Przykładowe skrypty szkoleniowe

Kurs programowania C++ składał się z 10 części, które wprowadziły nauczycieli w świat programowania zorientowanego obiektowo (OOP), a także szereg pomocniczych narzędzi i bibliotek, takich jak STL, Boost, bez których używanie C++ byłoby bardzo trudne.

Historia

Język C++ został stworzony w latach osiemdziesiątych XX wieku jako obiektowe rozszerzenie języka C. Początkowo najważniejszą zmianą wprowadzoną w C++ w stosunku do C było programowanie obiektowe, później jednak zaimplementowano wiele innych ulepszeń, mających uczynić ten język wygodniejszym i bardziej elastycznym od swojego pierwowzoru. Nazwa języka odzwierciedla fakt, że język ten jest rozszerzeniem języka C.

Cechy języka

Język C++ jest obecnie najważniejszym, najczęściej stosowanym językiem programowania. Dzieje się tak, ponieważ:

- zawiera on niewielką ilość stosunkowo łatwych do opanowania słów kluczowych,
- programy źródłowe są stosunkowo łatwo przenoszone między różnymi platformami,
- dysponuje obszernymi bibliotekami na wszelkie okazje – Internet, grafika, nauka, inżynieria,
- jest bardzo uniwersalny.

No to zaczynamy

Nie tak prędko, aby rozpocząć swoją przygodę z C++ konieczny jest kompilator. Skąd go wziąć? Opisane jest na stronie Kompilatory C/C++ (http://pl.cpp.wikia.com/wiki/Kompilatory).
Aby móc Ci to wytłumaczyć, musimy przeanalizować w jaki sposób powstaje program w C++.

Tworzenie programu w C++ składa się z **4 etapów**. Pierwszy etap to edycja. W tym etapie wykorzystywany jest program edytora (np. Notatnik w Windows). Jest to moim zdaniem najważniejsza faza, gdyż tutaj wpisujesz instrukcje, które wykona komputer. Następnie plik z instrukcjami zapisywany jest na dysku. Kolejnym etapem jest kompilacja programu. Kompilacja polega na „przetłumaczeniu" instrukcji w C++ na język wewnętrzny procesora. Zanim nastąpi kompilacja wykonywany jest automatycznie program preprocesora. Preprocesor to taki program, który wykonuje instrukcje zwane dyrektywami preprocesora. Zadaniem tych instrukcji jest wykonanie pewnych czynności, zanim jeszcze dokonana zostanie kompilacja programu. Te czynności to np. dołączenie innych plików tekstowych. Preprocesor wywoływany jest przez kompilator, zanim program zostanie przekształcony na język maszynowy. Ostatnim etapem jest łączenie. Na czym to polega? Otóż programy w C++ składają się z funkcji lub obiektów zdefiniowanych w innych plikach (np. bibliotekach standardowych). Kod tworzony przez kompilator zawiera puste miejsca, które muszą zostać wypełnione przez funkcje z tych innych plików, tak aby stworzyć program w pełni wykonywalny.

Pierwszy program - "Hello world"

Działanie naszego programu będzie polegało na wyświetleniu linii tekstu.

```
// "Hello World" pierwszy program w C++.
#include <iostream>
#include <conio.h>

//using namespace std;
//przez powyższą funkce nie musisz wszędzie pisać std::

int main()
{
  std::cout << "Hello World!!!\n";
//można to też zapisać tak:
//std::cout << "Hello World!!!" << endl;
  getch();
  return 0;
}
```

Teraz przeanalizujmy program linijka po linijce, aby wszystko stało się jasne.

```
// "Hello World" pierwszy program w C++.
```

Znaki // oznaczają dla kompilatora, że po nich znajduje się komentarz, a nie instrukcje programu. Programiści używają komentarzy dla dokumentacji programu. Jest to szczególnie użyteczne w przypadku dużych projektów. Komentarze ułatwiają zrozumienie programu innym osobom

i nie powodują żadnych działań ze strony komputera, a podczas kompilacji są (przez kompilator) pomijane.

#include <iostream>
#include <conio.h>

Jest to dyrektywa preprocesora, czyli instrukcja dla programu preprocesora. Linie rozpoczynające się od znaku # są przetwarzane przez preprocesor zanim program zostanie skompilowany. Instrukcja include nakazuje preprocesorowi umieszczenie w programie zawartości pliku nagłówkowego iostream.h . Ten plik musi być dołączony, ponieważ nasz program wysyła informacje wyjściowe na ekran (io - input/output, stream - strumień).
using namespace std;
Gdy się to umieści na początku kodu, nie trzeba już pisać przy funkcjach std:: (np. zamiast std::cout wystarczy cout)

int main()

Każdy program w C++ składa się z jednej lub kilku funkcji, z których tylko jedna musi nazywać się main. Programy C++ zawsze rozpoczynają wykonywanie funkcji od main, nawet jeśli nie jest to pierwsza funkcja. Słowo int oznacza, że funkcja zwraca wartość będącą liczbą całkowitą. Lewy nawias klamrowy ‘ { ‘ pokazuje początek bloku funkcji, odpowiadający mu prawy nawias kończy ostatnio rozpoczęty i jeszcze nie zamknięty blok funkcji. Więcej o funkcjach będzie w 3 odcinku naszego kursu.

std::cout << “Hello World\n”;
//lub
std::cout << “Hello World!!!” << endl;

