

Programowanie w języku C/C++ dla starszych klas uczniów szkół podstawowych



Autor: Zbigniew Wojtkowiak

Spis treści

Temat 1. „Hello Word”	3
Temat 2. Zmienne	7
Temat 3. Instrukcja warunkowa IF ... ELSE ...	11
Temat 4. Pętla FOR	15
Temat 5. Pętle „while” i „do... while...”	17
Temat 6. Kalkulator IF	20
Temat 7. Tabliczka mnożenia	22
Temat 8. Funkcja SWITCH – Menu	24
Temat 9. Znajdowanie maksimum	28
Temat 10. Sortowanie bąbelkowe	30
Temat 11. Borland C++ Builder instalacja	31
Temat 12. Borland C++ Builder – Hello World!	33
Temat 13. Światła drogowe	38
Temat 14. Kółko i krzyżyk	40
Temat 15. Kalkulator graficzny	46
Temat 16. Cymbergaj	50

Temat 1. „Hello Word”

Cel zajęć

Podczas pierwszych zajęć uczniowie zapoznają się z podstawowymi funkcjami dostępnymi w kompilatorze Dev C++. Poznają historię C++, jego składnię oraz napiszą swój pierwszy program.

Instalacja oraz zapoznanie się z programem „Dev C++”

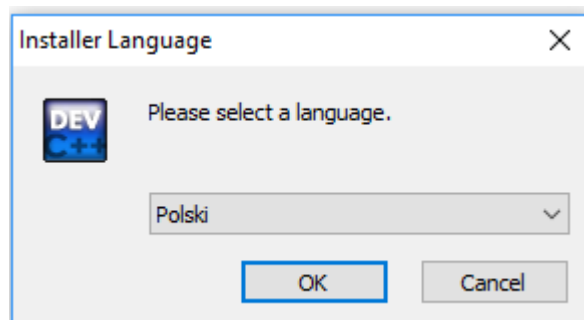
Programowanie jest to jeden z działów informatyki, zajmujący się procesem projektowania, tworzenia, testowania i utrzymywania kodu źródłowego programów komputerowych.

Programowanie jest sztuką, która wymaga szerokiej wiedzy z zakresu matematyki oraz informatyki. Dziedziny te dawniej stanowiły jedną całość, jednak ze względu na bardzo szybki rozwój informatyki, uległa ona wyodrębnieniu. Istniejące języki programowania są stale rozwijane i modyfikowane. W tym podręczniku zajmiemy się jednym z najbardziej rozpowszechnionych, jakim jest C++.

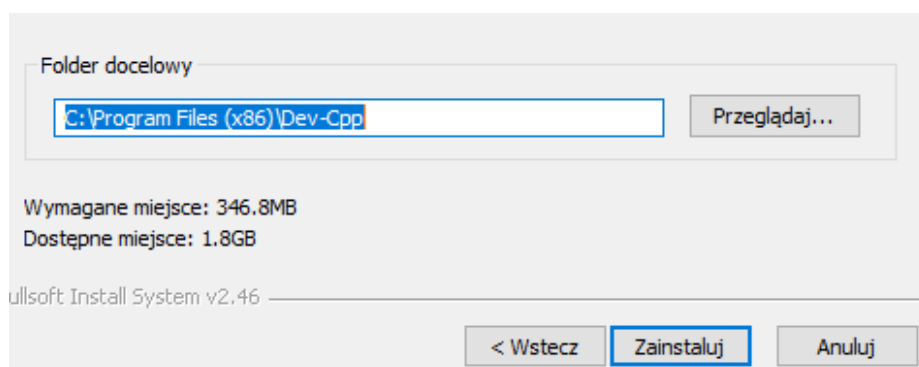
C++ jest jednym z pierwszych obiektowych języków programowania, powstał jako rozszerzenie języka C, został zaprojektowany przez Bjarne Stroustrupa w 1979r. Dzięki dużej wydajności kodu ciągle pozostaje jednym z najpopularniejszych języków programowania ogólnego przeznaczenia na świecie. Programy napisane w tym języku znajdują zastosowanie w aplikacjach oraz systemach operacyjnych, są one niezależne od konkretnej platformy systemowej oraz sprzętowej. Duża wydajność stworzonych aplikacji wynika z bezpośredniego dostępu do zasobów sprzętowych uruchamianych programów.

Instalacja oraz zapoznanie się z programem „Dev C++”

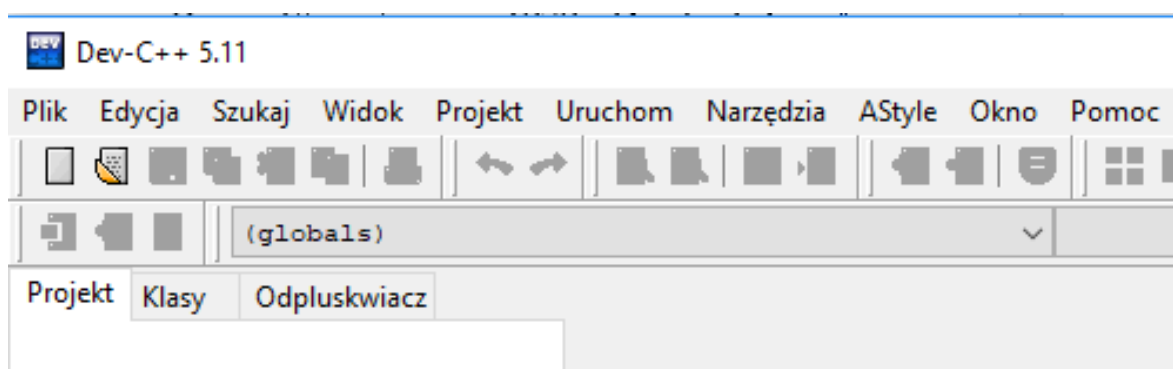
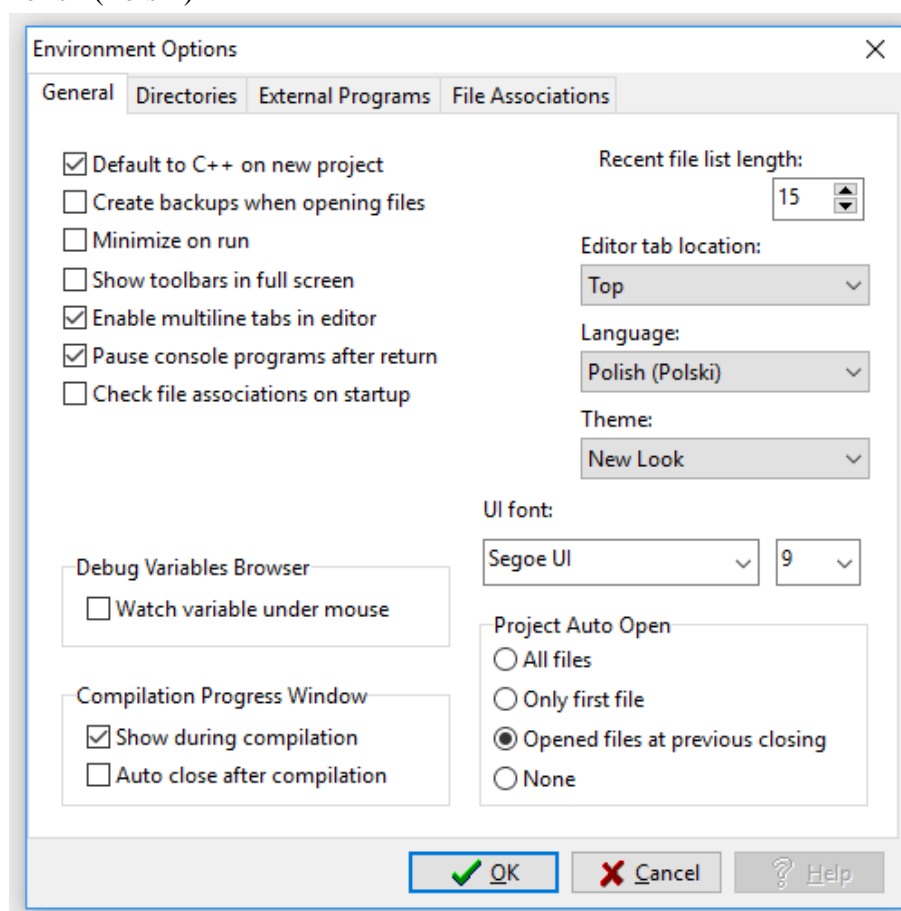
Pierwsze programy będziemy tworzyli z wykorzystaniem darmowego środowiska programistycznego wydanego na licencji GPL (licencja wolnego i otwartego oprogramowania). Program ten obsługuje języki C i C++. Działa on zarówno w systemach operacyjnych klasy Windows, jak i Linux. Jest zintegrowany z MinGW (windowsowym portem kompilatora GCC). Kody opisane w skrypcie powstały z użyciem wersji programu: Dev C++ 5.11.



Instalacja przebiega w sposób prosty. Do instalacji używamy pliku "Dev-Cpp 5.11 TDM-GCC 4.9.2 Setup.exe". Dużą zaletą programu jest to, że jest on dostępny w języku polskim. Do instalacji wymagane jest 346,8 MB miejsca na dysku.



Jeżeli język nie został automatycznie zmieniony na „Polski” w czasie instalacji, możemy go zmienić w poniższej lokalizacji: Tools → Environment Options → zakładka General → okno Language ustawiamy na Polish (Polski)



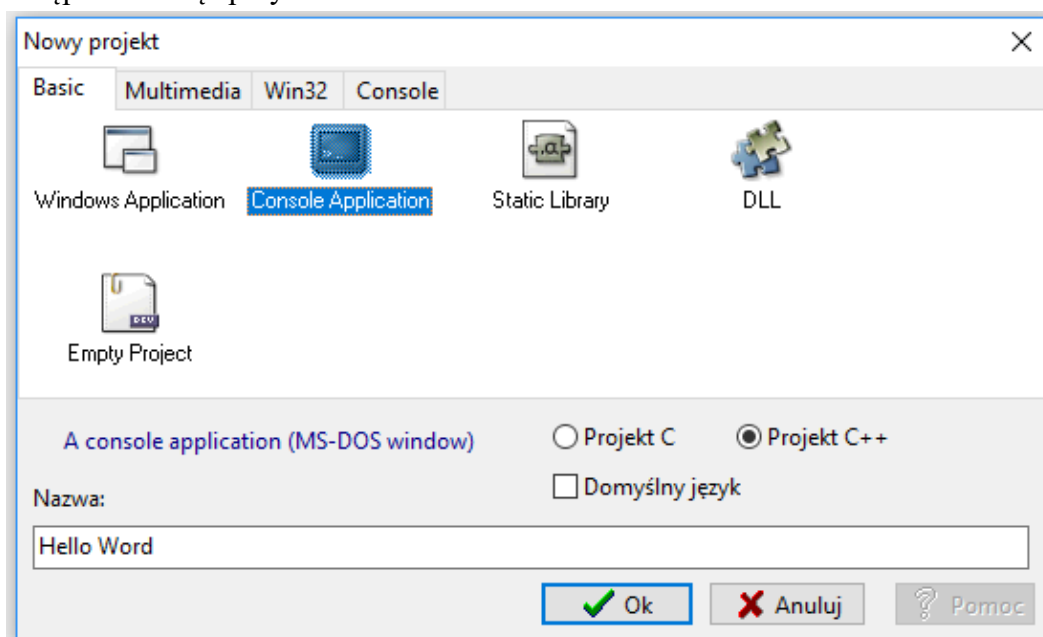
Na górnej wstążce znajdują się standardowe zakładki do obsługi oraz edycji tworzonych programów. Poniżej na pasku narzędziowym, po uruchomieniu programu aktywne są dwie pierwsze ikony:

„Nowy projekt” oraz „Otwórz projekt lub plik”. W zakładce „Projekt” wyświetlane będą aktualnie otwarte projekty, natomiast w zakładce „Klasy” wyświetlane są zdefiniowane w projekcie klasy, funkcje i zmienne (o tym czym są te pojęcia dowiemy się w dalszej części skryptu).

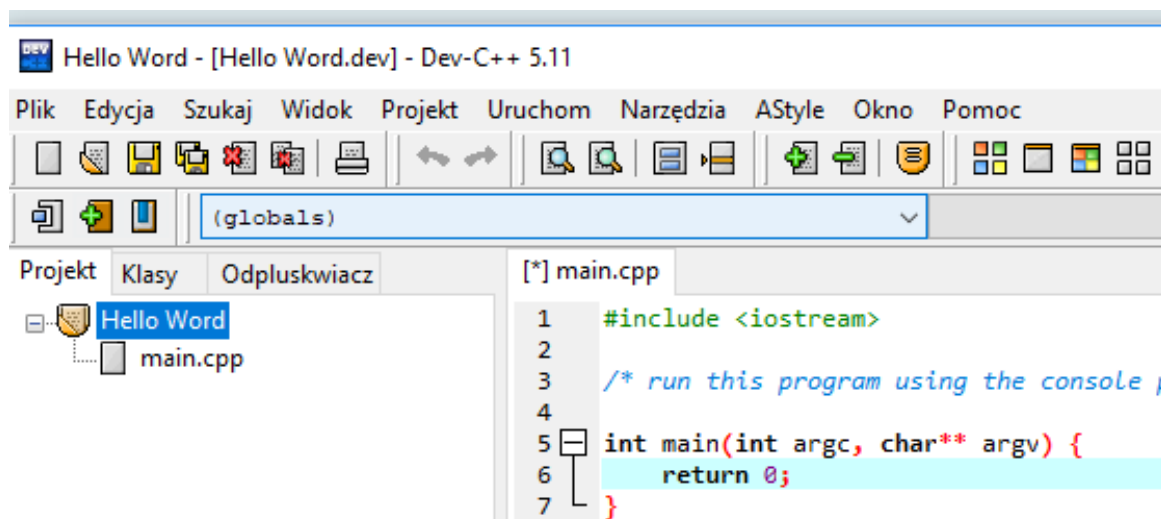
Pierwszy program

Tworzenie projektu możliwe jest poprzez wybranie z menu: Nowy→Projekt lub klikając na pasku narzędzi pierwszą ikonę i kliknięciu w opcję „Projekt...”.


Po tych czynnościach na monitorze pojawi się okno w którym wybieramy rodzaj tworzonej aplikacji. Wybieramy ikonę **Console Application** (aplikacja konsolowa), w polu **Nazwa** możemy wpisać nazwę swojego projektu, zmienić używany język (upewniamy się, że zaznaczona jest opcja **Projekt C++**), a następnie kliknąć przycisk **ok**.



Po kliknięciu przycisku **ok** wybieramy lokalizację, w której będzie zapisany nasz projekt. Po zaakceptowaniu nasz projekt pojawi się w pierwszej zakładce, natomiast w oknie z prawej strony pojawi się automatycznie utworzony plik ***.cpp**.

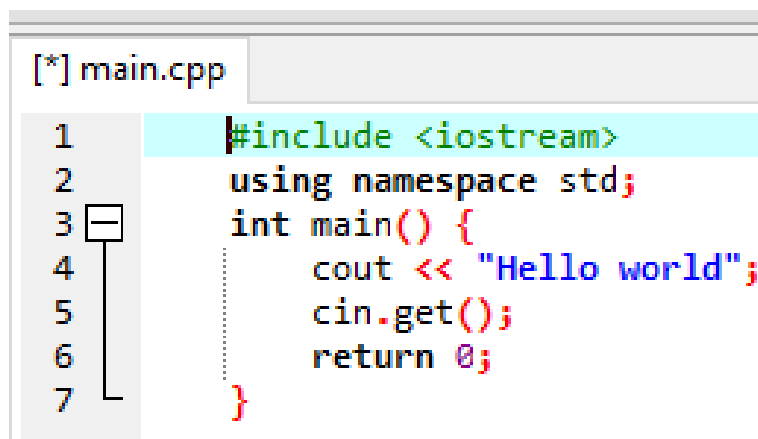


Każdy program C ++ zaczyna się od funkcji main(). Aby skompilować i uruchomić program należy wcisnąć przycisk F11, lub kliknąć w menu Uruchom→Kompiluj i uruchom. Ta sama opcja jest

dostępna z poziomu paska narzędziowego . Po skompilowaniu i uruchomieniu program będzie oczekiwał na wciśnięciu przycisku Enter.

