest dzięki:

```
Button1->Enabled = TRUE;
```

Analogicznych ustawień dokonamy też przy pomocy inspektora obiektów.

**Font** — ustala rodzaj czcionki napisów widocznych w obszarze komponentu.

**Hint** — ta właściwość sprawia, że można wpisać „dymek podpowiedzi” (hint, tooltip). Dodatkowo własność `ShowHint` musi być ustalona jako `TRUE`.

**Height** i **Width** — określają rozmiar komponentu.

**Text** — dzięki tej właściwości tekst wyświetlany jest na obszarze komponentu. Stosujemy ją m.in. do obiektów typu `TEdit`.

**Top**, **Left** — określają odległości komponentu od krawędzi odpowiednio górnej i lewej formularza.

**Visible** — określa, czy komponent ma być widoczny. Jeżeli w programie napiszemy:

```
Button1->Visible = FALSE;
```

komponent pozostanie całkowicie niewidoczny do czasu wywołania:

```
Button1->Visible = TRUE;
```

Czynność tę można również wykonać przy pomocy inspektora obiektów.

### **Zdarzenia klasy TControl**

Klasa TControl udostępnia nam również szereg użytecznych zdarzeń. Do najczęściej używanych należą:

OnClick — po kliknięciu obszaru komponentu zostanie wywołana funkcja obsługi wybranego zdarzenia.

OnDblClick — dwukrotne kliknięcie obszaru komponentu spowoduje wywołanie funkcji odpowiedniego zdarzenia.

OnResize — wywołuje np. funkcję obsługi zdarzenia po zmianie rozmiaru komponentu.

OnMouseDown — wywołuje reakcję na zdarzenie polegające na kliknięciu komponentu.

OnMouseMove — każdy ruch myszką nad komponentem wywoła funkcję odpowiedniego zdarzenia.

OnMouseUp — jak wyżej, tyle że w przypadku puszczenia przycisku muszki.

TControl udostępnia nam również zdarzenia związane z przesuwaniem komponentów przy pomocy myszki: OnDragOver, OnDragDrop, OnEndDrag, OnStartDock czy OnStartDrag.

### **Klasa TGraphicControl**

Reprezentuje nieaktywne komponenty wykorzystywane w różnego rodzaju operacjach związanych z grafiką. Widoczne na ekranie, komponenty te mogą wyświetlać tekst lub grafikę. Z najpowszechniej stosowanych komponentów tego typu należy wymienić: TBevel, TImage, TPaintBox, TShape, TSpeedButton, TSplitter oraz TCustomLabel, od którego wywodzą się z kolei TDBText i TLabel. Komponenty tego rodzaju mogą nie tylko obsługiwać zdarzenia, których źródłem jest myszka, ale również mogą być używane w funkcjach obsługi innych zdarzeń. Jako przykład praktycznego wykorzystania jednego z komponentów TLabel niech nam posłuży przykład funkcji obsługi zdarzenia reagującego na zmianę położenia myszki na formularzu:

```
void __fastcall TForm1::OnMouseMove(TObject *Sender, TShiftState Shift,
```



```
int X, int Y)
{
    Label1->Font->Style = TFontStyles() << fsBold;
    Label1->Font->Size = 16;
    Label1->Font->Color = clBlue;
    Label1->Top = Y;
    Label1->Left = X;
    Label1->Caption = "Tekst ciągnięty za myszką X=" + IntToStr(X) +
        +" Y= " +IntToStr(Y);
}
```

Aplikacja po uruchomieniu będzie pokazywać odpowiedni napis oraz aktualne współrzędne kursora myszki. Funkcja ta została zbudowana w bardzo prosty sposób. Na obszar formularza umieszczony został komponent typu `TLabel`. Następnie w karcie zdarzeń inspektora obiektów wybrane zostało zdarzenie `OnMouseMove`. Kolejnym krokiem było zdefiniowanie ciała funkcji `Tform1::OnMouseMove`. Do konwertowania wartości numerycznych współrzędnych na napisy została wykorzystana funkcja:

```
extern PACKAGE AnsiString __fastcall IntToStr(int Value);
```

Funkcja ta konwertuje dane typu całkowitego (int) na dane typu `AnsiString`.