Linia ta to pojedyncza instrukcja.Nakazuke ona wyświetlenie napisu “Hello World” oraz przeniesienie kursora do nowej linii (co czyni instrukcja /n lub endl<. Składa się ona z obiektu strumienia standardowego wyjścia cout (ang. see out), przyłączonego normalnie do ekranu, operatora <<, zwanego operatorem wstawiania do strumienia, napisu “Hello World\n” oraz średnika ;. Każda instrukcja musi się kończyć średnikiem (jest to tzw. zakończenie instrukcji, ale o tym dalej). Brak średnika na końcu instrukcji powoduje wygenerowanie błędu składniowego. Operator << nazywany jest operatorem wstawiania do strumienia, natomiast napis “Hello World\n” to jego prawy operand (wartość znajdująca się na prawo od operatora). Backslash to znak specjalny, zmieniający znaczenie następnego znaku. Kiedy znak \ zostanie napotkany w napisie, to następny znak jest z nim łączony w celu stworzenia sekwencji o specjalnym

znaczeniu. Sekwencja “\n” i “endl” oznacza przejście do nowej linii.

getch();

To jest również pojedyncza instrukcja, która wymaga wciśnięcia dowolnego przycisku na klawiaturze, aby przejść dalej. By jej użyć trzeba użyć conio.h Można to przetłumaczyć jako “get char” - “weź znak”. Na jej końcu są dwa nawiasy ().
O ich roli dowiesz się później. Funkcja ta musi (!!!) być zakończona średnikiem (jak zresztą większość funkcji).

return 0;

Za pomocą słowa kluczowego return zwracana jest wartość z funkcji. Wartość 0 dla funkcji main oznacza, że program zakończył się pomyślnie. Więcej na temat zwracania różnych wartości przez funkcje będzie w 3 odcinku.

Zmienne i arytmetyka w C++

Na początek mały programik.

```
// Program, który prosi użytkownika o podanie
// 2 liczb, a następnie je mnoży
// i wyświetla wynik na ekranie.

#include <iostream>

using namespace std;

int main()
{
 // Do dobrych nawykow nalezy inicjowanie zmiennych
 // wartosciami
 int liczba1 = 0;
 int liczba2 = 0;
 int iloczyn = 0;

 cout<<”Wprowadz pierwsza liczbe calkowita\n”;
 cin>>liczba1;
```

```
 cout<<”Wprowadz druga liczbe calkowita\n”;
 cin>>liczba2;

 // Pobieramy znak nowej linii ktory
 // pozostal w strumieniu po poprzedniej operacji
 cin.get();

 iloczyn=liczba1*liczba2;
 cout<<” Iloczyn wynosi : “<<iloczyn<<endl;

 //Czekamy na wcisniecie klawisza Enter
 cout<<”Wcisnij Enter zeby zakonczyc”<<endl;
 cin.get();
 return 0;
}
```

Jeśli chodzi o linie:

```
// Program, który prosi użytkownika
// o podanie 2 liczb, a następnie je mnoży
// i wyświetla wynik na ekranie.

#include <iostream>

using namespace std;

int main()
{
```

patrz wyżej.

```
int liczba1,liczba2,iloczyn;
```

Linia ta to deklaracja trzech zmiennych całkowitych o nazwach liczba1, liczba2, iloczyn. Deklarację można również napisać w ten sposób:

```
int liczba1;
int liczba2;
int iloczyn;
```

Powyżej zmienna deklarowana jest w trzech oddzielnych instrukcjach. Poniżej deklarujemy zmienne w jednej instrukcji, ale w trzech liniach :

```
int liczba1,
liczba2,
iloczyn;
```

Nazwa zmiennej to dowolny dozwolony identyfikator. Identyfikator to układ znaków składający się z liter, cyfr i znaków podkreślenia(_). Nie może on rozpoczynać się cyfrą ani być słowem kluczowym. W C++ rozróżniane są duże i małe litery, więc Liczba1 i liczba1 to dwie różne zmienne. Deklaracje zmiennych mogą być umieszczane wszędzie w funkcji, pod warunkiem, że deklaracja pojawi się zanim zmienna zostanie użyta. Każda zmienna ma nazwę, typ, rozmiar i wartość. Rozmiar zmiennej określa ile komórek w pamięci zajmuje zmienna. Jeśli chodzi o typ zmiennej to określa go słowo kluczowe typu danych poprzedzające nazwę zmiennej (w naszym wypadku jest to słowo int wskazujące na to, że zmienna jest typu integer - liczbą całkowitą). Wartość zmiennej liczba1 jest nadawana instrukcją:

```
cin>>liczba1;
```

Instrukcja ta używa obiektu strumienia wejściowego cin(ang. see in) oraz operatora pobierania za strumienia >> do uzyskania liczby wpisywanej przez użytkownika. Cin i operator >> pobiera wejście ze standardowego strumienia wejściowego jakim zazwyczaj jest klawiatura i przypisuje ją zmiennej liczba1. Inaczej mówiąc cin przekazuje wartość wpisaną przez użytkownika do zmiennej liczba1. Tak naprawdę wartość ta jest przekazywana do komórek w pamięci komputera, którym kompilator przypisał nazwę zmiennej.

```
iloczyn=liczba1*liczba2;
```

Instrukcja oblicza iloczyn zmiennych liczba1 oraz liczba2 oraz przypisuje wynik do zmiennej iloczyn. Instrukcja ta używa operatora przypisania = oraz operatora * wskazującego na mnożenie. Są one nazywane operatorami dwuargumentowymi, ponieważ każdy z nich ma dwa operandy. W przypadku operatora * prawym operandem jest zmienna liczba2, a lewym operandem zmienna liczba1. Operandy operatora = to wartość wyrażenia liczba1*liczba2 oraz zmienna iloczyn.