Poniżej prostsza wersja programu z użyciem funkcji „cout”.

„Cout” to standardowy strumień wyjścia, który drukuje ciąg znaków „Witaj, świecie!” na monitorze. Na końcu linii kończących funkcje umieszczony jest średnik „;”.

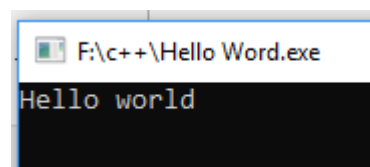


```
[*] main.cpp
1  #include <iostream>
2  using namespace std;
3  int main() {
4      cout << "Hello world";
5      cin.get();
6      return 0;
7  }
```

Opis wykorzystanych funkcji w powyższym programie:

#include <iostream> //biblioteka związana ze strumieniami cin i count (wyjścia/wejścia),
using namespace std ; //ustawienie wykorzystania przestrzeni nazwy „std”, nie będzie zatem potrzeby pisania pełnej składni „std::cout” (na tym poziomie nie będziemy zajmować zagadnieniem przestrzeni nazw),
int main() {} //funkcja wywoływana przez system, ciało funkcji umieszczone jest w nawiasie klamrowym { },
cout << "Hello world"; // Napis umieszczony w cudzysłowie zostaje przekierowany (symbol „<<”) na ekran (count – wyjściowy strumień danych) ,
cin.get(); //program oczekuje na wciśnięcie klawisza Enter.
return 0; //informujemy system, że program może zakończyć działanie bez zgłaszania błędów

Po skompilowaniu program uruchomi się i wyświetli następujący tekst „Hello world”.



```
F:\c++\Hello Word.exe
Hello world
```

W przypadku problemów z kompilacją programu należy zapisać go w innej lokalizacji (najlepiej dysku C:).

Warto zapamiętać

Większość programów będzie rozpoczynało się od wczytanie biblioteki „iostream” oraz określenia przestrzeni nazw „std”:

```
#include <iostream>
using namespace std;
```

Temat 2. Zmienne

Cel zajęć

Poznanie definicji zmiennych, rodzajów zmiennych występujących w języku programowania C++ oraz użycia strumieni przepływu danych do wyświetlania danych na monitorze oraz przypisywania wartości wprowadzanych z klawiatury do zmiennych.

Zmienna – jest to miejsce w programie, do przechowywania wartości, najczęściej liczbowych lub tekstowych. Posiada ona określoną nazwę. W programowaniu nazwa zmiennej powinna jak najlepiej określać to, czego dotyczy. W języku programowania C++ musimy określić z jakim typem zmiennych mamy do czynienia.

Podstawowe rodzaje zmiennych w C++:

int – zmienne całkowite (liczby od -2147483648 do 2147483647)

float – zmienne rzeczywiste (zmiennoprzecinkowe)

char – pojedynczy znak (np.: a)

string – (z ang. łańcuch) zmienne służące do przechowywania łańcuchów znaków

Strumień danych (operatory „>>” lub “<<”) w C ++ służą m. in. do przesyłania, wprowadzania, przypisywania oraz wyświetlania danych.

Do przesyłania danych będziemy używać dwóch głównych poleceń:

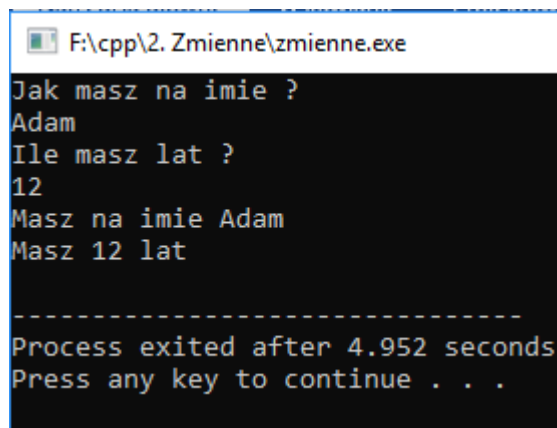
cout – przekierowuje strumień na monitor (wyświetlanie w konsoli)

cin – przekierowuje zewnętrzny strumień danych (np. ciąg znaków wprowadzony z klawiatury) do pamięci wewnętrznej (przypisanie do zmiennej).

Dodanie polecenia << **endl**; powoduje, że program w konsoli przechodzi do kolejnej linii.

Program z wykorzystaniem zmiennych

Poniżej efekt działania programu, którego skrypt znajduje się na kolejnej stronie.



```
F:\cpp\2. Zmienne\zmienne.exe
Jak masz na imie ?
Adam
Ile masz lat ?
12
Masz na imie Adam
Masz 12 lat

-----
Process exited after 4.952 seconds
Press any key to continue . . .
```

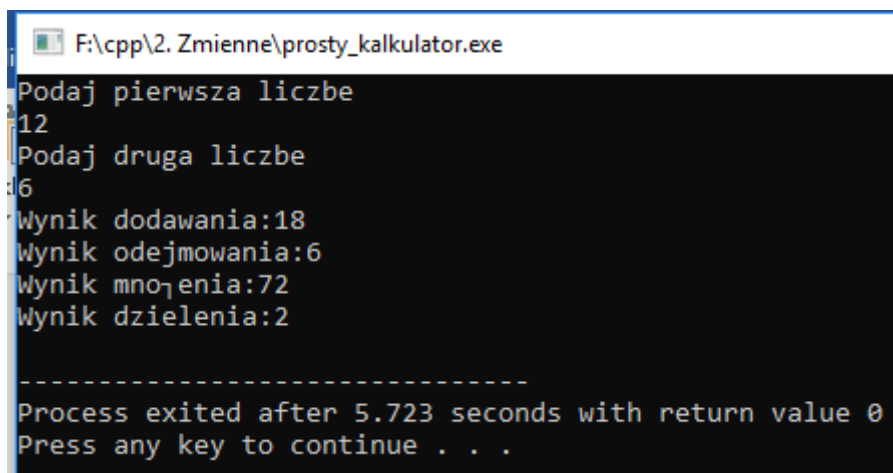
Skrypt programu:

```
zmienne.cpp
1  #include <iostream> //wczytanie biblioteki
2  using namespace std; //określenie przestrzeni nazw
3
4  string imie; //deklaracja zmiennej imie
5  int wiek;    //deklaracja zmiennej wiek
6
7  int main()
8  {
9      cout << "Jak masz na imie ?" << endl;
10     cin >> imie; //przypisanie ciągu znaków do zmiennej
11     cout << "Ile masz lat ? " << endl;
12     cin >> wiek; //przypisanie wprowadzonej liczby do zmiennej
13
14     cout << "Masz na imie ";
15     cout << imie << endl; //wyswietlenie zmiennej w konsoli
16     cout << "Masz ";
17     cout << wiek; //wyswietlenie zmiennej wiek w konsoli
18     cout << " lat" << endl;
19
20     return 0; // informacja o braku błędów
}
```

Pytanie: co się stanie jeżeli dla zmiennej „imie” wprowadzimy wartość liczbową? Co się stanie, jeżeli dla zmiennej „wiek” wprowadzimy ciąg znaków ?

Zadanie do samodzielnej realizacji

Napisz program, który umożliwi wprowadzenie dwóch liczb zmiennoprzecinkowych z klawiatury, a następnie wypisze na ekranie wynik dodawania, odejmowania, mnożenia oraz dzielenia tych dwóch liczb. Efekt działania programu, który chcemy otrzymać:



```
F:\cpp\2. Zmienne\prosty_kalkulator.exe
Podaj pierwsza liczbe
12
Podaj druga liczbe
6
Wynik dodawania:18
Wynik odejmowania:6
Wynik mnożenia:72
Wynik dzielenia:2

-----
Process exited after 5.723 seconds with return value 0
Press any key to continue . . .
```

Na kolejnej stronie znajdziemy przykładowy skrypt programu.

Przykładowy skrypt programu:

zmienne.cpp	prosty_kalkulator.cpp
1	<code>#include <iostream> //wczytanie biblioteki</code>
2	<code>using namespace std; //określenie przestrzeni nazw</code>
3	
4	<code>float a,b; //deklaracja zmiennych</code>
5	
6	<code>int main()</code>
7	<code>{ cout << "Podaj pierwsza liczbe" << endl;</code>
8	<code>cin >> a; //przypisanie wprowadzonej wartości do zmiennej a</code>
9	<code>cout << "Podaj druga liczbe" << endl;</code>
10	<code>cin >> b; //przypisanie wprowadzonej wartości do zmiennej b</code>
11	
12	<code>cout << "Wynik dodawania:"; </code>
13	<code>cout << a+b << endl; //wyswietlenie wyniku + w konsoli</code>
14	<code>cout << "Wynik odejmowania:";</code>
15	<code>cout << a-b << endl; ///wyswietlenie wyniku - w konsoli</code>
16	<code>cout << "Wynik mnożenia:";</code>
17	<code>cout << a*b << endl; ///wyswietlenie wyniku * w konsoli</code>
18	<code>cout << "Wynik dzielenia:";</code>
19	<code>cout << a/b << endl; ///wyswietlenie wyniku / w konsoli</code>
20	
21	<code>return 0; // informacja o braku błędów</code>
22	<code>}</code>

Możemy również przekazać na wyjście programu kilka wartości jednocześnie, zarówno ciągów znaków umieszczonych w cudzysłowie „,” oraz wartości zmiennych.

```
cout << "Wynik dodawania: " << a << " + " << b << " = ";
cout << a+b << endl; //wyswietlenie wyniku + w konsoli
cout << "Wynik odejmowania: " << a << " - " << b << " = ";
cout << a-b << endl; ///wyswietlenie wyniku - w konsoli
cout << "Wynik mnożenia: " << a << " x " << b << " = "; |
cout << a*b << endl; ///wyswietlenie wyniku * w konsoli
cout << "Wynik dzielenia: " << a << " : " << b << " = ";
cout << a/b << endl; ///wyswietlenie wyniku / w konsoli
```

```
F:\cpp\2. Zmienne\prosty_kalkulator_v2.exe
Podaj pierwsza liczbe
12
Podaj druga liczbe
6
Wynik dodawania: 12 + 6 = 18
Wynik odejmowania: 12 - 6 = 6
Wynik mnozenia: 12 x 6 = 72
Wynik dzielenia: 12 : 6 = 2
-----
Process exited after 3.107 seconds with return value 0
Press any key to continue . . .
```

Zadanie dodatkowe

Napisz programy liczące pola różnych figur geometrycznych na podstawie wprowadzanych danych z klawiatury.

Temat 3. Instrukcja warunkowa IF ... ELSE ...

Cel zajęć

Zapoznanie z instrukcją warunkową "IF ... ELSE ..." oraz jej wykorzystanie w programowaniu.

Podstawowe informacje na temat instrukcji warunkowej

```
if (czy pada deszcz?)  
{  
    zabierz parasol;  
}  
else  
{  
    nie zabieraj parasola;  
}
```

Po lewej stronie został przedstawiony prosty schemat instrukcji warunkowej. Warunek umieszczamy w nawiasie okrągłym „()”. Jeżeli warunek zostaje spełniony, wykonywane są instrukcje umieszczone w nawiasie klamrowym „{ }”. Funkcja „else” nie jest elementem obowiązkowym przy instrukcji warunkowej. Polecenia zawarte w nawiasie klamrowym zostaną wykonane w przypadku niespełnienia warunku umieszczonego w nawiasie okrągłym przy funkcji IF.

W C++ operator „=” służy to przypisywaniu wartości. Operatorem porównania dwóch liczb w C++ są dwa znaki równa się „==”.

Operatory porównania oraz spójniki logiczne w C++

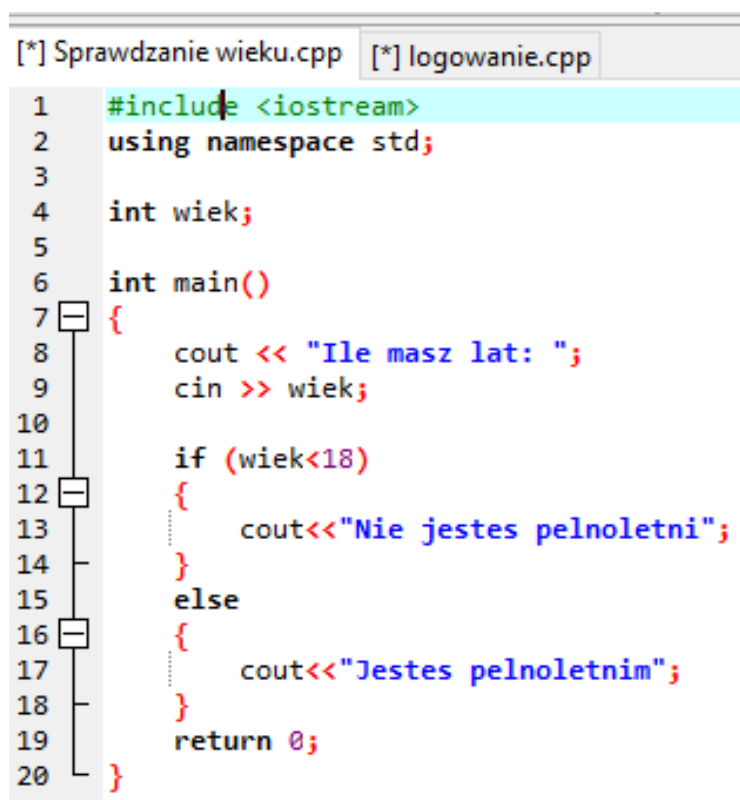
- "<" - mniejszy
- ">" - większy
- "<=" - mniejszy równy
- ">=" - większy równy
- "==" - równy
- "!=" - różny
- „&&” – spójnik logiczny „i” używany do łączenia warunków ze sobą; oba warunki muszą być spełnione
- „||” – spójnik logiczny „lub” używany do łączenia warunków ze sobą; wystarczy, że jeden z warunków jest spełniony,

W dalszej części skryptu dwa proste przykłady użycia instrukcji warunkowej if ... else

Program sprawdzający wiek

Przebieg programu:

1. Wczytujemy bibliotekę oraz określenie przestrzeni nazw std.
2. Deklarujemy zmienną *wiek* typu całkowitego (int - integer)
3. Wprowadzamy wartość z klawiatury
4. Zmienna *wiek* jest porównywana z liczbą 18.
5. Jeżeli wprowadzona przez nas liczba jest mniejsza od 18, na ekranie pojawi się napis, że jesteśmy niepełnoletni.
6. W przeciwnym wypadku wyświetlany jest napis, że jesteśmy pełnoletni.
7. Gdy wprowadzimy np. ciąg znaków, wtedy program napotka na błąd i przestanie działać.



```
[*] Sprawdzanie wieku.cpp  [*] logowanie.cpp
1  #include <iostream>
2  using namespace std;
3
4  int wiek;
5
6  int main()
7  {
8      cout << "Ile masz lat: ";
9      cin >> wiek;
10
11     if (wiek<18)
12     {
13         cout<<"Nie jestes pelnoletni";
14     }
15     else
16     {
17         cout<<"Jestes pelnoletnim";
18     }
19     return 0;
20 }
```

Logowania do konta

Tym razem wprowadzamy dwie zmienne typu string (*login* i *hasło*). Sprawdzane będą dwa warunki, aby zalogować się do konta musimy podać prawidłowy login oraz prawidłowe hasło. Oba warunki są połączone spójnikiem logicznym „i” oznaczanym w C++ jako „&&”. Przebieg programu jest taki sam jak powyżej. Po spełnieniu obu warunków w konsoli pojawi się informacja, że udało nam się zalogować do konta. W przypadku wprowadzenia błędnych danych na ekranie pojawi się informacja, że wprowadziliśmy błędny login lub hasło.

```

[*] Sprawdzanie wieku.cpp  [*] logowanie.cpp
1  #include <iostream>
2  using namespace std;
3
4  string login, haslo;
5
6  int main()
7  {
8      cout << "Podaj login: ";
9      cin >> login;
10     cout << "Podaj haslo: ";
11     cin >> haslo;
12
13     if ((login=="admin")&&(haslo=="haslo"))
14     {
15         cout<<"Udalo Ci sie zalogowac!";
16     }
17     else
18     {
19         cout<<"Podales niewlasciwy login lub haslo";
20     }
21     return 0;
22 }

```

Zadania do samodzielnej realizacji – wskaźnik BMI

Napisanie programu wyznaczającego Wskaźnik Masy Ciała BMI (*body mass index*) oraz wyświetlającego na ekranie informację czy waga naszego ciała jest w normie. Poniżej wzór na wartość BMI:

$$BMI = \frac{masa}{wzrost^2}$$

gdzie masa wyrażona jest w kilogramach, natomiast wzrost wyrażony jest w metrach. Dla większej dokładności wyników, niech wprowadzana wartość wzrostu wyrażona będzie w centymetrach. W stosowanej przez nas bibliotece nie ma funkcji podnoszącej liczbę do kwadratu, dlatego użyjemy mnożenia.