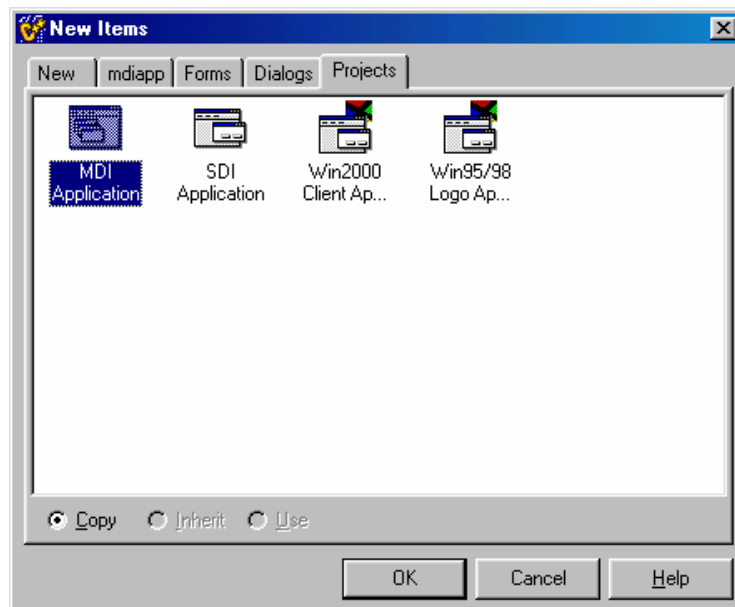
## **Klasa `TWinControl`**

Wszystkie okna edycyjne, listy wyboru, przyciski itp. są obiektami potomnymi tej klasy. Komponenty okienkowe mogą być aktywne, posiadają swoje własne identyfikatory oraz możliwość przewijania. Klasa ta posiada szereg właściwości, metod i zdarzeń. Wykorzystanie kilku z nich prezentuje poniższy przykład:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    Memo1->Brush->Color = clBlue;
    Memo1->Font->Color = clYellow;
    Memo1->Font->Style = TFontStyles() << fsBold;
    Memo1->Left = ClientRect.Left;
    Memo1->Text = "Tekst";
    Form1->ScrollBy(1,1);
    Memo1->ScrollBy(1,1);
}
```

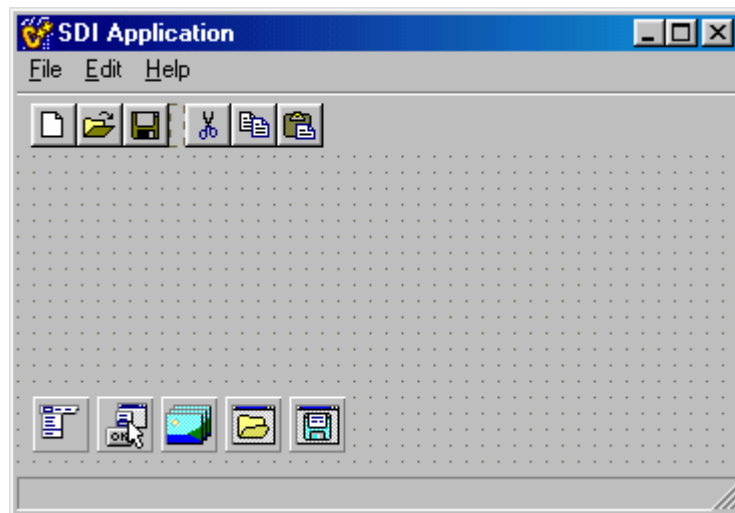
## 9. Aplikacje SDI oraz MDI

Wszystkie przedstawione w poprzednich rozdziałach przykładowe aplikacje w każdym szczególe konstruowaliśmy samodzielnie nabierając wprawy w manipulowaniu komponentami biblioteki VCL. Należy jednak pamiętać, że istnieje dużo prostszy (ale mniej kształcący) sposób zaprojektowania formularza. Można w tym celu skorzystać z menu File|New|Projects. W ten sposób dostaniemy się do zakładki z gotowymi szablonami formularza:



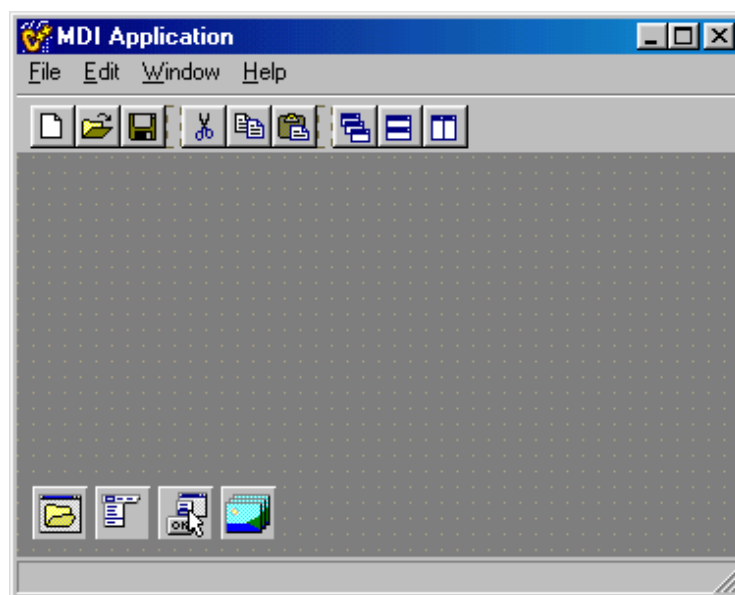
### **Aplikacje jednodokumentowe**

Wybierając przykładowy projekt aplikacji jednodokumentowej SDI Application (ang. *Single Document Interface*) otrzymujemy gotowy do użycia i ewentualnie dalszego uzupełniania formularz. Widzimy na nim gotowe komponenty TSaveDialog, TOpenDialog, TImageList, TActionList, TMainMenu, TStatusBar. Wszystkie mają już odpowiednio wykonane przypisania. Aplikacja taka, przynajmniej w swej warstwie edycyjnej jest praktycznie gotowa do użycia.