Niektóre wbudowane typy danych w C++

Dane znakowe. Wartością danej typu znakowego char mogą być pojedyncze litery, cyfry oraz inne znaki. W pamięci z reguły zajmują 1 bajt.

Liczby całkowite.

short int , int, unsigned int, long int, unsigned long int

Liczba bajtów, zajmowanych przez typy zmiennych, zależna jest od systemu operacyjnego, a od tego zależy jaką maksymalną wartość może przechowywać zmienna danego typu. Przykładowo zmienna typu short int zajmować będzie najmniej miejsca w pamięci. Przechowywane w niej wartości będą obejmowały mały zakres (np. 0-255), ale to zależy już od systemu operacyjnego bądź kompilatora. Jak sprawdzić ile bajtów zajmuje dany typ pokażę w dalszej części kursu. Typ unsigned long int oraz unsigned int oznacza dodatnią liczbę całkowitą.

Liczby zmiennoprzecinkowe.

float, double, long double

Arytmetyka w C++

Poniżej przedstawiam operatory arytmetyczne oraz opisy ich działania:

OPERATOR ARYTMETYCZNY	WYRAŻENIE ALGEBRAICZNE	WYRAŻENIE W C++	WYNIK W C++	OPIS
+	2+5	2+5	7	Dodawanie
-	5-2	5-2	3	Odejmowanie
*	2x5	2*5	10	Mnożenie
/	7/2	7/2	3	Dzielenie całkowite. Wynik z dzielenia całkowitego jest zawsze liczbą całkowitą, część ułamkowa jest po prostu obcinana.
/	7/2	7.0/2	3,5	Dzielenie z liczbami zmienno-pozycyjnymi.
%	7 mod 2	7%2	1	Dzielenie modulo. Operator dzielenia modulo dostarcza resztę po dzieleniu całkowitym. Używać go można jedynie z liczbami całkowitymi.

Operatory relacyjne oraz wprowadzenie do if

Struktura if pozwala na podejmowanie decyzji opartych na prawdzie lub fałszu jakiś warunków.
Struktura ta ma postać:

```
if (warunek)
{
 instrukcja;
```

```
...
...
...
 instrukcja;
 instrukcja;
 //itd.
}
```

Jeśli warunek jest spełniony, instrukcje wewnątrz nawiasów klamrowych zostaną wykonane. Jeśli warunek nie będzie spełniony instrukcje te zostaną pominięte. Warunki mogą być formułowane za pomocą operatorów relacyjnych lub operatorów równości, które przedstawiam poniżej:

Operator	Przykład	Znaczenie
==	x==y	true jeśli zmienne x i y są sobie równe
!=	x!=y	true jeśli zmienne x i y nie są sobie równe
>	x>y	true jeśli x jest większy od y
<	x<y	true jeśli x jest mniejszy od y
>=	x>=y	true jeśli x jest większy lub równy y
<=	x<=y	true jeśli x jest mniejszy lub równy y

Pamiętaj, aby nie mylić operatora przypisania = z operatorem równości ==. Zasady pierwszeństwa operatorów w C++:

1. Operatory w wyrażeniach znajdujących się wewnątrz nawiasów są obliczane w pierwszej kolejności. W przypadku zagnieżdżonych nawiasów, operatory w najbardziej wewnętrznej parze obliczane są pierwsze.
2. Dzielenie, mnożenie i dzielenie modulo obliczane są w następnej kolejności.
3. Następnie wykonywane jest dodawanie i odejmowanie.
4. Dalej wykonywane są operatory relacyjne(<,>,<=,>=).
5. Potem operatory równości(== i !=).
6. Operator przypisania (=).

Spróbuj określić jaki będzie porządek wykonywania działań w poniższych wyrażeniach:

```
Z=w/x-y+r%q*c;
```

```
Z=a*b*c*d/e;
```

```
Z=y*x%f-g*c+g;
```

A teraz czas na zastosowanie tego, co już umiemy. Poniżej przedstawiam malutki programik do analizy.

```
// Program, który prosi użytkownika
// o podanie 1 liczby, a następnie sprawdza
// czy podana liczba jest parzysta.

#include <iostream>
#include<conio.h>

using namespace std;

int main()
{
 int liczba;
 cout<<”Wprowadz liczbe calkowita a”;
 cout<<” powiem Ci czy jest ona liczba parzysta czy nie ?\n”;
 cin>>liczba;
 if(liczba%2==0){
  cout<<”Ta liczba “<<liczba<<” jest parzysta. “;
 }
 if(liczba%2!=0){
  cout<<”Ta liczba “<<liczba<<” jest nieparzysta. “;
 }
 getch();
 return 0;
}
```

Wyjaśnienia wymaga linia:

```
cout<<”Ta liczba “<<liczba<<” jest parzysta. “;
```

Ta instrukcja używa kaskadowej operacji wstawiania do strumienia. Najpierw wstawiany jest napis “Ta liczba “, następnie wstawiana jest wartość zmiennej liczba, a na końcu napis “ jest parzysta. “. Dzięki temu nie musimy pisać:

```
cout<<”Ta liczba “;
cout<<liczba;
cout<<” jest parzysta. “;
```

Ćwiczenia

1. Napisz program, który wczytuje 2 liczby i podaje, która jest większa lub mniejsza albo czy są one równe.
2. Napisz program, który wczytuje 2 liczby i podaje iloraz, iloczyn, sumę i różnicę.
3. Napisz program, który wczytuje dwie liczby całkowite i oblicza pole prostokąta oraz jego obwód.
4. Napisz program, który odczytuje 2 liczby i określa czy pierwsza jest wielokrotnością drugiej.
5. Zapoznaj się dokładnie z dokumentacją dostarczoną wraz z kompilatorem. Przeczytaj dokładnie fragmenty dotyczące kompilacji oraz łączenia.
6. Napisz program, który wczytuje 3 liczby i mówi, która jest większa.