Zakresy wartości BMI:

- mniej niż 16 – wygłodzenie
- 16 - 16.99 – wychudzenie
- 17 - 18.49 – niedowaga
- 18.5 - 24.99 - wartość prawidłowa
- 25 - 29.99 – nadwaga
- 30 - 34.99 - I stopień otyłości
- 35 - 39.99 - II stopień otyłości
- powyżej 40 - III stopień otyłości

Prosty przykład programu uwzględniającego tylko trzy przedziały (uwaga zastosowano polecenie „else... if ...” które umożliwia sprawdzanie kolejnych warunków w przypadku niespełnienia pierwszej instrukcji warunkowej):

```
BMI.cpp
1  #include <iostream>
2  using namespace std;
3
4  float waga, wzrost, BMI;
5
6  int main()
7  {
8      cout << "Podaj swoja wage w kilogramach: ";
9      cin >> waga;
10     cout << "Podaj swoj wzrost w centymetrach: ";
11     cin >> wzrost;
12     BMI=waga/(wzrost*wzrost/10000);
13
14     if (BMI>=25)
15     {
16         cout<<"Twoje BMI wynosi " << BMI << " i jest powyzej optymalnej wartosci";
17     }
18     else if (BMI<18,5)
19     {
20         cout<<"Twoje BMI wynosi " << BMI << " i jest ponizej optymalnej wartosci";
21     }
22     else
23     {
24         cout<<"Twoje BMI wynosi " << BMI << " i jest w normie";
25     }
26     return 0;
27 }
```

*****Dodatkowo możemy określić jaki jest optymalny zakres wagi dla naszego wzrostu.**

Temat 4. Pętla FOR

Cel zajęć

Uczniowie poznają kolejny element programistyczny jakim jest pętla "FOR". Poznają również przykładowe zastosowania pętli.

Pętla jest jednym z podstawowych elementów programistycznych. Umożliwia ona cykliczne wykonywanie danych instrukcji określoną liczbę razy (pętla FOR) lub do momentu zajścia pewnych warunków (pętla WHILE i DO WHILE), które warunkują zakończenie pętli.

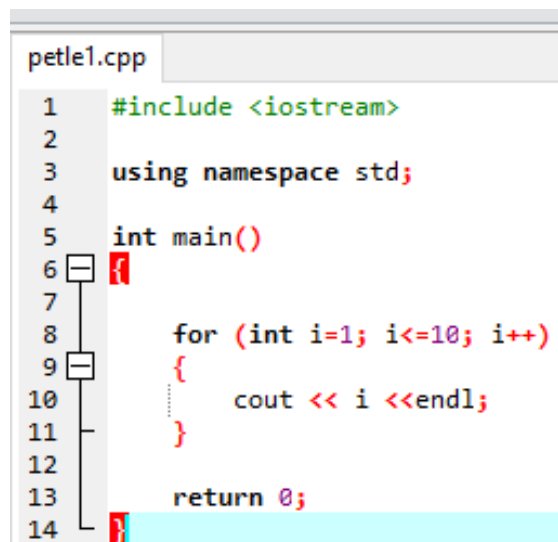
Kod programu, który ma być wykonywany w każdej kolejnej iteracji pętli FOR umieszczony jest w nawiasie klamrowym „{ }”.

Budowa pętli for jest podzielona na cztery części A, B, C oraz D.

FOR (A,B,C) {D} Są to:

- nadawanie początkowych wartości zmiennym (oznaczona jako „A”),
- warunek kończący pętlę for (oznaczony jako „B”),
- zwiększenie (zmniejszenie) licznika pętli (oznaczony jako „C”),
- powtarzana instrukcja (oznaczony jako „D”) bądź blok instrukcji.

Po prawej stronie przykład pętli wypisującej na ekranie kolejne liczby od 1 do 10, jedna pod drugą.



```
petle1.cpp
1  #include <iostream>
2
3  using namespace std;
4
5  int main()
6  {
7
8      for (int i=1; i<=10; i++)
9      {
10         cout << i << endl;
11     }
12
13     return 0;
14 }
```

Zapis „i++” oznacza to samo co zapis „i=i+1”. W tym wypadku w każdej kolejnej iteracji wartość zmiennej jest zwiększana o 1.

Jeżeli w pętli mamy do wykonania tylko jedną instrukcję zamiast zapisu:

```
for (int i=1; i<=10; i++)
{
    cout << i << endl;
}
```

możemy użyć zapisu bez nawiasu „{ }”:

```
for (int i=1; i<=10; i++)
    out << i << endl;
```

Jednakże, na wstępnym etapie programowania zalecane jest używanie nawiasów {}. Podobny skrócony zapis można użyć dla innych funkcji np. funkcji IF.

W niektórych wypadkach, może się tak zdarzyć, że warunek zakończenia pętli nigdy nie zostanie spełniony. W poniższym wypadku warunek „i<0” będzie spełniony w każdej iteracji. Liczby będą wypisywane w nieskończoność.

```
petle1.cpp [*] petle2.cpp
1  #include <iostream>
2
3  using namespace std;
4
5  int main()
6  {
7
8      for (int i=-1; i<0; i=i-1)
9      {
10         cout << i << endl;
11     }
12
13     return 0;
14 }
```

W kolejnym przykładzie został przedstawiony program wykorzystujący dwie dodatkowe biblioteki:

- <windows.h>, z którego pochodzi funkcja Sleep(1000) – program zostaje wstrzymany na określoną liczbę ms, w tym wypadku 1000ms,
- <cstdlib>, z którego pochodzi funkcja systemu ("cls") czyszcząca ekran.

Zamiast zapisu „i=i-1” został użyty zapis „i--”, który ma identyczne działanie.

Efektom działania poniższego programu będzie wyświetlanie na ekranie kolejnych liczb od 15 do 1.

Przed każdą kolejną iteracją ekran programu jest czyszczony – funkcja system("cls").

```
petle1.cpp [*] petle2.cpp [*] petle3.cpp
1  #include <iostream>
2  #include <windows.h>
3  #include <cstdlib>
4
5  using namespace std;
6
7  int main()
8  {
9
10     for (int i=15; i>=0; i--)
11     {
12         Sleep(1000);
13         system("cls");
14         cout << i << endl;
15     }
16     cout<<"KONIEC ODLICZANIA";
17     return 0;
18 }
```

Po ostatniej iteracji ekran zostaje również wyczyszczony i pojawia się napis „KONIEC ODLICZANIA”.

Temat 5. Pętle „while” i „do... while...”

Cel zajęć

Zapoznanie z pętlą „while” i „do ... while ...” oraz wykorzystanie ich w prostych programach.

Podstawowe informacje o pętli „while ...” i „do ... while ...”

Kolejne dwie pętle, które będą omawiane, są to tak zwane pętle warunkowe. Różnią się od siebie tym, że w przypadku pętli „while” warunek sprawdzany jest już na samym początku jej wykonywania. Może się zatem zdarzyć, że struktura nie zostanie wykonana ani razu. W przypadku pętli „do while” warunek jest sprawdzany po zakończeniu pętli. Pętla „do while” zostanie wykonana co najmniej jeden raz. Konstrukcja funkcji jest bardzo zbliżona do konstrukcji funkcji IF.

Instrukcja warunkowa	Pętla „while”	Pętla „do while”
if (warunek) { Instrukcja wykonana w przypadku spełnienia warunku; }	while (warunek) { powtarzana instrukcja ; }	do { powtarzana instrukcja ; } while (warunek)

Różnice najlepiej zobaczyć na prostym przykładzie z bakteriami. W każdej godzinie populacja bakterii rośnie dwukrotnie. Warunkiem zakończenia pętli jest stan populacji mniejszy, bądź równy 1000000000 (miliard).

Jaki będzie ostatni wynik w przypadku uruchomienia pętli „while” oraz „do while”?

Przykład programu z wykorzystaniem pętli „while ...”

```
1  #include <iostream>
2
3  using namespace std;
4
5  int populacja=1; int godzin=0;
6
7  int main()
8  {
9      while (populacja <= 1000000000)
10     {
11         godzin++;
12         populacja = populacja*2;
13
14         cout<<"minelo godzin: "<<godzin;
15         cout<<" liczba bakterii: "<<populacja<<endl;
16     }
17     return 0;
18 }
```

Przykład programu z wykorzystaniem pętli „do ... while ...”

```
1  #include <iostream>
2
3  using namespace std;
4
5  int populacja=1; int godzin=0;
6
7  int main()
8  {
9      do
10     {
11         godzin++;
12         populacja = populacja*2;
13
14         cout<<"minelo godzin: "<<godzin;
15         cout<<" liczba bakterii: "<<populacja<<endl;
16     }while (populacja <= 1000000000);
17
18     return 0;
19 }
```

Wyniki wykonania obu programów będą różne.

Zgadnij o jakiej liczbie myślę

W kolejnym projekcie wykorzystamy pętlę warunkową do stworzenia mini gry. Naszym zadaniem będzie odgadnięcie liczby wylosowanej przez komputer z zakresu od 1 do 100.

W C++ funkcja „**rand() % 100 + 1**” zwróci losową liczbę z zakresu od 1 do 100. Funkcja ta znajduje się w bibliotece: **<stdlib.h>**.

Program będzie przebiegał w następujący sposób:

1. Wartość zmiennej próba zostaje ustawiona na 0.
2. Komputer losuje liczbę z zakresu od 1 do 100, przypisując ją do zmiennej liczba,
3. Wartość zmiennej próba zostaje zwiększona o 1.
4. Na ekranie pojawia się komunikat proszący nas o wpisanie liczby z zakresu od 1 do 100, zostaje ona przypisana jako wartość zmiennej odpowiedz.
5. Jeżeli wartość zmiennej odpowiedz jest większa od zmiennej liczba na ekranie pojawia się komunikat: „Za dużo, spróbuj jeszcze raz”.
6. Jeżeli wartość zmiennej odpowiedz jest mniejsza od zmiennej liczba na ekranie pojawia się komunikat: „Za mało, spróbuj jeszcze raz”.
7. Jeżeli wartość zmiennej odpowiedz będzie równa zmiennej liczba zakończ działanie pętli, w przeciwnym wypadku wróć do kroku 3.
8. Po zakończeniu pętli program poinformuje nas za którą próbą wprowadziliśmy poprawną wartość.

```

Zgadnij jaka mysle liczbe? 50
Za duzo, sprobuj jeszcze raz
Zgadnij jaka mysle liczbe? 25
Za malo, sprobuj jeszcze raz
Zgadnij jaka mysle liczbe? 37
Za malo, sprobuj jeszcze raz
Zgadnij jaka mysle liczbe? 45
Za duzo, sprobuj jeszcze raz
Zgadnij jaka mysle liczbe? 40
Za malo, sprobuj jeszcze raz
Zgadnij jaka mysle liczbe? 43
Za duzo, sprobuj jeszcze raz
Zgadnij jaka mysle liczbe? 42
Brawo udalo ci sie za 7 razem
-----
Process exited after 40.31 seconds with return value 0
Press any key to continue . . .

```

```

1  #include <iostream>
2  #include <stdlib.h>    /* rand */
3  using namespace std;
4  int liczba, odpowiedz=0, proba=0 ;
5
6  int main()
7  {
8      liczba = rand() % 100 + 1; //losowanie liczb z przedalu od 1 do 100
9
10     while (liczba!=odpowiedz)
11     {
12         proba=proba+1;
13         cout<<"Zgadnij jaka mysle liczbe? ";
14         cin >> odpowiedz;
15
16         if (odpowiedz>liczba)
17         {
18             cout<<"Za duzo, sprobuj jeszcze raz " << endl;
19         }
20         else if (odpowiedz<liczba)
21         {
22             cout<<"Za malo, sprobuj jeszcze raz" << endl;
23         }
24     }
25     cout<<"Brawo udalo ci sie za " << proba << " razem";
26     return 0;
27 }

```

Zadania dodatkowe

Dodanie opcji pozwalającej na wprowadzenie przez użytkownika innego zakresu losowanych liczb niż od 1 do 100.

Narysuj schemat blokowy działania programu.

Temat 6. Kalkulator IF

Cel zajęć

Wykorzystanie dotychczas zdobytej wiedzy do wykonania kalkulatora tekstowego z użyciem funkcji „if ...”. Są to zajęcia powtórzeniowe z wykorzystania instrukcji warunkowej „if ... elseif... else ...”.

Poniżej został przedstawiony krokowy przebieg programu, spróbuj napisać program samodzielnie bez zaglądania na kolejną stronę skryptu, gdzie znajduje się przykładowy program. Możesz skorzystać z ostatniego skryptu z drugich zajęć, gdzie program zwracał wyniki wszystkich czterech działań matematycznych.

Do prawidłowego działania programu wystarczy nam podstawowa biblioteka **<iostream>**, dwie zmienne typu float (*a* i *b* - zmienne rzeczywiste) oraz jedna zmienna typu int (*działanie* – zmienna całkowita).

Krokowy przebieg programu

Krok 1. Wprowadzenie dwóch liczb z klawiatury oraz przypisanie ich do zmiennych *a* i *b*.

Krok 2. Wyświetlenie na ekranie Menu programu, wskazującego, który numer wprowadzony z klawiatury, odpowiada za wykonanie konkretnego działania.

Krok 3. Wybór przy pomocy klawiatury numeru działania oraz przypisanie go do wartości zmiennej *odpowiedz*.

Krok 4. Sprawdzenie przy pomocy instrukcji warunkowej IF, które działanie zostało wybrane oraz wypisanie wyniku działania na ekranie.

Poniżej przebieg wykonania całego programu. Pamiętajmy o wprowadzeniu warunku uniemożliwiającego dzielenie przez 0.

```
Podaj pierwsza liczbe: 12
Podaj druga liczbe: 1.23

WYBOR DZIALANIA
-----
1. Dodawanie
2. Odejmowanie
3. Mnozenie
4. Dzielenie
Wybierz: 1
12 + 1.23 = 13.23
-----
Process exited after 11.3 seconds with return value 0
Press any key to continue . . .
```

Separatorem miejsc dziesiętnych w C++ jest kropka „.”.

Przykładowy program kalkulatora z użyciem instrukcji warunkowej IF

```
1  #include <iostream>
2  using namespace std;
3
4  float a,b;
5  int dzialanie;
6
7  int main()
8  {
9      cout << "Podaj pierwsza liczbe: ";
10     cin >> a;
11     cout << "Podaj druga liczbe: ";
12     cin >> b;
13
14     cout<<endl;
15     cout << "WYBOR DZIALANIA" << endl;
16     cout << "-----" << endl;
17     cout << "1. Dodawanie" << endl;
18     cout << "2. Odejmowanie" << endl;
19     cout << "3. Mnozenie" << endl;
20     cout << "4. Dzielenie" << endl;
21
22     cout << "Wybierz: ";
23     cin >> dzialanie;
24
25     if (dzialanie==1)
26         cout<< a << " + " << b << " = " <<a+b;
27
28     else if (dzialanie==2)
29         cout<< a << " - " << b << " = " <<a-b;
30
31     else if (dzialanie==3)
32         cout<< a << " X " << b << " = " <<a*b;
33
34     else if (dzialanie==4)
35         if (b==0) cout << "Nie dzielimy przez zero!";
36         else cout<< a << " : " << b << " = " <<a/b;
37
38     else cout<<"Nie ma takiej opcji w menu!";
39
40     return 0;
41 }
```

Temat 7. Tabliczka mnożenia

Cel zajęć

Wykonanie programu, który będzie sprawdzał naszą wiedzę z tabliczki mnożenia. W czasie tych zajęć zostanie wykorzystana dotychczas zdobyta wiedza na temat pętli oraz instrukcji warunkowych.