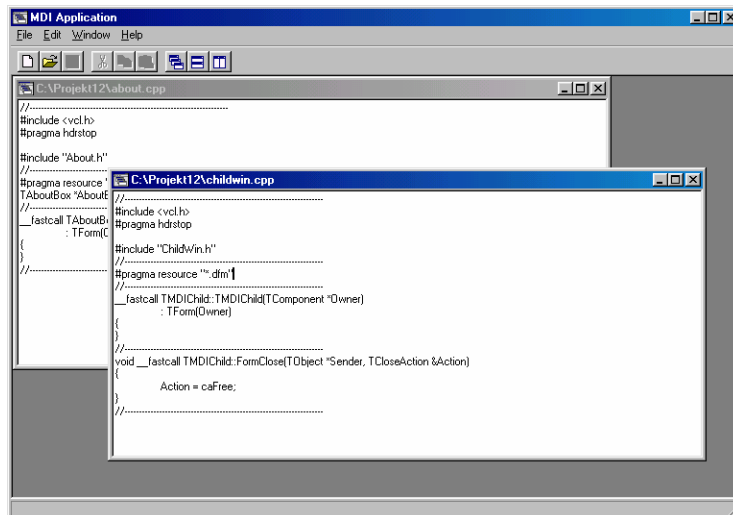


## Aplikacje wielodokumentowe

Aplikacje wielodokumentowe – MDI Application (ang. *Multi Document Interface*) służą do zarządzania zdarzeniami zachodzącymi w kilku oknach jednocześnie. Podstawową rzeczą, jaka odróżnia je od aplikacji SDI, jest styl stosowanego formularza. O ile w przypadku aplikacji SDI styl formularza reprezentowany przez właściwość `FormStyle` jest typu `fsNormal` (zob. karta właściwości inspektora obiektów), to w przypadku aplikacji MDI formularz posiadać będzie styl `fsMDIForm`. Wszystkie jego okna potomne reprezentowane będą przez formularze `fsMDIChild`. Centralną część formularza widocznego stanowi tzw. okno klienta (ang. *client window*).



Wszystkie okna potomne (ang. *child window*) umieszczane w oknie klienta, są całkowicie mu podporządkowane, tak jak pokazuje to poniższy rysunek:



Okna takie możemy dowolnie konfigurować korzystając z przycisków znajdujących się na pasku menu lub bezpośrednio z głównego menu (menu systemowego).

## **Literatura**

1. Mirosław J. Kubiak „Programuję w językach C/C++ i C++ Builder” HELION 2003
2. Maciej Dorobek „C++ Builder. Podręcznik” MIKOM 2002
3. Andrzej Daniluk „C++ Builder 5. Ćwiczenia praktyczne” HELION 2001
4. Jarrod Hollingworth, Dan Butterfield, Bob Swart, Jamie Allsop „C++ Builder Vademecum profesjonalisty.” HELION 2001
5. Jim Mischel, Jeff Duntemann „Borland C++ Builder” MIKOM 1997
6. Andrzej Stasiewicz „Borland C++ Builder. Całkiem inny świat” HELION 1998