Kurs część 2

Algorytmy

Procedura służąca rozwiązaniu problemu, określona przez działania jakie należy wykonać oraz porządek w jakim te działania zostaną wykonane, nosi nazwę algorytmu. Przed napisaniem programu służącego rozwiązaniu określonego problemu, niezbędne jest dogłębne poznanie, zrozumienie i przemyślany plan jego rozwiązania. Rozważmy taki oto przykład (bardzo często pojawiający się w różnych podręcznikach programowania) “algorytm ubierania się” : (1) załóż majtki, (2) załóż skarpetki, (3) załóż podkoszulek, (4) załóż spodnie, (5) załóż sweter, (6) załóż kurtkę, (7) załóż buty. Mamy tu wymienione działania, jakie należy wykonać wyrażone w czasowniku “załóż” oraz porządek, w jakim mają one być wykonane wyrażony cyframi. Gdybyśmy nie trzymali się określonego porządku wykonania działań, moglibyśmy np. założyć skarpetki na buty albo majtki na spodnie. Dlatego bardzo ważny jest etap projektowania programu oraz konstruowania algorytmów. Jeśli na tym etapie popełnione zostaną błędy logiczne, Twój program nie będzie działał prawidłowo, mimo np. właściwej składni.

Struktury sterujące

Zazwyczaj instrukcje programu wykonywane są jedna po drugiej, w porządku w jakim zostały napisane. Jest to wykonanie sekwencyjne. Język C++ pozwala programiście określić inny sposób wykonania programu, tzn. kolejne wykonane wyrażenie będzie inne niż następne występujące w sekwencji. Jest to tzw. przekazywanie sterowania. Wszystkie programy mogą być napisane w oparciu o trzy struktury sterujące: sekwencji, wyboru, powtórzenia. Struktura sekwencji jest wbudowana w C++. Dopóki nie jest powiedziane inaczej, instrukcje wykonywane są jedna po drugiej. C++ daje programiście trzy struktury wyboru: instrukcja if, instrukcja if/else oraz switch. Są również trzy struktury powtórzenia: while, do/while, for. Daje to nam w sumie

siedem struktur sterujących. Słowa te są słowami kluczowymi i nie mogą być one użyte jako identyfikatory, a także dla nazw zmiennych.

Struktury wyboru

Struktura if

Instrukcja if ma następującą składnię:

```
if(warunek)
{
  instrukcja;
  instrukcja;
  instrukcja;
  ...;
}
```

Jeśli warunek jest prawdziwy - instrukcje wewnątrz nawiasów klamrowych są wykonywane. Jeśli warunek jest fałszywy - instrukcje te są pomijane. Jeśli po instrukcji if ma być wykonana tylko jedna instrukcja można pominąć nawiasy klamrowe, np.:

```
if(ocena>=3)
  cout<<”Zdales”;
```

Struktura wyboru if jest stosowana wtedy, gdy chcemy skorzystać z alternatywnych sekwencji wykonania programu. Załóżmy taki warunek:

Jeśli ocena jest równa lub większa niż 3
Wyświetl “Zdałeś”

Jeśli warunek jest prawdziwy (true), wyświetlone zostaje „Zdałeś”, a dopiero potem zostaje wykonane następne wyrażenie w programie. Jeśli natomiast warunek jest fałszywy (false), instrukcja „Wyświetl “Zdałeś”” jest pomijana i wykonywana jest kolejna instrukcja. Przełóżmy teraz to, co napisałem na język C++:

```
// ocena.cpp - struktura wyboru if.
#include <iostream>
#include<conio.h>
```

```
using namespace std;
int ocena;
int main()
{
 cout<<”Wprowadz ocene :”<<endl;
 cin>>ocena;

 if(ocena>2) cout<<”Zdales!!!”<<endl;
 //tutaj mogą następować inne instrukcje programu
 getch();
 return 0;
}
```

Struktura if/else

Struktura if/else ma następującą składnię:

```
if(warunek)
{
 instrukcja;
 instrukcja;
 instrukcja;
 ...;
}
else
{
 instrukcja;
 instrukcja;
 instrukcja;
 ...;
}
```

Jeśli warunek jest prawdziwy (true), instrukcje wewnątrz nawiasów klamrowych są wykonywane, jeśli warunek jest fałszywy (false) - wykonywane są instrukcje wewnątrz nawiasów klamrowych po instrukcji else, np.:

```
if(ocena>=3)
  cout<<”Zdales”;
else
  cout<<”Nie zdales”;
```

Jeśli do wykonania jest tylko jedna instrukcja, to można pominąć nawiasy klamrowe.
W przypadku, gdy warunek będzie prawdziwy, tzn. wartość zmiennej ocena będzie równa lub większa od trzech, zostanie wyświetlony napis “Zdales”, a dopiero wówczas zostaną wykonane kolejne instrukcje programu. Jeżeli warunek będzie fałszywy, tzn. wartość zmiennej ocena będzie mniejsza od trzech, wyświetlony będzie napis “Nie zdales”, po tym będą wykonywane kolejne instrukcje programu. Oprócz instrukcji if/else C++ dostarcza operator warunkowy (?:). Jest to jedyny operator trójargumentowy w C++. Pierwszy operand jest warunkiem, drugi jest wartością dla całego wyrażenia warunkowego jeśli warunek jest prawdziwy, a trzeci jest wartością dla całego wyrażenia, jeśli warunek jest fałszywy, np.:

```
  cout<<(ocena>=3 ? “Zdales” : “ Nie zdales”);
```