Program będzie dziesięciokrotnie losował dwie liczby z przedziału od 1 do 10 („`rand() % 10 + 1`”). Całość zostanie umieszczona w pętli iteracyjnej „`for (int i=1; i<=10; i++) { }`”. Instrukcje umieszczone wewnątrz pętli zostaną wykonane 10 razy.

Wewnątrz pętli iteracyjnej zostanie umieszczona pętla warunkowa „`while (iloczyn != odpowiedz) { }`” powtarzana tak długo, dopóki nie wprowadzimy poprawnej odpowiedzi. Pętla zostaje ustawiona w taki sposób, aby została wykonana co najmniej raz.

Krokowy przebieg programu

Krok 1. Ustaw wartość zmiennej *punkty* na 0

Krok 2. Ustaw zmienną *i* na 1

Krok 3. Wylosuj dwie liczby z zakresu od 1 do 10

Krok 4. Przypisz je do zmiennych *czynniki1* oraz *czynniki2*

Krok 5. Ustaw wartość zmiennej *iloczyn* na wynik mnożenia zmiennych *czynniki1* i *czynniki2*

Krok 6. Ustaw wartość zmiennej *odpowiedz* na 0

Krok 7. Ustaw wartość zmiennej *proba* na 0

Krok 8. Zmień wartość zmiennej *proba* o 1

Krok 9. Wyświetl na ekranie działanie, którego wynik mamy podać

Krok 10. Wprowadź wynik działania przy pomocy klawiatury i przypisz tę wartość do zmiennej *odpowiedz*

Krok 11. Porównaj wartość zmiennej *odpowiedz* z wartością zmiennej *iloczyn*

Krok 12. Jeżeli zmienna *odpowiedz* nie będzie równa wartości zmiennej *iloczyn*, zmień wartość zmiennej *punkty* o -1 i wróć do Kroku 8.

Krok 13. Zmień wartość zmiennej *punkty* o 1 oraz poinformuj, za którym razem wprowadziliśmy poprawną wartość (wypisz wartość zmiennej *proba*)

Krok 14. Zwiększ zmienną *i* o 1, jeżeli $i \leq 10$ wróć do Kroku 3

Krok 15. Wypisz na ekranie wartość zmiennej *punkty*.

Przykładowy skrypt programu znajduje się na kolejnej stronie. Pamiętajmy o użyciu dodatkowej biblioteki `<stdlib.h>` w którym znajduje się funkcja `rand()`.

Przykładowy program „Tabliczka mnożenia”

```
1  #include <iostream>
2  #include <stdlib.h>      /* srand, rand */
3  using namespace std;
4  int czynnik1, czynnik2, iloczyn, odpowiedz, proba, punkty=0;
5
6  int main()
7  {
8      cout<<"Przed Toba 10 zadan z Tabliczki mnozenia" << endl;
9      for (int i=1; i<=10; i++)
10     {
11         czynnik1 = rand() % 10 + 1;
12         czynnik2 = rand() % 10 + 1;
13         iloczyn=czynnik1*czynnik2;
14         proba=0;
15         odpowiedz=0;
16         while (iloczyn!=odpowiedz)
17         {
18             proba=proba+1;
19             cout<<"Ile to jest: " << czynnik1 << " X " << czynnik2 << " = ?    ";
20             cin >> odpowiedz;
21
22             if (odpowiedz>iloczyn)
23             {
24                 cout<<"Za duzo, sprobuj jeszcze raz " << endl;
25                 punkty=punkty-1;
26             }
27             else if (odpowiedz<iloczyn)
28             {
29                 cout<<"Za malo, sprobuj jeszcze raz" << endl;
30                 punkty=punkty-1;
31             }
32         }
33         cout<<"Brawo udalo ci sie za " << proba << " razem" << endl;
34         punkty=punkty+1;
35     }
36     cout<<"Zdobyles lacznie " << punkty << " punktow" << endl;
37
38     return 0;
39 }
```

Zadania dodatkowe

Dodanie opcji pozwalającej wybrać zakres losowanych liczb.

Modyfikacja programu w taki sposób, żeby umożliwiał również ćwiczenie zadań z dzielenia liczb.
Podpowiedź:

```
while (czynnik2 != odpowiedz) {}
cout << "ile to jest: " << iloczyn << " / " << czynnik1 << " = ?    ";
if (odpowiedz > iloczyn)
```

Temat 8. Funkcja SWITCH – Menu

Cel zajęć

Zapoznanie się z funkcją wyboru „SWITCH” oraz jej wykorzystanie do wykonania prostego menu, kalkulatora oraz programu określającego liczbę dni w danym miesiącu.

Opis funkcji SWITCH

Inną opcją określania wyboru w C++ jest funkcja **SWITCH** (z ang. przełącznik). Jest to tzw. „**funkcja wielokrotnego wyboru**”. Działanie tej funkcji jest zupełnie inne, niż działanie funkcji **IF**.

Do wykonania danej instrukcji IF mogliśmy dokładnie określić jakie warunki muszą zajść, żeby dane zdarzenie miało miejsce. W przypadku **SWITCHA** nie możemy łączyć warunków ze sobą, możemy wykonywać decyzje tylko i wyłącznie na podstawie wartości jednej zmiennej, która musi być typu podstawowego (głównie całkowitego).

O ile możliwości instrukcji wielokrotnego wyboru są znacznie mniejsze od instrukcji IF, to umożliwia ona w niektórych przypadkach zwiększenie szybkości działania programu lub estetyki kodu.

Poniżej przykładowy program z menu, który może zostać wykorzystany przy realizacji kolejnych projektów.

Instrukcja ta zawiera kilka słów kluczowych:

„**case 1:**” – odpowiada za wybór poszczególnych opcji,

„**break;**” - po wykonaniu instrukcji zawartej w danej opcji program kończy działanie instrukcji wyboru,

„**default:**” – jeżeli żaden warunek nie zostanie spełniony lub program nie będzie zawierał instrukcji **break;** wykonają się instrukcje umieszczone po tym słowie kluczowym.

Proste menu programu

```
1  #include <iostream>
2  using namespace std;
3
4  int opcja;
5
6  int main()
7  {
8      cout << "Menu Programu - " << endl;      //Wyswietlenie menu
9      cout << "-----" << endl;      //nowa linia <<endl
10     cout << "1. Pierwsza opcja" << endl;
11     cout << "2. Druga opcja" << endl;
12     cout << "3. Trzecia opcja" << endl;
13     cin >> opcja;
14
15     switch(opcja)      // uruchomienie switcha z parametrem wywołania
16     {
17         case 1:
18             cout << "Wybrales pierwsza opcje" << endl;
19             break;      // break; powoduje zakonczenie dzialania switcha
20
21         case 2:
22             cout << "Wybrales druga opcje" << endl;
23             break;
24
25         case 3:
26             {
27                 cout << "Wybrales druga opcje" << endl;
28             }
29             break;
30         default: cout<<"Niepoprawny wybor opcji";
31     }
32     return 0;
33 }
```

* Kalkulator z wykorzystaniem Switcha

Poniżej kod programu dający taki sam efekt działania, jak program wykonany w czasie 6 zajęć. W poniższym przypadku całość została umieszczona w pętli „**for (;;) {}**”, która będzie działała w nieskończoność. W celu opuszczenia pętli został dodana dodatkowa opcja „5. Zakończ działanie programu”. Po wprowadzaniu z klawiatury cyfry 5 zostanie wykonana instrukcja warunkowa **exit(0)**; (program się zakończy).

```

1  #include <iostream>
2  using namespace std;
3
4  float a,b; // określenie typu zmiennych float - UWAGA separatorem w C++ jest kropka "."
5  char dzialanie; //dzialanie jest to pojedynczy znak wprowadzany z klawiatury
6
7  int main()
8  {
9      for(;;) // taki zapis petli for spowoduje, że będzie działał w nieskończoność
10     {
11         cout << "Podaj pierwsza liczbe: ";
12         cin >> a; // przypisanie wprowadzonej liczby do zmiennej a
13         cout << "Podaj druga liczbe: ";
14         cin >> b; // przypisanie wprowadzonej liczby do zmiennej b
15
16         cout<<endl;
17         cout << "WYBOR DZIALANIA" << endl; //Wyswietlenie menu, kolejne linie tekstu
18         cout << "-----" << endl; //pojawiają się w nowym wierszu <<endl
19         cout << "1. Dodawanie" << endl;
20         cout << "2. Odejmowanie" << endl;
21         cout << "3. Mnozenie" << endl;
22         cout << "4. Dzielenie" << endl;
23         cout << "5. Zakonczone dzialanie programu" << endl;
24         cin >> dzialanie; //przypisanie wprowadzonego znaku do zmiennej dzialanie
25         cout << endl;
26
27         switch(dzialanie) //uruchomienie switcha
28         {
29             case '1':
30                 cout<< a << " + " << b << " = " <<a+b;
31                 break;
32             case '2':
33                 cout<< a << " - " << b << " = " <<a-b;
34                 break;
35             case '3':
36                 cout<< a << " X " << b << " = " <<a*b;
37                 break;
38             case '4':
39                 if (b==0) cout << "Nie dzielimy przez zero!";
40                 else cout<< a << " : " << b << " = " <<a/b;
41                 break;
42             case '5':
43                 exit(0);
44                 break;
45
46             default: cout<<"Nie ma takiej opcji w menu!";
47         }
48         getch();getch();
49         system("cls");
50     }
51     return 0;
52 }

```

*Określenie liczby dni w danym miesiącu

Poniżej dodatkowy program z wykorzystaniem switcha wyświetlający liczbę dni w danym miesiącu. W przypadku 2 miesiąca – lutego liczba dni może wynosić 28 lub 29, wszystko zależy w jakim roku wypada:

- co 4 lata, rok dzieli się bez reszty przez 4 – warunek **(rok%4==0)**
- jednak od tej reguły są wyjątki raz na 100 pełnych lat miesiąc przestępny nie jest dodawany, gdy dany rok dzieli się bez reszty przez 100nie ma roku przestępnego – warunek **(rok%100!=0)**
- chyba, że dany rok dzieli się przez 400 lata, w takim wypadku mamy do czynienia z rokiem przestępnym, gdy rok dzieli się bez reszty przez 400 – warunek **(rok%400==0)**

Połączenie wszystkich warunków

[(rok%4==0) i (rok%100!=0)] lub (rok%400==0)

```
1  #include <iostream>
2
3  using namespace std;
4  int numer_miesiaca;
5
6  int main()
7  {
8      cout << "Podaj numer miesiaca: ";
9      cin >> numer_miesiaca;
10
11     switch(numer_miesiaca)
12     {
13         case 1:
14         case 3:
15         case 5:
16         case 7:
17         case 8:
18         case 10:
19         case 12:
20             cout<<"Ten miesiac ma 31 dni!";
21             break;
22
23         case 4:
24         case 6:
25         case 9:
26         case 11:
27             cout<<"Ten miesiac ma 30 dni!";
28             break;
29
30         case 2:
31             {
32                 int rok;
33                 cout<<"Podaj rok: ";
34                 cin >> rok;
35
36                 if (((rok%4 == 0) && (rok%100 != 0)) || (rok%400 == 0))
37                     cout<<"Ten miesiac ma 29 dni!";
38                 else cout<<"Ten miesiac ma 28 dni!";
39             }
40             break;
41         default: cout<<"Niepoprawny numer miesiaca!";
42     }
43
44     return 0;
45 }
```

Temat 9. Znajdowanie maksimum

Cel zajęć

Napisanie programu znajdującego maksymalną wartość spośród wprowadzanych liczb. Uczniowie dowiedzą się na czym polega optymalizacja kodu. Czasami znalezienie najbardziej optymalnego rozwiązania jest bardzo trudne. Uczniowie poznają dwa różne przykłady rozwiązania tego samego problemu związanego ze znajdowaniem maksymalnej wartości liczbowej.

Wykonanie programu porównującego trzy liczby

Zacznijmy od samodzielnego napisania programu, który będzie porównywał 3 liczby wprowadzone z klawiatury.

Poniżej schemat krokowy programu.

Krok 1. Wprowadzenie zmiennych.

Krok 2. Porównanie pierwszej zmiennej z pozostałymi dwiema, jeżeli jest ona większa bądź równa od pozostałych dwóch wartości, zmienna ta zostaje wypisana na ekranie.

Krok 3. Jeżeli warunek z kroku 2 nie jest spełniony, to druga zmienna jest porównywana z pozostałymi dwiema zmiennymi. Gdy ta jest większa bądź równa od pozostałych dwóch, to zmienna ta zostaje wypisana na ekranie.

Krok 4. Jeżeli warunek z kroku 2 i 3 nie jest spełniony to trzecia zmienna jest porównywana z pozostałymi dwiema zmiennymi, jeżeli jest ona większa bądź równa od pozostałym dwóm wartości, to zmienna ta zostaje wypisana na ekranie.

Przykładowy program

```
1  #include <iostream>
2
3  using namespace std;
4
5  int a,b,c;
6
7  int main()
8  {
9      cout << "Podaj 3 liczby rozdzielone spacja: ";
10     cin>>a>>b>>c;
11
12     if ((a>=b) && (a>=c))
13         cout<<"Najwieksza liczba to "<<a;
14
15     else if ((b>=a) && (b>=c))
16         cout<<"Najwieksza liczba to "<<b;
17
18     else if ((c>=a) && (c>=b))
19         cout<<"Najwieksza liczba to "<<c;
20
21     return 0;
22 }
```

Program wydaje się prosty, ale co w przypadku chęci porównania ze sobą znacznie większej liczby zmiennych? Objętość programu będzie przyrastać bardzo szybko.

Zastanówmy się w jaki sposób można uprościć ten program?

W celu uproszczenia programu możemy wprowadzić zmienną pomocniczą m , do której zostaje przypisana wartość pierwszej zmiennej. Każda kolejna zmienna jest porównywana ze zmienną m , jeżeli wartość danej zmiennej jest większa od zmiennej m , to zmienna m zostaje ustawiona na wartość zmiennej z którą była porównywana. Ta sama procedura jest powtarzana dla wszystkich wprowadzonych zmiennych.

Poniżej przykładowy program. Został w nim zastosowany uproszczony zapis dla instrukcji warunkowej „if ...” bez użycia nawiasów klamrowych:

if (warunek) instrukcja;

Druga wersja programu z wykorzystaniem zmiennej pomocniczej

```
1  #include <iostream>
2  using namespace std;
3
4  int a,b,c,d,e,f,m;
5
6  int main()
7  {
8      cout << "Podaj 6 liczby rozdzielone spacja: ";
9      cin>>a>>b>>c>>d>>e>>f;
10
11     m=a;
12     if (b>m) m=b;
13     if (c>m) m=c;
14     if (d>m) m=d;
15     if (e>m) m=e;
16     if (f>m) m=f;
17     cout<<"Najwieksza liczba to "<<m;
18
19     return 0;
20 }
```

Temat 10. Sortowanie bąbelkowe

Cel zajęć

Zapoznanie z mechanizmem sortowania bąbelkowego.

Sortowanie bąbelkowe (ang. bubble sort) – prosta metoda sortowania. Polega ona na porównywaniu dwóch kolejnych elementów i zamianie ich kolejności, jeżeli element z lewej strony jest większy od tego z prawej. Sortowanie kończy się, gdy podczas kolejnego przejścia nie dokonano żadnej zmiany.

Sortowane elementy umieszczamy w **Tablicy**. Tablica jednowymiarowa jest to struktura składająca się z kilkunastu elementów. Kolejne elementy w tablicy w C++ są numerowane od 0 (w niektórych językach programowania mogą być indeksowane od 1).

Do zamiany elementów w tablicy użyjemy funkcji **swap(a,b)**. Działa w taki sposób, że zamienia dwie wartości użyte jako parametry tej funkcji.

Przykład uruchomienia funkcji swap(a,b) dla a = 120 oraz b = 100:

- zmienna „a” zostaje zapisana do tymczasowej zmiennej „t”: $t=a$,
- zmienna „a” przyjmuje wartość zmiennej „b”: $a=b$,
- zmienna „b” przyjmuje wartość tymczasowej zmiennej „t”: $b=t$.

Po uruchomieniu funkcji $a=100$, natomiast $b=120$.