## Zaliczenie

1. Napisać program rozwiązujący równanie liniowe postaci  $ax+b=0$ , gdzie  $a$  i  $b$  oznaczają liczby podane przez użytkownika. Aplikacja ma podać rozwiązanie wypisując na ekran stosowny tekst (np. Równanie  $0x+6=0$  nie ma rozwiązań, Rozwiązaniem równania  $2x-6=0$  jest  $x=3$ , Rozwiązaniem równania  $0x+0=0$  jest dowolna liczba  $x$ ). Napis ten ma być koloru zielonego jeśli równanie posiada jedno rozwiązanie, niebieskiego – jeśli rozwiązaniem równania jest dowolny  $x$ , czerwonego – gdy równanie nie ma rozwiązania.
2. Napisać program rozwiązujący równanie kwadratowe postaci  $ax^2+bx+c=0$ , gdzie  $a$ ,  $b$  i  $c$  oznaczają liczby podane przez użytkownika. Aplikacja ma podać rozwiązanie wypisując na ekran stosowny tekst (np. Równanie  $1x^2+8x+1=0$  nie ma rozwiązań, Rozwiązaniem równania  $1x^2-2x+1=0$  jest  $x=1$ , Rozwiązaniem równania  $0x^2+0x+0=0$  jest dowolna liczba  $x$ , Rozwiązaniem równania  $2x^2-3x-2=0$  jest  $x_1=-0.5$  i  $x_2=2$ ). Napis ten ma być koloru zielonego jeśli równanie posiada rozwiązanie, niebieskiego – jeśli rozwiązaniem równania jest dowolny  $x$ , czerwonego – gdy równanie nie ma rozwiązania.
3. Napisać program umożliwiający grę w kółko i krzyżyk (na planszy  $3 \times 3$ ) pomiędzy dwoma graczami lub graczem i komputerem.

### Oceny:

- dostateczny – poprawne rozwiązanie zadania 1.
- dobry – poprawne rozwiązanie zadań 1, 2.
- bardzo dobry – poprawne rozwiązanie zadań 1, 2, 3.

### Wskazówki i uwagi:

- w programach 1 i 2 użyteczne mogą być funkcje `StrToFloat`, `StrToInt`, `FloatToStr`, `IntToStr` umożliwiające odpowiednio konwersję napisu na liczbę rzeczywistą, napisu na liczbę całkowitą, liczby rzeczywistej na napis i liczby całkowitej na napis.
  - końcowa ocena programów odbędzie się na dwóch ostatnich zjazdach.