Struktura wyboru switch

Składnia instrukcji:

```
switch (wyrażenie sterujące)
{
  case etykieta:
    instrukcja;
    ...;
  break ;
  case etykieta:
    instrukcja ;
    ... ;
  break;
   ...
  default:
    instrukcja ;
    ... ;
}
```

Struktura switch składa się z serii przypadków case i opcjonalnego przypadku default. Po słowie kluczowym switch następuje wyrażenie sterujące (jest nim np. zmienna ujęta w nawiasy). Wartość tego wyrażenia jest porównywana z każdą z etykiet case. Jeśli wynik porównania jest prawdziwy, wykonane zostaną instrukcje po tym przypadku case. Instrukcja break powoduje, że wykonanie programu jest kontynuowane od pierwszej instrukcji po strukturze switch. Jeżeli nie byłoby nigdzie w strukturze switch instrukcji break, to po wystąpieniu dopasowania instrukcje wszystkich pozostałych przypadków case zostaną wykonane. Jeśli nie wystąpi dopasowanie w żadnym z przypadków case, przypadek domyślny default zostanie wykonany. Każdy przypadek case może zwierać jedną lub więcej instrukcji.
W przypadku case nie są wymagane nawiasy klamrowe, gdy wykonane ma być wiele instrukcji. Instrukcja switch może być stosowana tylko do testowania stałych wyrażeń całkowitych. Oto praktyczne zastosowanie struktury switch:

```
// ocena.cpp - struktura wyboru switch.

#include <iostream>

using namespace std;

int main()
{
 unsigned short int ocena;
 cout<<”Wprowadz ocene :\n”;
 cin>>ocena;
 switch(ocena)
 {
  case 1:
   cout<<”Pala\n”;
  break;
  case 2:
   cout<<”Bardzo slabo\n”;
  break;
  case 3:
   cout<<”Trojka to dobra ocena\n”;
  break;
  case 4:
   cout<<”Czworka to bardzo dobra ocena\n”;
  break;
```

```
  case 5:
    cout<<”Piatka to super ocena\n”;
  break;
  case 6:
    cout<<”Szostka to najfajniejsza ocena\n”;
  break;
  default:
    cout<<”To nie jest ocena\n”;
 }

 //tutaj mogą następować inne instrukcje programu

 return 0;
}
```

Po wprowadzeniu wartości zmiennej poprzez instrukcję cin>>ocena; struktura switch zostaje uruchomiona. Wartość wyrażenia sterującego, mającego postać (ocena) porównywana jest z każdą z etykiet case. Jeśli użytkownik wprowadził np. 4, wystąpi dopasowanie (case 4:) i instrukcje dla tego przypadku zostaną wykonane (cout<<”Czworka to bardzo dobra ocena\n”;); wyświetlony zostanie napis “Czworka to bardzo dobra ocena” i nastąpi wyjście ze struktury switch bezpośrednio po instrukcji break;

Struktury powtórzenia

Struktura powtórzenia while

Składnia instrukcji:

```
while(warunek)
{
 instrukcja;
 instrukcja;
 ...;
}
```

Podobnie jak w instrukcji if, jeśli po while ma być tylko jedna instrukcja, to można pominąć nawiasy klamrowe. Dopóki warunek będzie prawdziwy (true), instrukcje wewnątrz nawiasów klamrowych będą powtarzane. Jeśli warunek stanie się fałszywy (false) - powtarzanie kończy

się i wykonywana jest kolejna instrukcja programu. Do częstych błędów należy tworzenie nieskończonych pętli, tj. takich, w których wyrażenie warunkowe nigdy nie staje się fałszywe. Aby zobaczyć działanie struktury powtórzenia while w działaniu napiszemy teraz krótki program, służący obliczaniu średniej arytmetycznej klasy. Przed przystąpieniem do pisania programu musimy się zastanowić jakie instrukcje i w jakiej kolejności program ma wykonać. Algorytm będziemy formułowali niejako “od góry” czyli od ogółu do szczegółu. Napiszmy więc co program ma robić:

1. Oblicz średnią arytmetyczną klasy.

Niestety taki stopień uogólnienia nie sposób przełożyć na C++. Spróbujmy napisać trochę bardziej szczegółowy algorytm obliczania średniej:

1. Inicjuj zmienne.
2. Wprowadź i zsumuj stopnie.
3. Oblicz i wyświetl średnią.

Niestety, aby w łatwy sposób napisać program w C++, musimy algorytm obliczania średniej jeszcze bardziej uszczegółowić:

1. Ustaw średnią na zero.
2. Ustaw liczbę uczniów.
3. Ustaw licznik na jeden.
4. Ustaw wynik na zero.
5. Jeśli licznik jest mniejszy lub równy liczbie uczniów:
    1. Wprowadź następny stopień.
    2. Dodaj stopień do wyniku.
    3. Dodaj jeden do licznika.
6. Ustaw średnią klasy na wynik podzielony przez liczbę uczniów.
7. Wyświetl średnią klasy.