Kod programu

int tablica [10] – tworzy tablicę 10 elementową, tablica zawiera 10 nieposortowanych elementów

Poniżej kod programu, został w nim użyta pętla „do ... while ()” sprawdzająca, czy w kolejnym przebiegu programu miało miejsce zamienienie kolejnością, któreś pary liczb.

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int tablica[10] = {10, 9, 8, 7, 6, 5, 4, 3, 2, 1}; //stworzenie zmiennej tablicowej
7      int stop=0;
8
9      do
10     {
11         stop=1; // użycie zmiennej jako pola znacznikowego
12         for (int j=0; j<9; j++)
13             if (tablica[j]>tablica[j+1])
14             {
15                 swap(tablica[j], tablica[j+1]); // funkcja swap() zamienia miejscami
16                 stop=0; // dwa elementy w tablicy, jeżeli element j
17                     // jest większy od kolejnego elementu
18             } while (stop==0); // jeżeli nie miała miejsca żadna zamiana miejscami pętla
19                     // się kończy (stop=1)
20         for (int i = 0; i<10; i++)
21             cout << tablica[i] << " "; // funkcja cout nie potrafi wyświetlić wartości tablicowych
22                     // dlatego każdy element tablicy wyświetlany jest oddzielnie
23         return 0;
24     }
```

Temat 11. Borland C++ Builder instalacja

Licencja **Borland C++ Buildera 6 Personal Edition** jest typu „free for non commercial use” (wersja Personal do użytku własnego oraz nauki). Rozmiar archiwum: 97 MB.

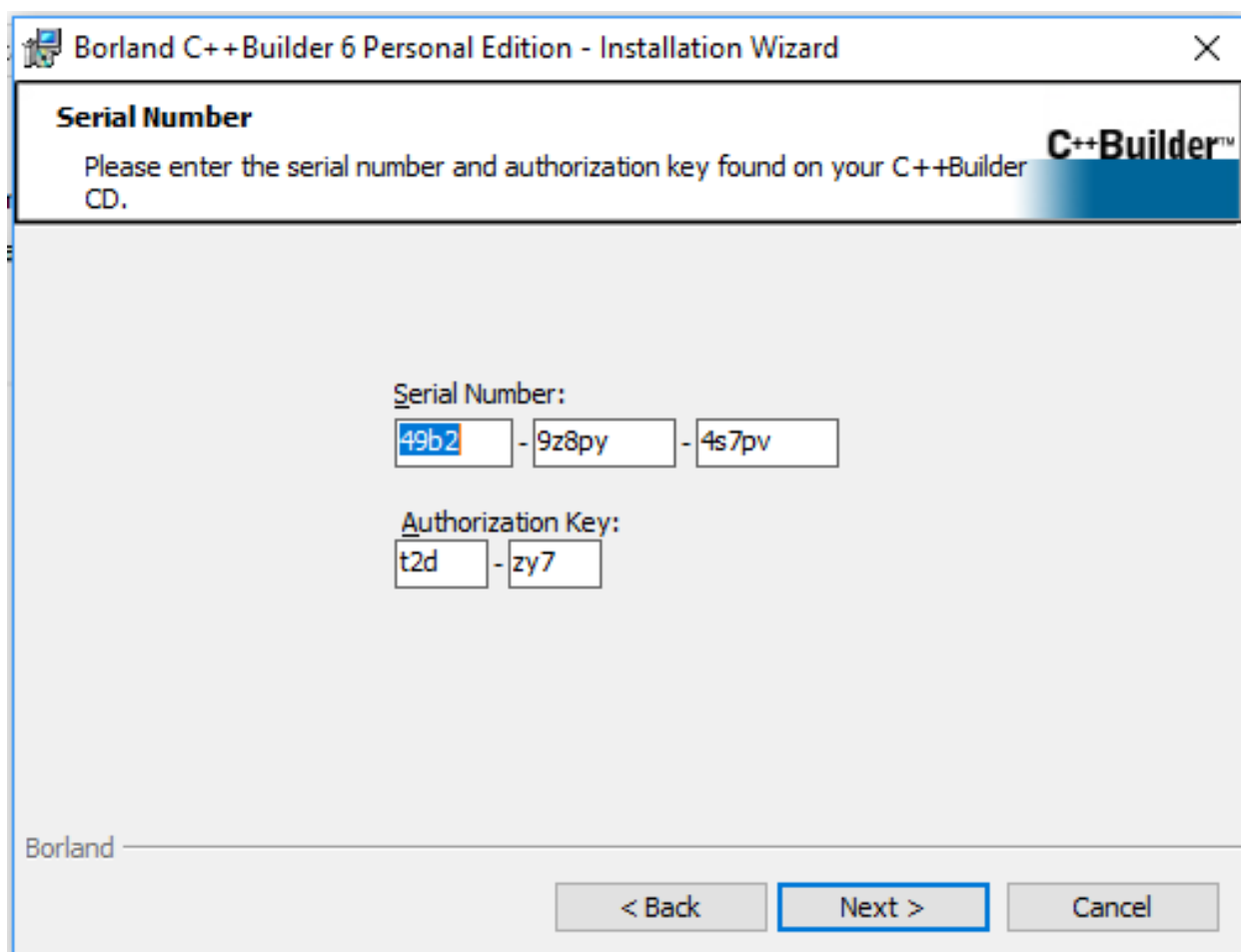
Klucz publiczny pochodzący z oficjalnej dystrybucji firmy Borland:

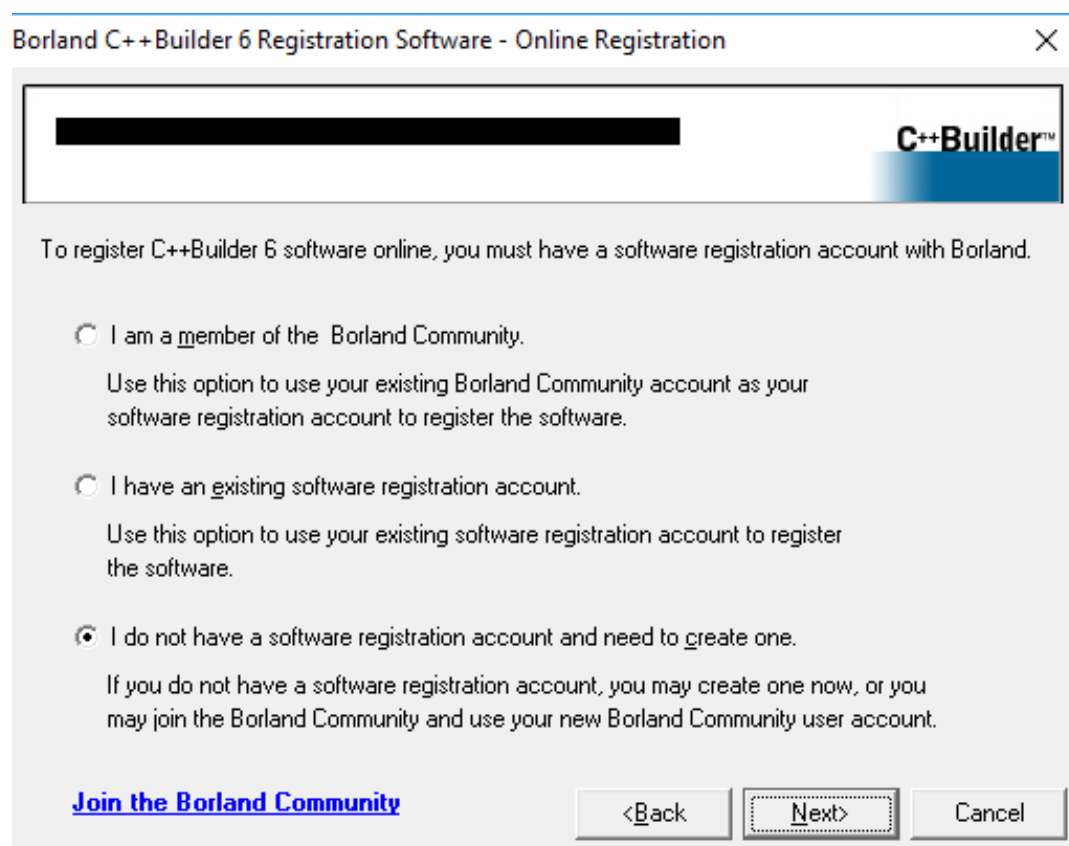
Borland - Serial Number: 49b2-9z8py-4s7pv

Authorization Key: t2d-zy7

Po rejestracji na stronie stajemy się pełnoprawnymi użytkownikami programu (program w celach testowych można uruchomić bez konieczności zakładania konta na stronie internetowej). Wersja testowa jest ograniczona 30 dniowym okresem próbnym. Jeśli nie zarejestrujesz pakietu, to wciąż będziesz mógł korzystać ze wszystkich jego funkcji.

Do rejestracji niezbędne jest połączenie z Internetem.





Temat 12. Borland C++ Builder – Hello World!

Cel zajęć

Zapoznanie się z podstawowymi funkcjami programu Borland C++ Builder 6. Wykonanie pierwszej aplikacji okienkowej.

Opis podstawowych elementów narzędzia Borland C++ Builder

Borland C++ Builder jest narzędziem umożliwiającym w szybki sposób tworzenia prostych aplikacji okienkowych. Dana aplikacja będzie składała się z kilku obiektów (niezależnych elementów) posiadających określone atrybuty (cechy). W programie każdy z obiektów będzie posiadał niezależnie wykonywane skrypt.

Po uruchomieniu programu pojawia się okno projektu „Form1”, stanowi ono „tło” naszej aplikacji. „Form1” jest podstawowym typem obiektu występującym w Borlandzie. Z lewej strony znajduje się „Object Inventor” – w zakładce „Properties” znajduje się szereg ustawień dotyczących wizualizacji danego elementu - są to **atrybuty** tego obiektu.

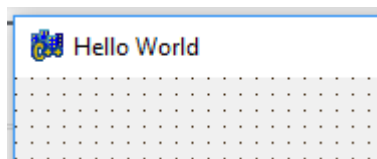
W zakładce „Events” znajdują się instrukcje, które mają być wykonane przy zajściu jakiegoś określonego zdarzenia. W przypadku zajścia któregoś może dojść do zmiany atrybutu określonego obiektu (np. po kliknięciu przycisku dojdzie do zmiany atrybutu odpowiedzialnego za widoczność danego obiektu „Label1 -> Visible = true/false”).

Wykonanie elementów graficznych programu

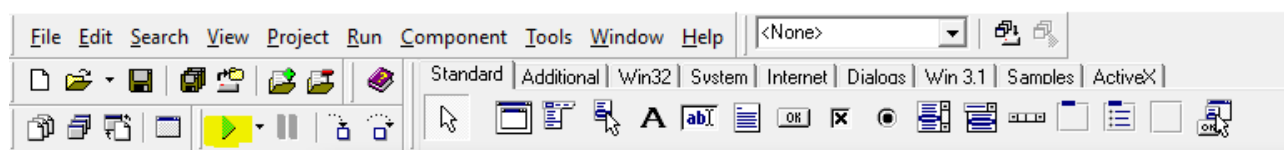
1. Dla „Form1” w zakładce „Properties” w polu Caption wpisujemy Hello World. Nazwa ta będzie wyświetlana jako tytuł naszej aplikacji. W tym przypadku Caption jest atrybutem obiektu Form1, natomiast „Hello World” jest wartością tego atrybutu.

Poniżej możemy ustawić wysokość okna „ClientHeight” na 250 oraz jego szerokość „ClientWidth” na 320.

Caption	Hello World
ClientHeight	250
ClientWidth	320





2. Program uruchamiany jest przy pomocy przycisku:

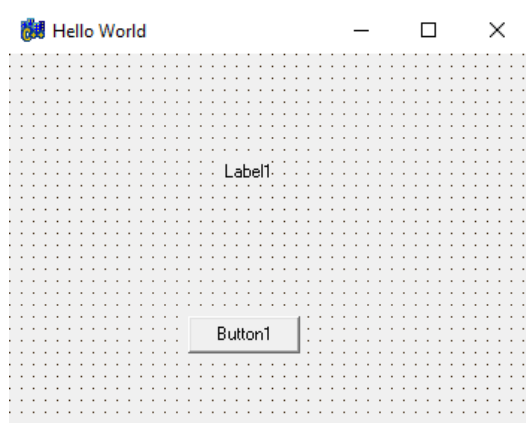


W tej chwili program automatycznie się kończy.

3. Kolejne obiekty (elementy graficzne programu) dodajemy bezpośrednio z górnej wstążki. Obiekt

dodajemy poprzez jego kliknięcie na górnej wstążce, a następnie klikamy na polu naszej aplikacji. Możemy go również dodać automatycznie poprzez dwukrotne kliknięcie lewym przyciskiem myszy na ikonę danego obiektu na górnej wstążce.

Do naszej aplikacji dodajemy przycisk „Button”  znajdziemy go w zakładce Standard oraz napis „Label”  z tej samej zakładki.



Po kliknięciu w dany obiekt możemy zmienić ich właściwości w zakładce „Properties” w oknie „Object Inspector”.

4. Klikamy lewym przyciskiem myszy na obiekt „Label1”, po czym zmieniam następujące ustawienia „Properties” na:

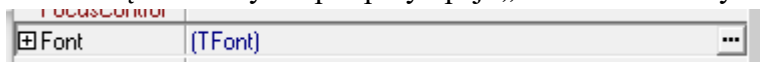
Caption = *Hello World!*

Height = **37**

Width = **175**

Visible = false

Po kliknięciu w trzy kropki przy opcji „Font” ustawiamy rozmiar czcionki „Size” na 24:



Ustawienie atrybutu Visible na wartość false spowoduje, że dany obiekt jest niewidoczny po uruchomieniu programu.

5. Dla obiektu „Button1”, zmieniamy następujące ustawienia „Properties” na:

Caption = *Kliknij mnie!*

Height = **80**

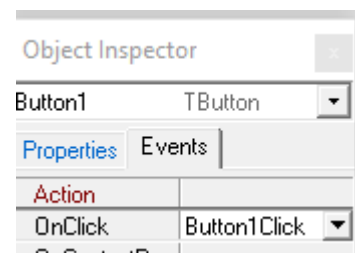
Width = **250**

Zmieniamy również rozmiar czcionki „Size” na 24

Program przycisku Button1

Po dwukrotnym kliknięciu „Button1” możemy zmodyfikować skrypt, który będzie wykonywany po kliknięciu w przycisk w trakcie działania aplikacji. Innym sposobem na dodanie danego zdarzenia jest jego wybór w zakładce „Events” dla danego obiektu.

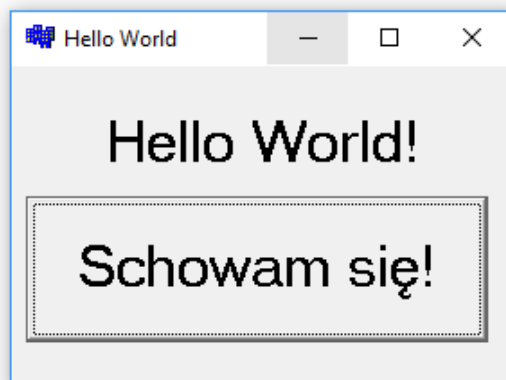
Dwukrotne kliknięcie w dany obiekt powoduje dodanie domyślnego zdarzenia dla danego obiektu. W przypadku przycisku będzie to interesująca nas opcja Button1Click (kiedy przycisk kliknięty).



Na początku napis jest ukryty, a na przycisku wyświetlony jest tekst „Kliknij mnie!” (do uruchomienia programu możemy użyć również przycisku F9).



Po uruchomieniu programu wartość dla pola „Label1 -> Visible” została przez nas ustawiona na false (**Visible = false**). W przypadku kliknięcia w przycisk instrukcja warunkowa if jest spełniona, wartość atrybutu „Label1 -> Visible” zostaje ustawiona na true (pojawienie się napisu) oraz wartość atrybutu „Button1->Caption” zostaje ustawiona na „Schowam się” (zmiana wyświetlanego napisu).



Po ponownym wciśnięciu przycisku, wartość atrybutu „Label1 -> Visible” zostaje ustawiona na false (napis znika) oraz wartość atrybutu „Button1->Caption” zostaje ustawiona na „Pokażę się”.




Poniżej fragment kodu, który musimy umieścić w okienku, które się pojawiło.

```
//-----  
void __fastcall TForm1::Button1Click(TObject *Sender)  
{  
    if (Label1->Visible==false)  
    {  
        Label1 -> Visible = true;  
        Button1 -> Caption = "Schowam się!";  
    }  
    else  
    {  
        Label1 -> Visible = false;  
        Button1 -> Caption = "Pokażę się!";  
    }  
}  
//-----
```

Zapalanie diody

W drugim programie zamiast pojawiającego się napis, użyjemy obiektu, dla którego będziemy zmieniać wartość atrybutu odpowiedzialnego za kolor. Możemy skorzystać z poprzedniego programu i zapisać nasz projekt pod inną nazwą.

Wykonanie projektu, na podstawie poprzedniego programu „Hello World”:

1. Usuwamy Obiekt „Label1” z naszego projektu (klikamy go lewym przyciskiem myszy i wciskamy klawisz Delete na klawiaturze).
2. Zwiększamy rozmiar okna projektu „Form1”. Nie musimy wpisywać wartości z klawiatury, wystarczy rozciągnąć okno przy pomocy myszki. Po najejchaniu kursorem na lewy dolny róg okna możemy je powiększyć lub zmniejszyć.
3. Dodajemy do projektu nowy obiekt z zakładki Additional: Shape , po przejściu do zakładki klikamy dwukrotnie odpowiednią ikonę.
4. Zmieniamy następujące ustawienia dodanego obiektu „Shape1”:

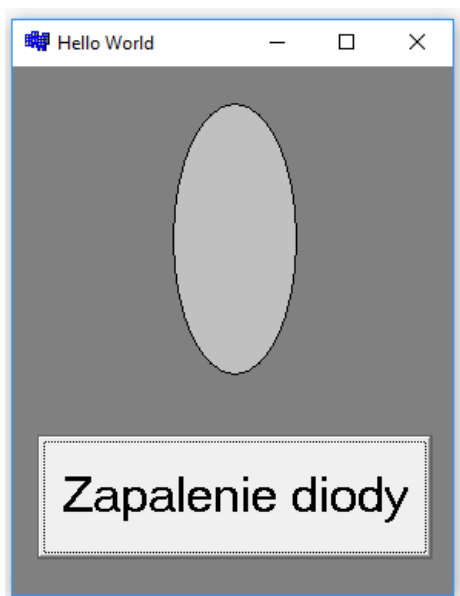
Brush: Color = **clSilver**

Shape = **stEllipse**

Położenie i kształt ustawiamy przy pomocy myszy.

5. Dla obiektu „Button1”, zmieniamy w ustawieniach „Properties” napis, który będzie wyświetlany po uruchomieniu programu:

Caption = **Zapalenie diody**



Modyfikacja programu

Warunkiem, który będzie sprawdzany w funkcji IF jest dotychczasowy kolor obiektu „Shape1”
Shape1 -> Brush -> Color==clSilver. Po kliknięciu w przycisk oraz spełnieniu warunku atrybut **Shape1 -> Brush -> Color** zostanie ustawiony na **clRed**. W przypadku kliknięcia przycisku i niespełnienia warunku **Shape1 -> Brush -> Color==clSilver** wartość atrybutu **Shape1 -> Brush -> Color** zostanie ustawiony na **clSilver**. Dodatkowo zmieniają się również napisy wyświetlane na przycisku „Button1” – zmiana atrybutu **Caption**.

Zmodyfikowany program:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    if (Shapel ->Brush->Color==clSilver)

    {   Shapel ->Brush->Color=clRed;
        Button1 -> Caption = "Zgaszenie diody";
    }
    else
    {
        Shapel ->Brush->Color=clSilver;
        Button1 -> Caption = "Zapalenie diody";
    }

}
```

Zadanie dodatkowe

Dodanie przycisków umożliwiających zmianę koloru diody.

Temat 13. Światła drogowe

Cel zajęć

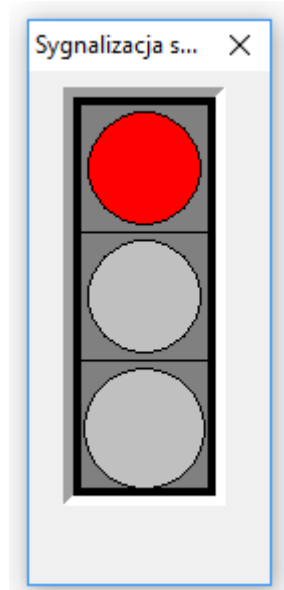
Wykonanie świateł drogowych, zmieniających się z określoną częstotliwością.

Utworzenie elementów graficznych aplikacji

1. Uruchomiamy program „Borland C++ Builder” z domyślnym formularzem
2. Zmieniamy tytuł formularza (**Caption**) na „Sygnalizacja swietlna”
3. Zmieniamy właściwość **BorderStyle** na **bsDialog**
4. Zmieniamy wymiary na **Height = 295** oraz **Width = 125**
5. W zakładce **Standard** klikamy dwukrotnie ikonę Panelu:

Zmieniamy właściwości dodanego Panelu w następujący sposób:

BevelInner = **bvNone**
BevelOuter = **bvLowered**
BevelWidth = **5**
Caption = **Empty**
Color = **clBlack**
Height = **209**
Left = **17**
Top = **8**
Width = **81**



6. Gdy Panel jest nadal zaznaczony, przechodzimy do zakładki **Additional**

i klikamy dwukrotnie na element dodający nowy kształt do projektu: **Shape**

Dla tak dodanego kształtu ustawiamy następujące parametry:

Brush: Color = **clGray**
Brush: Style = **bsSolid**
Height = **65**
Left = **8**
Top = **8**
Width = **65**

7. Tak utworzony Panel kopiujemy (CTRL+C) i wklejamy kopie (CTRL+V) do projektu.

Gdy nowy kształt jest już dodany, zmieniamy jego właściwości na:

Left = 8 oraz **Top** = 72.

W ten sam sposób dodajemy kolejny kształt zmieniając jego ustawienia na:


Left = 8 oraz **Top** = 136

8. Nadszedł czas na dodanie świateł, możemy skopiować jeden z dotychczas utworzonych paneli.

Tą operację powtarzamy jeszcze dwukrotnie, tworząc trzy kształty, dla których ustawiamy poniższe parametry:

: Color = clRed Height = 57 Left = 8	: Color = clSilver Height = 57 Left = 8	: Color = clSilver Height = 57 Left = 8
--	---	---

Name	= shpRed	Name	= shpYellow	Name	= shpGreen
Shape	= stCircle	Shape	= stCircle	Shape	= stCircle
Top	= 12	Top	= 76	Top	= 140

9. Na zakładce **System** wyboru komponentów kliknij dwukrotnie ikonę **Timer** , będzie on odpowiedzialny za zmiany kolorów świateł.

Programowanie aplikacji:

1. Naciskamy klawisz F12, aby wyświetlić formularz.
2. Dwukrotnie klikamy Timer w formularzu, aby uzyskać dostęp do zdarzenia **OnTimer**
3. Zmień kod w następujący sposób:

```
void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
    // jeżeli pali się światło czerwone i zgaszone jest światło żółte
    if( shpRed->Brush->Color == clRed && shpYellow->Brush->Color == clSilver)
    {
        // zapal światło żółte:
        Timer1->Interval      = 2000;
        shpRed->Brush->Color    = clRed ;
        shpYellow->Brush->Color = clYellow;
        shpGreen->Brush->Color = clSilver;
    }

    // jeżeli pali się światło czerwone i żółte
    else if( shpYellow->Brush->Color == clYellow && shpRed->Brush->Color == clRed)
    {
        // zapal światło zielone
        Timer1->Interval      = 5000;
        shpRed->Brush->Color    = clSilver;
        shpYellow->Brush->Color = clSilver;
        shpGreen->Brush->Color = clGreen;
    }

    // jeżeli pali się światło zielone
    else if( shpGreen->Brush->Color == clGreen )
    {
        // zapal światło żółte
        Timer1->Interval      = 2000;
        shpRed->Brush->Color    = clSilver;
        shpYellow->Brush->Color = clYellow;
        shpGreen->Brush->Color = clSilver;
    }

    // w przeciwnym wypadku pali się światło żółte
    else // if(shpYellow->Brush->Color == clYellow)
    {
        // zostaje zapalone światło czerwone
        Timer1->Interval      = 5000;
        shpRed->Brush->Color    = clRed;
        shpYellow->Brush->Color = clSilver;
        shpGreen->Brush->Color = clSilver;
    }
}
```

4. Możemy przejść do testowania utworzonej aplikacji.

Zadanie dodatkowe

Dodanie drugiego sygnalizatora, który będzie pokazywał przeciwne ustawienie świateł niż

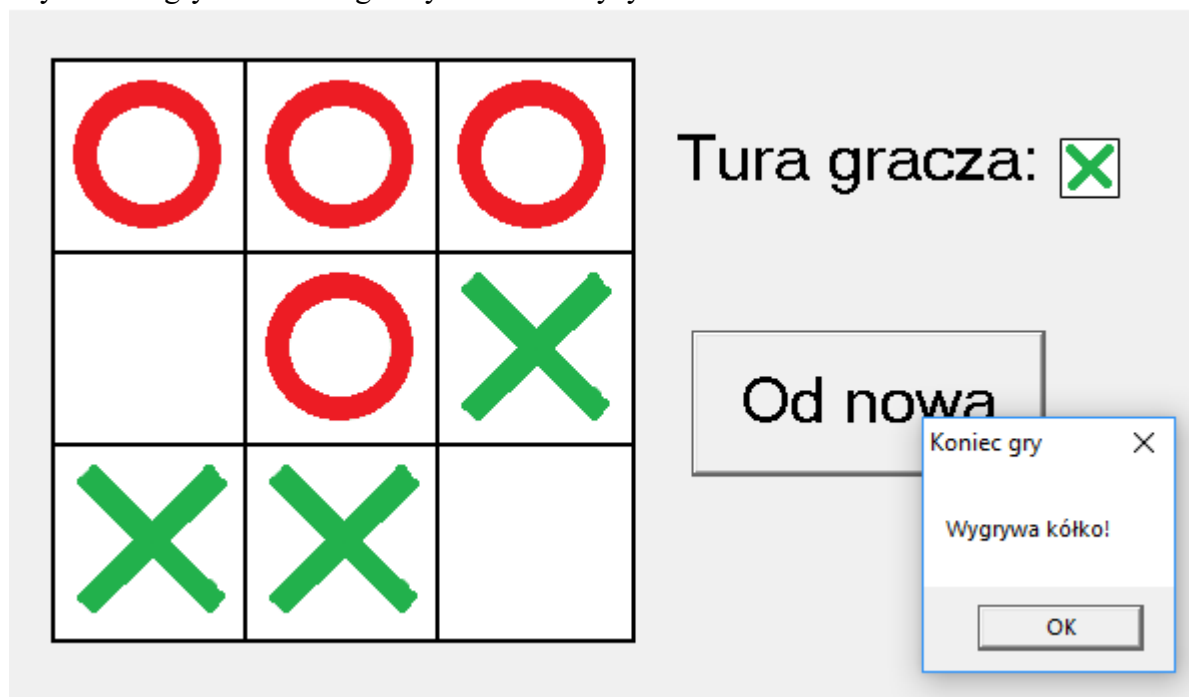
dotychczasowy sygnalizator (np. gdy na pierwszej sygnalizacji pali się światło czerwone, to dla drugiej sygnalizacji zapalone będzie światło zielone).

Wykorzystanie Timera do stworzenia „Migającej diody”.

Temat 14. Kółko i krzyżyk

Cel zajęć

Wykonanie gry dla dwóch graczy: kółko i krzyżyk.



Utworzenie elementów graficznych aplikacji

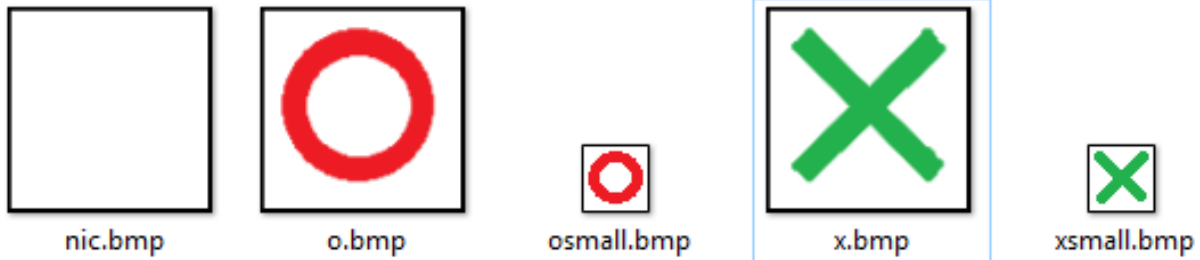
1. Uruchomiamy program „Borland C++ Builder” z domyślnym formularzem
2. Zmieniamy tytuł formularza (**Caption**) na „Kolko i krzyzyk”
3. Zmieniamy właściwość **BorderStyle** na **bsDialog**
4. Zmieniamy wymiary na **Height = 350** oraz **Width = 600**
5. Zapisujemy projekt
6. W projekcie będziemy potrzebować następujące ikony o rozmiarach 100x100 pikseli:

- puste pole (nic.bmp)
- duże kółko (o.bmp)
- duży krzyżyk (x.bmp)

oraz dwie małe ikony o rozmiarach 30x30 pikseli:

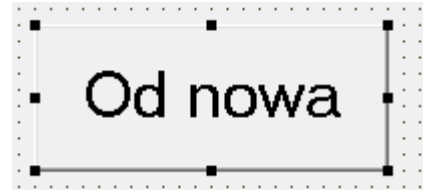
- małe kółko (osmall.bmp)
- mały krzyżyk (xsmall.bmp).

Możemy je wykonać w dowolnym edytorze graficznym (np. w programie Paint).

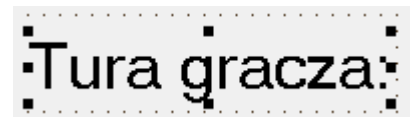



7. Umieszczamy je w folderze „img”, który umieszczamy w katalogu naszego projektu.

8. Dodajemy przycisk „Button1” na którym będzie wyświetlany napis „Od nowa”. Dostosowujemy wielkość czcionki.



9. Dodajemy pole tekstowe „Label1” z napisem „Tura gracza:”. Dostosowujemy wielkość czcionki.



10. Dodajemy nowy obiekt do naszego projektu: Image , znajdziemy go w zakładce Additional. Zmieniamy jego atrybuty:

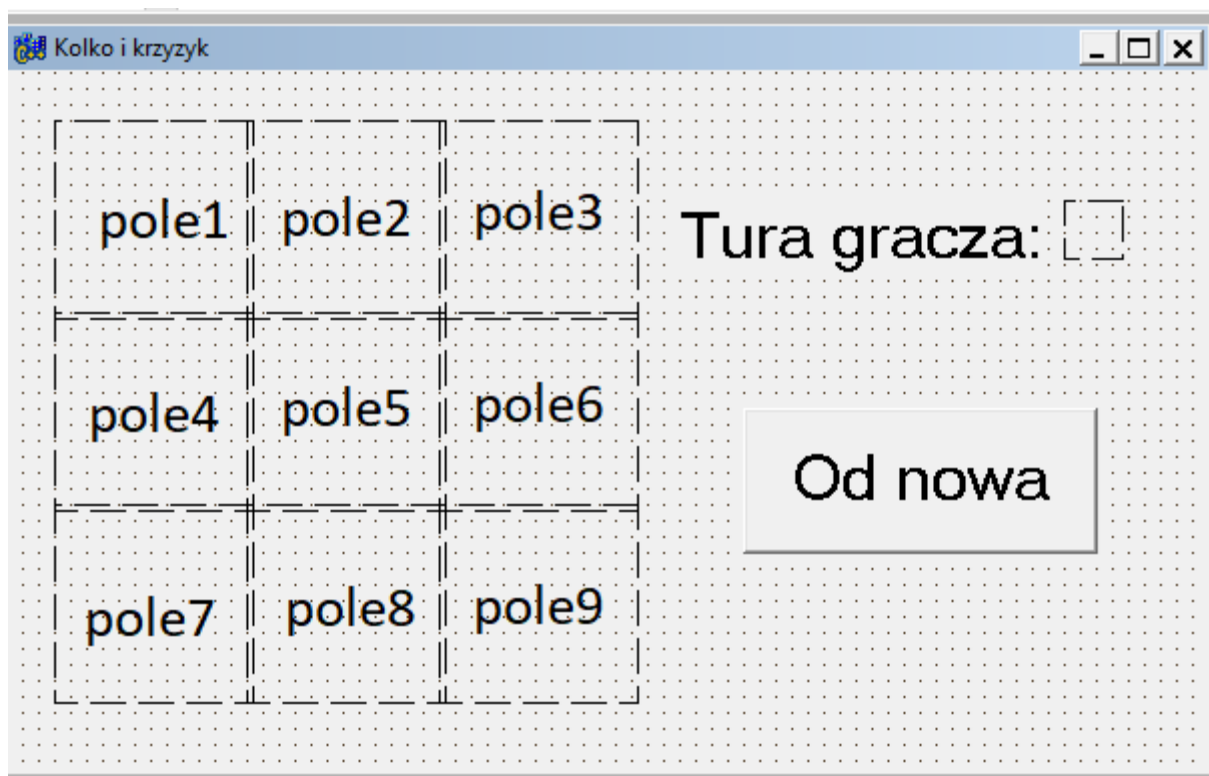
Height = 30
Width = 30
Name = tura

11. Kopiujemy obiekt „tura” (CTRL+C) i wklejamy go do naszego projektu (CTRL+V) . Zmieniamy jego atrybuty:

Height = 100
Width = 100
Name = pole1

12. Kopiujemy ośmiokrotnie obiekt „pole1” i zmieniamy ich nazwy na pole2, pole3, pole4, pole5, pole6, pole7, pole8, pole9.