Z takim stopniem szczegółowości możemy przystąpić do pisania programu w języku C++. Oto “przełożony” na język C++ algorytm obliczania średniej:

```
// srednia.cpp struktura while .

#include <iostream>
```

```
using namespace std;

int main()
{
 double srednia=0;
 int stopien, licznikPetli, wynik, liczbaUczniow;

 stopien=0;
 licznikPetli=1;
 wynik=0;
 liczbaUczniow = 0;

 cout<<”Ilu jest uczniow w Twojej klasie?”<<endl;
 cin>>liczbaUczniow;

 while(licznikPetli<=liczbaUczniow)
 {
  cout<<”Wprowadz stopien “<<licznikPetli<<” ucznia”<<endl;
  cin>>stopien; // pojedynczy stopień
  wynik=wynik+stopien;
  licznikPetli=licznikPetli+1;
 }

 srednia=static_cast<double>(wynik)/liczbaUczniow;
 cout<<”Srednia w Twojej klasie wynosi\t”<<srednia;

 return 0;
}
```

Zgodnie z naszym planem program najpierw inicjuje zmienne. Zmienna średnia została zadeklarowana jako double, czyli liczba z miejscami dziesiętnymi. Zmienne wykorzystywane do przechowywania wyników, powinny być zwykle inicjowane wartością zero przed ich użyciem. Jeśli zmienna nie zostanie zainicjowana przed jej użyciem, będzie zawierać poprzednią wartość przechowywaną w komórkach pamięci. Zmienne licznika są z reguły inicjowane wartością zero lub jeden w zależności od ich użycia. Niezainicjowane zmienne przechowują błędne wartości (tzw. niezdefiniowaną wartość) ostatnio przechowywaną pod adresem zarezerwowanym dla tej zmiennej. Według konwencji przyjętej przez wielu programistów, zmienne inicjowane są w ten sposób, że każda zmienna jest inicjowana w oddzielnej linii. Poprawia to czytelność

programu. Dobrze jest, gdy przyjmiemy pewne reguły, co do zasad nazywania zmiennych i ściśle się ich trzymamy. Ułatwia to późniejsze testowanie i wykrywanie błędów w programie. Warto nadawać zmiennym nazwy zgodne z ich przeznaczeniem. Ja preferuję rozpoczynanie nazw zmiennych z małej litery. Po inicjacji zmiennych program prosi użytkownika o podanie liczby uczniów jego klasie. Wartość wpisana przez użytkownika jest przypisywana do zmiennej "liczbaUczniow". W ten sposób zmienna "liczbaUczniow" jest inicjowana wartością wpisywaną przez użytkownika. Następnie program rozpoczyna pętlę while. Dopóki warunek jest prawdziwy, tzn. "licznikPetli" jest mniejszy lub równy zmiennej "liczbaUczniow", instrukcje znajdujące się w nawiasach klamrowych są wykonywane. Program prosi użytkownika o podanie stopnia ucznia poprzez kaskadową instrukcję cout<<"Wprowadz stopien "<<licznikPetli<<" ucznia\t";. Używa on zmiennej "licznikPetli" do wyświetlania numeru ucznia, którego stopień chcemy wprowadzić. Następnie instrukcja wynik=wynik+stopien; dodaje do zmiennej wynik wartość zmiennej stopien, wówczas wartość tego działania przypisywana jest do zmiennej wynik. Po tym zmienna "licznikPetli" jest zwiększana o jeden. Po tym warunek pętli while jest ponownie sprawdzany. Jeśli jest on fałszywy, będzie wykonywana następna instrukcja po nawiasach klamrowych. Jak wiemy z poprzedniego odcinka kursu, wynik z dzielenia całkowitego i jest liczbą całkowitą, średnia tymczasem nie zawsze jest całkowita. Chcąc wytworzyć zmiennoprzecinkowe obliczenie z wartościami całkowitymi, musimy utworzyć tymczasowe wartości, które są liczbami zmiennoprzecinkowymi (tzn. z miejscami dziesiętnymi). C++ dostarcza jednoargumentowy operator rzutowania, abyśmy mogli to wykonać (static_cast<typ danych>(zmienna)).

Użycie operatora rzutowania jest nazywane jawną konwersją. Wartość przechowywana w zmiennej wynik jest nadal liczbą całkowitą. Natomiast obliczenie z wartości zmiennoprzecinkowej wartości (tymczasowej wersji zmiennej wynik) podzielonej przez całkowitą wartość zmiennej "liczbaUczniow". Kompilator C++ zna informację tylko o sposobie obliczania wyrażeń, w których typy danych operandów są identyczne. Aby móc przeprowadzić wyrażenie z mieszanymi operandami, przeprowadzana jest promocja (zwana konwersją niejawną) na wybranych operandach, np. jeżeli mamy jeden operand typu double, a drugi typu int operandy typu int są promowane do double. Dokładne reguły promocji zostaną omówione w następnych odcinkach kursu. Po przeprowadzeniu jawnej konwersji zmiennej wynik i niejawnej konwersji zmiennej "liczbaUczniow" przeprowadzane jest działanie dzielenia na tymczasowych zmiennoprzecinkowych wartościach tych zmiennych i wynik tego działania przypisywany jest do zmiennej średnia. Po tym program wyprowadza wartość średniej na ekran.

Struktura powtórzenia for

Struktura ta ma składnię:

```
for (inicjacja zmiennej; warunek kontynuacji pętli; inkrementacja/dekrementacja zmiennej)
{
 instrukcja;
 instrukcja;
 ...;
}
```

Struktura powtórzenia for jest doskonałym narzędziem wszędzie tam, gdzie zachodzi potrzeba powtarzania kontrolowanego licznikiem. W strukturze for wymagane są dwa średniki. Najlepiej będzie, gdy wyjaśnię na przykładzie jak działa ta struktura.

```
// for.cpp Powtarzanie kontrolowane licznikiem

#include <iostream>

using namespace std;
int main()
{
  for(int licznik=1;licznik<=100;licznik++)
    cout<<licznik<<”\n”;

  return 0;
}
```

Gdy struktura for rozpoczyna wykonywanie najpierw deklarowana jest (jako typ int) i inicjowana (wartością 1) zmienna licznik. Po tym sprawdzany jest warunek kontynuacji pętli. Ponieważ wartość zmiennej licznik jest 1, warunek jest prawdziwy, więc wyświetlane jest 1. Następnie zmienna licznik jest inkrementowana (licznik++). Więcej o operatorach inkrementacji dowiecie się w dalszej części kursu. Cały ten proces jest kontynuowany, aż zmienna licznik osiągnie wartość 100. Dla struktury for nie ma znaczenia czy użyjesz pre- czy postinkrementacji. Wartość licznika zwiększana jest dopiero, gdy ciało for zostanie wykonane.

Struktura powtórzenia do/while

Struktura do/while jest podobna do struktury while. W strukturze while warunek kontynuacji pętli jest sprawdzany, zanim jej ciało zostanie wykonane, natomiast w do/while sprawdza warunek kontynuacji pętli po tym, jak ciało pętli zostaje wykonane. Wynika z tego, że ciało zostanie wykonane przynajmniej jeden raz. Składnia instrukcji:

```
do
{
 instrukcja;
 instrukcja;
 ...;
}
while(warunek);
```

Kiedy warunek jest fałszywy wykonanie programu jest kontynuowane od pierwszej instrukcji po while.

Instrukcje break i continue

Instrukcja break, gdy jest wykonywana w strukturze while, for, do/while, switch, powoduje bezpośrednie wyjście z tej struktury.

```
//break.cpp przykład użycia break
#include <iostream>

using namespace std;

int main()
{

 for(int a=1;a<=20;a++)
 {
  if(a==10)
    break;

  cout<<a<<" ";
 }
```

```
 return 0;
}
```

Gdy wartość zmiennej a osiągnie wartość 10 pętla for zostanie przerwana.
Wyrażenie continue, gdy jest wykonywane w strukturze while, for, do/while, switch. pomija pozostałe wyrażenia w ciele tej struktury i kontynuuje następną iterację.

```
//continue.cpp przykład użycia break
#include <iostream>

using namespace std;

int main()
{

 for(int a=1;a<=20;a++)
 {
   if(a==10)
     continue;

   cout<<"a="<<a<<"\t";
 }

 return 0;
}
```

Gdy wartość zmiennej a będzie się równała 10 dalsze wyrażenia w pętli for zostaną pominięte, tzn. program wyświetli wszystkie wartości zmiennej a od 1 do 20 z pominięciem wyświetlenia wartości 10.

Operatory przypisania

C++ zwiera różne operatory, których celem jest skrócenie wyrażeń przypisania, np.:

```
a=a*3;
```

może zostać napisane jako :

```
a*=3;
```

Możliwości graficzne programu może zostać przekształcone do zmienna operator=wyrażenie; przy czym operator musi być jednym z operatorów dwuargumentowych +, -, *, /, %.

Operatory inkrementacji i dekrementacji

W C++ wbudowane są też jednoargumentowe operatory inkrementacji i dekrementacji.
Wyrażenie w postaci ++a inkrementuje zmienną a o 1, a następnie używa nowej wartości a w wyrażeniu w którym jest a. Wyrażenie w postaci a++ najpierw używa a w wyrażeniu, w którym a występuje, a następnie zwiększa a o jeden.
Wyrażenie w postaci --a zmniejsza a o jeden, a następnie używa nowej wartości a w wyrażeniu, w którym a występuje. Wyrażenie w postaci a-- najpierw wykorzystuje a w wyrażeniu, w którym a występuje, a potem zmniejsza a o jeden.

Operatory logiczne

C++ dostarcza również operatorów logicznych, aby umożliwić nam formułowanie bardziej złożonych warunków. C++ sprawdza pod względem prawdy lub fałszu wszystkie wyrażenia, które zawierają operatory relacyjne, równości oraz operatory logiczne.

Operator iloczynu logicznego(AND)

Operator iloczynu logicznego zapisujemy: &&. Tabela prawdy dla tego operatora:

Wyrażenie1	Wyrażenie2	Wyrażenie1 & Wyrażenie2
false	false	False
false	true	False
true	false	False
true	true	True

Czyli np. mamy wyrażenie:

```
if((a<10) && (b==5))
```

Jeśli wartość pierwszego wyrażenia i drugiego wyrażenia będzie miało wartość true, wartość całego wyrażenia będzie miało wartość true. Jeśli którekolwiek z tych wyrażeń będzie miało wartość false, wartość całego wyrażenia będzie miało wartość false. Operator iloczynu logicznego stosujemy wtedy, gdy chcemy się upewnić, że dwa warunki są prawdziwe.

Operator sumy logicznej (OR)

Operator sumy logicznej zapisujemy: ||. Tabela prawdy dla tego operatora pokazuje wszystkie możliwe kombinacje:

Wyrażenie1	Wyrażenie2	Wyrażenie1 || Wyrażenie2
false	false	False
false	true	True
true	false	True
true	true	True

Logiczne OR stosujemy wtedy, gdy chcemy się upewnić, że jeden lub dwa warunki są prawdziwe.