13. Utworzone elementy rozmieszczamy w sposób przedstawiony na poniższym rysunku (opisy są w rzeczywistości niewidoczne – służą jedynie prawidłowemu rozmieszczeniu elementów w aplikacji).



Programowanie obiektów

1. Zaczynamy od zdefiniowania zmiennych typu char (przechowują one pojedynczy znak). Zmienne *p1*, *p2*, *p3*, *p4*, *p5*, *p6*, *p7*, *p8*, *p9* będą przyjmowały trzy wartości: „n”, „x” lub „o”; będą się zmieniały wraz ze zmianą obrazka wybranego pola.

Zmienna *kto* będzie przyjmowała dwie wartości „o” lub „x” określające kolej konkretnego gracza.

Poniższy skrypt umieszczamy w oknie *Unit1.cpp*, zaraz po linijce *TForm1 *Form1;*

```
#pragma resource "*.dfm"
TForm1 *Form1;

char p1,p2,p3,p4,p5,p6,p7,p8,p9;
//p1..p9 są to pola w grze (ich zawartosc może być równa: p1='n'; nie lub 'x' lub 'o' )
char kto;
```

2. Po dwukrotnym kliknięciu w obszar formularza *Form1* pojawi się procedura, która będzie uruchamiana podczas inicjalizacji programu. W nawiasie klamrowym „{ }” umieszczamy następujący kod:

```

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    pole1->Picture->LoadFromFile("img/nic.bmp");
    pole2->Picture->LoadFromFile("img/nic.bmp");
    pole3->Picture->LoadFromFile("img/nic.bmp");
    pole4->Picture->LoadFromFile("img/nic.bmp");
    pole5->Picture->LoadFromFile("img/nic.bmp");
    pole6->Picture->LoadFromFile("img/nic.bmp");
    pole7->Picture->LoadFromFile("img/nic.bmp");
    pole8->Picture->LoadFromFile("img/nic.bmp");
    pole9->Picture->LoadFromFile("img/nic.bmp");
    tura->Picture->LoadFromFile("img/osmall.bmp");

    p1='n'; p4='n'; p7='n';
    p2='n'; p5='n'; p8='n';
    p3='n'; p6='n'; p9='n';

    kto='o';

    pole1->Enabled = true;
    pole2->Enabled = true;
    pole3->Enabled = true;
    pole4->Enabled = true;
    pole5->Enabled = true;
    pole6->Enabled = true;
    pole7->Enabled = true;
    pole8->Enabled = true;
    pole9->Enabled = true;
}

```

Składa się on z czterech części. Najpierw atrybuty (**Picture->LoadFromFile(...)**) dla wszystkich pól zostają ustawione w taki sposób, aby został załadowany obrazek nic.bmp oraz dla obiektu tura został załadowany obrazek osmall.bmp.

Następnie zmienne od *p1* do *p9* zostają ustawione na wartość „n”, zmienna *kto* zostaje ustawiona na wartość „o” (grę rozpoczyna gracz „kółko”) oraz wartość atrybutu **Enabled** dla pól zostaje ustawiona na wartość **true** (pola będą aktywne).

3. Sprawdzamy działanie dotychczas wykonanej części programu.

4. Dwukrotnie klikamy na obiekt „pole1” - instrukcja umieszczona w nawiasie klamrowym zostanie wykonana po kliknięciu tego obiektu myszką.

Pierwsza instrukcja warunkowa **if(p1=='n')** sprawdza czy wartość zmiennej *p1* jest równa „n” (czy to pole nie zostało już wcześniej kliknięte). Jeżeli warunek jest ten spełniony skrypt przy pomocy instrukcji warunkowej sprawdza, którego gracza jest kolej.

Jeżeli wartość zmiennej *kto* była równa „o” wtedy obrazek dla obiektu pole1 zostaje zmieniony na o.bmp, zmienna *p1* zostaje ustawiona na wartość „o”, zmienna *kto* zostaje ustawiona na wartość „x” (rozpocznie się tura kolejnego gracza) oraz obrazek dla obiektu tura zostanie ustawiony na xsmall.bmp.

W przeciwnym wypadku obrazek dla obiektu `pole1` zostaje zmieniony na `x.bmp`, zmienna `p1` zostaje ustawiona na wartość „x”, zmienna `kto` zostaje ustawiona na wartość „o” oraz obrazek dla obiektu `tura` zostanie ustawiony na `osmall.bmp`.

```
void __fastcall TForm1::pole1Click(TObject *Sender)
{
    if (p1=='n')
    {
        if (kto=='o')
        {
            pole1->Picture->LoadFromFile ("img/o.bmp");
            p1='o';
            kto='x';
            tura->Picture->LoadFromFile ("img/xsmall.bmp");
        }
        else
        {
            pole1->Picture->LoadFromFile ("img/x.bmp");
            p1='x';
            kto='o';
            tura->Picture->LoadFromFile ("img/osmall.bmp");
        }
        pole1->Enabled=false;
        sprawdz ();
    }
}
```

Po wykonaniu instrukcji warunkowej pole to zostaje zdezaktywowane, wartość atrybutu **Enabled** zostaje ustawiona na `false`.

Zostaje wywołana funkcja **sprawdz()**; którą musimy zadeklarować (dodać) na początku programu. Funkcje deklarujemy jednorazowo. Ten fragment skryptu nie będzie musiał być przez nas powtarzany kilkakrotnie.

5. Funkcję deklarujemy poprzez słowo **void**. Za każdym razem gdy w skrypcie programu pojawi się wywołanie tej funkcji zostanie ona wykonana. W naszym przypadku umieszczona w niej jest instrukcja warunkowa, która sprawdza, czy wartości zmiennych od `p1` do `p9` nie są równe w trzech sąsiadujących ze sobą komórkach, przy czym musi być różna od „n”. Mamy osiem takich możliwości. Są one połączone ze sobą spójnikiem logicznym lub „||”. Jeżeli chociaż jeden z tych warunków jest spełniony, oznacza to koniec gry. Sprawdzana jest wartość zmiennej `kto`. Jeżeli jest ona równa „x” wygrywa kółko, w przeciwnym wypadku wygrywa krzyżyk. Na ekranie pojawia się odpowiedni komunikat.

```

char kto;

void sprawdz()
{
    if((p1==p2 && p2==p3 && p1!='n') ||
        (p4==p5 && p5==p6 && p4!='n') ||
        (p7==p8 && p8==p9 && p7!='n') ||
        (p1==p4 && p4==p7 && p7!='n') ||
        (p2==p5 && p5==p8 && p2!='n') ||
        (p3==p6 && p6==p9 && p3!='n') ||
        (p1==p5 && p5==p9 && p1!='n') ||
        (p3==p5 && p5==p7 && p3!='n'))
    {
        char * w;

        if (kto=='x') w="Wygrywa kółko!";
        else w="Wygrywa krzyżyk!";

        Application->MessageBox(w, "Koniec gry", MB_OK);
    }
}

```

6. Skrypty kolejnych pól będzie zbliżony do skryptu obiektu pole1. Możemy skopiować całą instrukcję warunkową wykonywaną po kliknięciu tego obiektu i wkleić ją jako instrukcję realizowaną po kliknięciu kolejnego z pól. Zmiany w stosunku do wklejonej instrukcji warunkowej zostały zaznaczone żółtym kolorem.

Po dokonaniu zmian we wszystkich obiektach pól program powinien działać prawidłowo.

7. Pozostało zaprogramowanie przycisku „Od nowa”. Skrypt wykonywany po kliknięciu tego obiektu będzie identyczny ze skryptem wykonywanym po uruchomieniu aplikacji. Możemy go skopiować i po dwukrotnym kliknięciu w przycisk „Od nowa” wkleić go do tego obiektu.
8. Testujemy naszą aplikację.

```

..
void __fastcall TForm1::pole2Click(TObject *Sender)
{
    if(p2=='n')
    {
        if(kto=='o')
        {
            pole2->Picture->LoadFromFile("img/o.bmp");
            p2='o';
            kto='x';
            tura->Picture->LoadFromFile("img/xsmall.bmp");
        }
        else
        {
            pole2->Picture->LoadFromFile("img/x.bmp");
            p2='x';
            kto='o';
            tura->Picture->LoadFromFile("img/osmall.bmp");
        }
        pole2->Enabled=false;
        sprawdz();
    }
}

```

```

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    pole1->Picture->LoadFromFile("img/nic.bmp");
    pole2->Picture->LoadFromFile("img/nic.bmp");
    pole3->Picture->LoadFromFile("img/nic.bmp");
    pole4->Picture->LoadFromFile("img/nic.bmp");
    pole5->Picture->LoadFromFile("img/nic.bmp");
    pole6->Picture->LoadFromFile("img/nic.bmp");
    pole7->Picture->LoadFromFile("img/nic.bmp");
    pole8->Picture->LoadFromFile("img/nic.bmp");
    pole9->Picture->LoadFromFile("img/nic.bmp");
    tura->Picture->LoadFromFile("img/osmall.bmp");

    p1='n'; p4='n'; p7='n';
    p2='n'; p5='n'; p8='n';
    p3='n'; p6='n'; p9='n';

    kto='o';

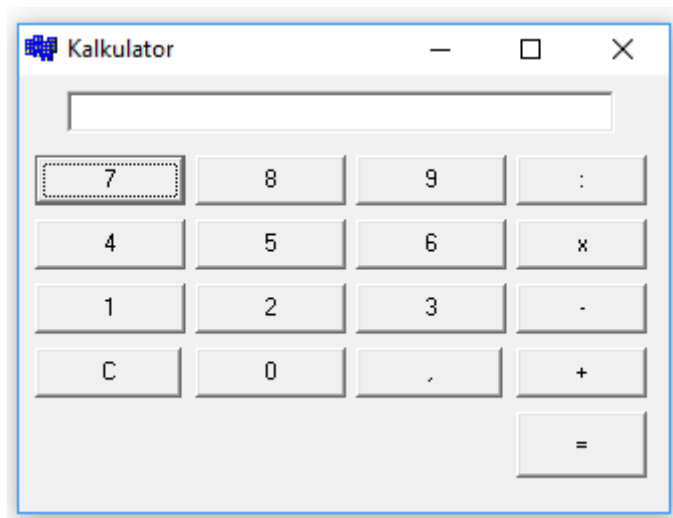
    pole1->Enabled = true;
    pole2->Enabled = true;
    pole3->Enabled = true;
    pole4->Enabled = true;
    pole5->Enabled = true;
    pole6->Enabled = true;
    pole7->Enabled = true;
    pole8->Enabled = true;
    pole9->Enabled = true;
}
..

```


Temat 15. Kalkulator graficzny

Cel zajęć

Wykonanie prostej wersji kalkulatora graficznego

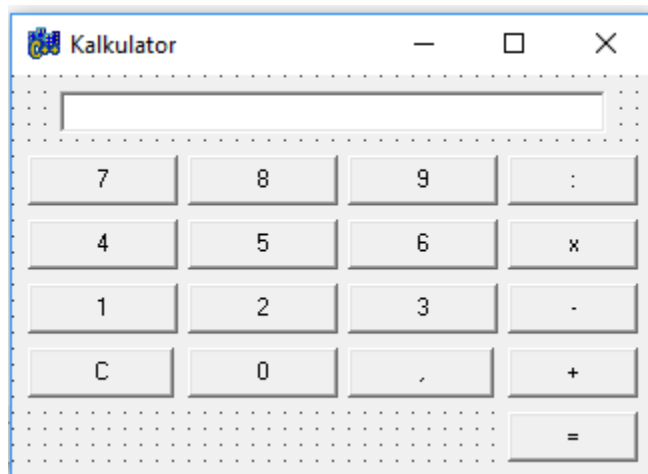


Utworzenie elementów graficznych aplikacji

1. Uruchomiamy program „Borland C++ Builder” z domyślnym formularzem
2. Zmieniamy tytuł formularza (**Caption**) na „Kalkulator”
3. Zmieniamy właściwość **BorderStyle** na **bsDialog**
4. Zmieniamy wymiary na **Height = 220** oraz **Width = 320**
5. Z zakładki Standard dodajemy nowy obiekt Edit  (pole tekstowe). Zmieniamy atrybut tego obiektu name na „wynik”.
6. Dodajemy siedemnaście przycisków ustawiając dla nich następujące wartości atrybutów
Caprion: 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, C, ”,”, :, x, -, +, =

Name: jeden, dwa, trzy, cztery, piec, szesc, siedem, osiem, dziewiec, zero, czysc, przecinekBTN, dzielenie, mnożenie, odejmowanie, dodawanie, wynik.

Rozmieszczamy je w sposób przedstawionym na poniższym rysunku.



Programowanie poszczególnych obiektów

1. Do prawidłowego działania programu będziemy potrzebować czterech zmiennych: dwóch przechowujących liczby zmiennoprzecinkowe typu float (zmienne *a* i *b*), zmienną *d* przechowującą znak działania, typu char (pojedynczy znak) oraz zmienną typu bool *przecinek*, która może przyjmować tylko wartości true lub false (1 lub 0).

```
TForm1 *Form1;  
//-----  
  
float a,b;  
char d;  
bool przecinek;
```

2. Po kliknięciu w klawisz „C” wartość pola tekstowego zostaje wyczyszczone. Po dwukrotnym kliknięciu w trybie edycji w przycisk „C” wpisujemy poniższe polecenie w nawiasie klamrowym {}.

```
void __fastcall TForm1::czyszcClick(TObject *Sender)  
{  
    wynik->Clear();  
}
```

3. Po kliknięciu w klawisz z dowolną liczbę, wartość atrybutu Text dla obiektu wynik jest przypisywana do zmiennej x typu AnsiString (ten typ zmiennej zapisuje łańcuch znaków jako tablice, dzięki czemu możemy odwołać się do konkretnej pozycji w tym łańcuchu). Do łańcucha znaków x dodawana jest wartość „0”. Łańcuch się wydłuża. Wartość atrybutu Text dla obiektu wynik zostaje ustawiona na wartość „wydłużonego” łańcucha znaków.

```
void __fastcall TForm1::zeroClick(TObject *Sender)  
{  
    AnsiString x = wynik->Text;  
    x += '0';  
    wynik->Text = x;  
}
```

4. Tą samą instrukcję kopiujemy do pozostałych przycisków, zmieniając symbol dopisywany do łańcucha znaków „**wynik->Text**”. Po kliknięciu klawisza jeden do ciągu znaków zostanie dopisana wartość „1”, po kliknięciu klawisza dwa zostanie dopisana wartość „2” i tak samo dla kolejnych klawiszy.


```

void __fastcall TForm1::jedenClick(TObject *Sender)
{
    AnsiString x = wynik->Text;
    x += '1';
    wynik->Text = x;
}

//-----
void __fastcall TForm1::dwaClick(TObject *Sender)
{
    AnsiString x = wynik->Text;
    x += '2';
    wynik->Text = x;
}

//-----

```

5. Skrypt umożliwiający dodanie przecinka zawiera instrukcję warunkową "if()", która sprawdza czy wartość zmiennej przecinek jest równa „false”. Wynikiem działania (!przecinek) dla wartości **false** będzie **true** (do ciągu znaków zostanie dodany przecinek, a wartość zmiennej przecinek zostanie ustawiona na true).

Po uruchomieniu programu zawartość zmiennej przecinek wynosi właśnie **false**.

```

void __fastcall TForm1::przecinekBTNClick(TObject *Sender)
{
    if (!przecinek)
    {
        AnsiString x = wynik->Text;
        x += ',';
        wynik->Text = x;
        przecinek = true;
    }
}

```

6. Programowanie przycisków działań. Po kliknięciu w jeden z przycisków odpowiedzialny za wykonywanie określonego działania:
- do zmiennej *a* zostanie przypisana wartość łańcucha znaków zawarta w atrybucie Text obiektu wynik (funkcja **StrToFloat()** zamienia łańcuch znaków na liczbę),
 - do zmiennej *d* zostanie przypisany odpowiedni znak działania,
 - atrybut **Text** obiektu **wynik** zostaje w kolejnym kroku wyczyszczony (ustawiony na ""),
 - do zmiennej przecinek zostaje przypisana wartość false.

```

//-----
void __fastcall TForm1::dzielenieClick(TObject *Sender)
{
    a = StrToFloat (wynik->Text);
    d = '/';
    wynik->Text = "";
    przecinek = false;
}

```

7. Podobny program dodajemy do kolejnych przycisków, zmieniając znak przypisywany do zmiennej *d*.

```
//-----  
void __fastcall TForm1::mnozenieClick(TObject *Sender)  
{  
    a = StrToFloat (wynik->Text);  
    d = '*';  
    wynik->Text = "";  
    przecinek = false;  
}  
//-----  
void __fastcall TForm1::odejmowanieClick(TObject *Sender)  
{  
    a = StrToFloat (wynik->Text);  
    d = '-';  
    wynik->Text = "";  
    przecinek = false;  
}  
//-----
```

8. Ostatnim przyciskiem, który pozostał do zaprogramowania jest znak równa się.

Wartość łańcucha znaków zostaje przypisana do zmiennej *b*. Wartość łańcucha znaków *s* zostaje ustawiona na „0”. W zależności od wartości zmiennej *d* (wyboru określonego działania) do atrybutu Text obiektu wynik zostanie przypisany określony wynik działania. W przypadku dzielenia dodatkowo sprawdzany jest warunek czy druga wartość nie jest równa 0. W takim wypadku do zmiennej *s* zostanie przypisany łańcuch znaków „Nie można dzielić przez 0”, który zostanie wyświetlony w polu tekstowym.

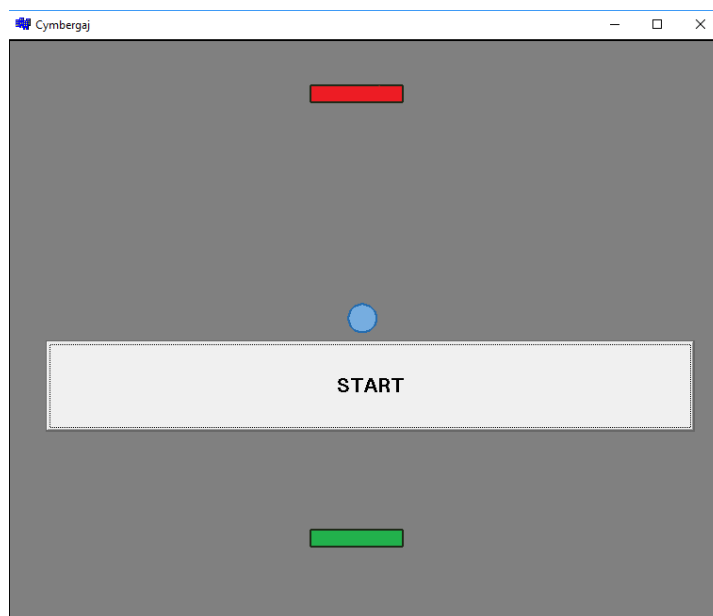
```
void __fastcall TForm1::rownaClick(TObject *Sender)  
{  
    b = StrToFloat(wynik->Text);  
    AnsiString s = 0;  
  
    switch (d)  
    {  
        case '+': s = FloatToStr(a+b); break;  
        case '-': s = FloatToStr(a-b); break;  
        case '*': s = FloatToStr(a*b); break;  
        case '/': if (b!=0) s = FloatToStr(a/b);  
        else s = ("Nie mozna dzielic przez 0");  
        break;  
    }  
  
    wynik->Text = s;  
}
```

Jest to bardzo prosta wersja programu, posiadająca pewne niedoskonałości. Wynikają one z uproszczenia działania programu kalkulatora do celów dydaktycznych (np. dwukrotne wciśnięcie klawisza któregoś z działań spowoduje powstanie błędu – brak łańcucha znaków do przypisania).



Temat 16. Cymbergaj

Cel zajęć

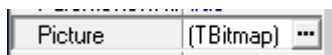
Wykonanie gry dla dwóch graczy „Cymbergaj”, w której zadaniem jest odbijanie piłeczki przy pomocy paletek.



Utworzenie elementów graficznych aplikacji

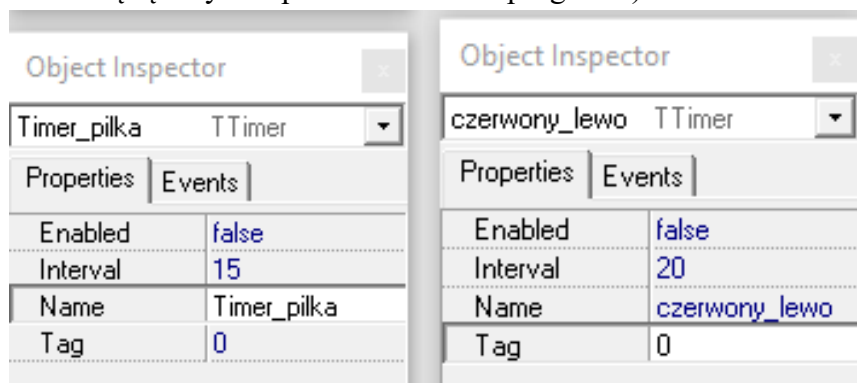
1. Uruchamiamy program „Borland C++ Builder” z domyślnym formularzem
2. Zmieniamy tytuł formularza (**Caption**) na „Kalkulator”
3. Zmieniamy właściwość **BorderStyle** na **bsDialog**
4. Zmieniamy wymiary na **Height = 650** oraz **Width = 780**
5. Dodajemy do projektu nowy obiekt z zakładki Additional: Shape , po przejściu do zakładki klikamy dwukrotnie odpowiednią ikonę. Zmieniamy atrybut **name** na **tlo**, ustawiamy atrybut **Align** na **alClient**  (figura będzie dostosowywała się do wielkości okna) oraz zmieniamy atrybut **Shape1 -> Brush -> Color** na **clGray**.
6. Dodajemy przycisk Button z napisem „START”, jego wielkość oraz położenie ustawiamy przy pomocy myszki.
7. Piłkę dodajemy jako obiekt **Image**, przy pomocy atrybutu Picture po kliknięciu w trzy kropki możemy wczytać obrazek z pliku.

Ustawiamy atrybut **Name** dla piłki na wartość „b”



8. W analogiczny sposób jak piłkę dodajemy dwie paletki. Przy górnej krawędzi dodajemy czerwoną paletkę (ustawiamy dla niej atrybut **Name** na „gracz_czerwony”). Przy dolnej krawędzi umieszczamy zieloną paletkę (ustawiamy dla niej atrybut **Name** na „gracz_zielony”).

9. Do sterowania paletkami oraz ruchu piłki wykorzystamy Timer'y. Zmieniamy wartość atrybutu Interval=15 lub 20, Name=[Timer_pilka, czerwony_lewo, czerwony_prawo, zielony_lewo, zielony_prawo] oraz Enable=False (Timery nie będą aktywne po uruchomieniu programu).



Programowanie poszczególnych obiektów

1. Na początku programu musimy zdefiniować dwie zmienne typu int x i y , będą one odpowiedzialne za ruch piłki. Im większa wartość ujemna tym szybciej piłka będzie się poruszać.

```
#pragma resource
TForm1 *Form1;

int x=-2, y=-2;
```

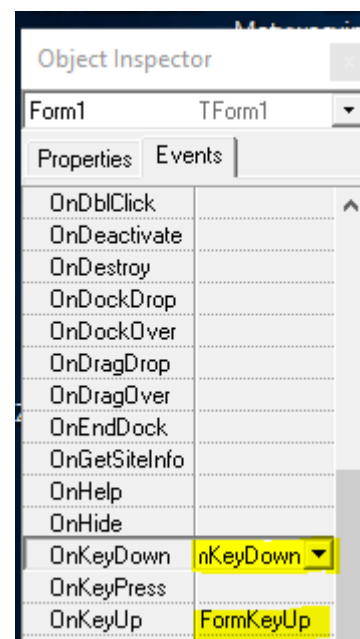
2. Zaczniemy od zaprogramowania ruchu paletek, po wciśnięciu danego klawisza będzie uruchamiany dany Taimer.

W celu wybrania i zaprogramowania odpowiedniego zdarzenia w **Object Inspector** klikamy dwukrotnie na szare pole obok zdarzenia **OnKeyDown** (kiedy klawisz wciśnięty). W miejsce, które się pojawiło, przepisujemy górny fragment kodu umieszczony na kolejnej stronie.

Następnie klikamy dwukrotnie lewym klawiszem myszy obok pola **OnKeyUp** (kiedy klawisz puszczoney). W miejsce, które się pojawiło, przepisujemy dolny fragment kodu umieszczony na kolejnej stronie.

Kody umieszczone w nawiasach okrągłych odpowiadają następującym przyciskom na klawiaturze:

Key==0x41 – klawisz „a”
 Key==0x44 – klawisz „d”
 Key==VK_LEFT – strzałka w lewo
 Key== VK_RIGHT – strzałka w prawo



```

//-----
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
    TShiftState Shift)
{
    if(Key==0x41) czerwony_lewo->Enabled=true;
    if(Key==0x44) czerwony_prawo->Enabled=true;
    if(Key==VK_LEFT) zielony_lewo->Enabled=true;
    if(Key==VK_RIGHT) zielony_prawo->Enabled=true;
}
//-----
void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key,
    TShiftState Shift)
{
    if(Key==0x41) czerwony_lewo->Enabled=false;
    if(Key==0x44) czerwony_prawo->Enabled=false;
    if(Key==VK_LEFT) zielony_lewo->Enabled=false;
    if(Key==VK_RIGHT) zielony_prawo->Enabled=false;
}
//-----

```

3. Po dwukrotnym kliknięciu timera zielony_lewo pomiędzy nawiasami klamrowymi wpisujemy poniższy kod odpowiedzialny za ruch gracza zielonego w lewo. Gdy wartość atrybutu Left dla gracza Zielonego nie będzie większa niż 10, nie będzie on przesuwany dalej w lewą stronę. W innym wypadku przesunie się o 10 jednostek (pikseli).

```

//-----
void __fastcall TForm1::zielony_lewoTimer(TObject *Sender)
{
    if (gracz_zielony->Left>10) gracz_zielony->Left-=10;
}

```

4. Po dwukrotnym kliknięciu timera zielony_prawo lewo pomiędzy nawiasami klamrowymi wpisujemy poniższy kod. W tym wypadku warunek jest bardziej rozbudowany, ponieważ porównujemy położenie obiektu gracza_zielonego w stosunku do szerokości obiektu tło, przy uwzględnieniu szerokości obiektu gracza_zielonego.

```

//
void __fastcall TForm1::zielony_prawoTimer(TObject *Sender)
{
    if (gracz_zielony->Left+gracz_zielony->Width<tlo->Width-10) gracz_zielony->Left+=10;
}
//

```

5. Programy dla Timerów **czerwony_lewo** i **czerwony_prawo** wykonujemy w sposób analogiczny do dwóch powyższych punktów, przepisując kod poniżej lub kopiując poprzednie kody i zmieniając nazwy obiektów.

```
void __fastcall TForm1::czerwony_lewoTimer(TObject *Sender)
{
    if (gracz_czerwony->Left>10) gracz_czerwony->Left-=10;
}
//-----

void __fastcall TForm1::czerwony_prawoTimer(TObject *Sender)
{
    if (gracz_czerwony->Left+gracz_czerwony->Width < tlo->Width-10) gracz_czerwony->Left+=10;
}
//-----
```

6. Po dwukrotnym kliknięciu w przycisk START pomiędzy nawiasami klamrowymi wpisujemy poniższy kod programu. Większość elementów jest już nam dobrze znana. Piłka (b) ustawia się pośrodku planszy, wyśrodkowują się paletki obu graczy, piłka się pokazuje, a zmienne odpowiedzialne za ruch zostają ustawione na wartości początkowe. Obiekt Timer_pilka zostaje aktywowany (zmiana atrybutu **Enabled** na **true**), po czym przycisk staje się niewidoczny (zmiana atrybutu Visible).

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    b->Left=360; b->Top=280;
    gracz_zielony->Left=320;
    gracz_czerwony->Left=320;
    b->Visible=true;
    x=-2;
    y=-2;
    Timer_pilka->Enabled=true;
    Button1->Visible=false;
}
```

7. Pozostało zaprogramowanie ostatniego Timera odpowiedzialnego za ruch piłki. Po dwukrotnym kliknięciu w obiekt Timer_pilka pomiędzy nawiasami klamrowymi umieszczamy kolejne części skryptu.

- a) Część programu odpowiedzialna za ruch piłki oraz odbijanie się od ścian:

```
b->Left-=x; b->Top-=y;
```

```
//odbij od lewej sciany
```

```
if (b->Left-5 <= tlo->Left) x=-x;
```

```
//odbij od prawej sciany
```

```
if (b->Left+b->Width+5 >= tlo->Width) x=-x;
```

- b) Część skryptu odpowiedzialna za wygraną gracza zielonego oraz za odbijanie się od czerwonej paletki.

```
//wygrana gracza zielonego
if (b->Top <= gracz_czerwony->Top+gracz_czerwony->Height-15)
{
    Timer_pilka->Enabled = false;
    b->Visible=false;
    Button1->Caption = "Wygrana gracza zielonego, chcesz spróbować jeszcze raz?";
    Button1->Visible=true;
}
//odbicie piłki (b) od górnej paletki (gracz_czerwony)

else if(b->Left > gracz_czerwony->Left-b->Width/2 &&
b->Left < gracz_czerwony->Left+gracz_czerwony->Width &&
b->Top+b->Height <= gracz_czerwony->Top+45)
{
    if (y>0) y=-y;
}
```

- c) Część skryptu odpowiedzialna za wygraną gracza czerwonego oraz za odbijanie się od zielonej paletki.

```
//wygrana gracza czerwonego
if (b->Top >= gracz_zielony->Top+gracz_zielony->Height+15)
{
    Timer_pilka->Enabled = false;
    b->Visible=false;
    Button1->Caption = "Wygrana gracza czerwonego, chcesz spróbować jeszcze raz ?";
    Button1->Visible=true;
}

//odbicie piłki (b) od dolnej paletki (gracz_zielony)
else if(b->Left > gracz_zielony->Left-b->Width/2 &&
        b->Left < gracz_zielony->Left+gracz_zielony->Width &&
        b->Top+b->Height >= gracz_zielony->Top)
{
    if (y<0) y=-y;
}
```

Po wygranej któregoś z graczy obiekt `Timer_pilka` się dezaktywuje, piłka znika, pojawia się za to przycisk `Button1` z informacją o tym, który gracz wygrał. Po ponownym kliknięciu przycisku gra rozpocznie się na nowo.

Po trafieniu piłki w paletkę (zarówno czerwoną jak i zieloną) zmienna `y` jest ustawiana na wartość „-y”, oznacza to, że piłka zmienia kierunek ruchu wzdłuż osi rzędnych (piłka się odbija).

8. Testujemy działanie wykonanej aplikacji.