Operator negacji logicznej

Operator negacji logicznej ! jest, w przeciwieństwie do operatorów && i ||, operatorem jednoargumentowym. Operator ten ma tylko jeden warunek jako wyrażenie. Jest on umieszczany przed warunkiem, np. :

```
if( !(wyrażnie==wyrażenie))
```

Możemy to zapisać również za pomocą operatora relacyjnego != :

```
 if(wyrażenie!=wyrażenie)
```

Tabela prawdy dla tego operatora:

Wyrażenie	!Wyrażenie
true	false
false	true

Ćwiczenia

1. Napisz program, który obliczy średnie zużycie paliwa. Program powinien pobierać liczbę przejechanych kilometrów i zatankowanych litrów przy każdym tankowaniu. Program powinien obliczyć i wyświetlić zużycie paliwa przy każdym tankowaniu, a także dla wszystkich tankowań.
2. Napisz program wyświetlający potęgi liczby 2 (2,4,8,16 ..).#
3. Napisz program odczytujący promień koła i obliczający oraz wyświetlający średnicę, obwód oraz pole.

4. Napisz program odczytujący 3 niezerowe wartości zmiennoprzecinkowe (float) oraz określający i wyświetlający informację, czy mogą one stanowić długość boków trójkąta.
5. Trójkąt prostokątny może mieć boki, których długości są liczbami całkowitymi. Trójka pitagorejska musi spełniać jeden warunek: suma kwadratów dwóch boków musi być równa kwadratowi przeciwprostokątnej. Znajdź wszystkie trójki pitagorejskie dla wartości nie dłuższych niż 500.

Kurs część 3

Funkcje

Wstęp

Programy komputerowe są tworami bardzo złożonymi. Umieszczenie wszystkich instrukcji w funkcji main() byłoby niewygodne z kilku powodów: po pierwsze wykrycie ewentualnych błędów byłoby utrudnione, po drugie instrukcje, które powtarzałyby się, trzeba byłoby wpisywać ponownie. Funkcje pozwalają programiście na budowanie programu z mniejszych części oraz ponowne wykorzystywanie tych komponentów. Programista może pisać funkcje do wykonywania określonych zadań. Mogą one być wykonywane w wielu miejscach w programie. Nic nie stoi na przeszkodzie, aby funkcje były wykonywane w różnych programach. Funkcje są wywoływane. Wywołanie funkcji określa jej nazwę oraz argumenty przekazywane do funkcji.

Prototypy funkcji

Programy z poprzednich części kursu zawierały wywołania do funkcji z biblioteki standardowej. Teraz nauczymy się pisania własnych funkcji, które będą mogły być wykorzystywane w naszych programach. Zanim wywołamy funkcję musimy przekazać kompilatorowi informację o nazwie funkcji, argumentach przez nią pobieranych oraz o zwracanej wartości. Kompilator używa tej informacji do sprawdzania wywołań funkcji, tzn. czy argumenty jakie przekazujemy do funkcji są odpowiednie itp. Pierwsze wersje kompilatorów nie przeprowadzały tego typu sprawdzenia, co prowadziło do częstych błędów.

Mały przykład:

```
#include <iostream>
using namespace std;
//prototyp funkcji max
int max(int,int,int );
```

```
int main()
{
  int x, y, z;
  cout<<”Wprowadz trzy liczby calkowite :”<<endl;
  cin>>x>>y>>z;
  cout<<”Najwieksza z tych liczb to :”<<max(x,y,z)<<endl;

return 0;
}

//definicja funkcji max
int max(int a,int b,int c)
{
 int max=a;
 if(b>max)
  max=b;
 if(c>max)
  max=c;

 return max;
}
```

Powyższy program pobiera trzy liczby całkowite od użytkownika i określa, która z nich jest największa. Linia:

```
//prototyp funkcji max
int max(int,int,int );
```

zawiera prototyp funkcji max. Pierwszy wyraz z lewej - int - określa typ wartości zwracany przez funkcję, w tym wypadku mamy liczbę całkowitą. Kolejny wyraz jest identyfikatorem (nazwą) funkcji. Wyrazy umieszczone w nawiasach określają typ argumentów pobieranych przez funkcję - są to trzy liczby całkowite. Prototyp mógłby również wyglądać tak:

```
//prototyp funkcji max
int max(int a, int b, int c );
```

przy czym identyfikatory argumentów w prototypach są pomijane przez kompilator. Można umieszczać je tam dla celów dokumentacyjnych. Kompilator odwołuje się do prototypu funkcji,

aby sprawdzić czy wywołanie funkcji max zawiera poprawny zwracany typ, poprawną liczbę argumentów, poprawne ich typy i porządek. Prototyp funkcji nie jest wymagany, jeżeli funkcja pojawiła się przed pierwszym wywołaniem, tzn. jeżeli ciało funkcji max() umieścilibyśmy przed main(), prototyp funkcji max() można pominąć. Część prototypu funkcji, która zawiera nazwę funkcji i typy jej argumentów, nazywana jest sygnaturą. Gdybyśmy prototyp funkcji określili jako:

```
void max(int, int, int );
```

kompilator wygenerowałby błąd, ponieważ zawracany typ void w prototypie nie zgadzałby się z tym w nagłówku. Ważną cechą prototypów jest koercja argumentów. Polega to na tym, że wartości argumentów, które nie odpowiadają dokładnie typom parametrów w prototypie funkcji, są przekształcane do właściwego typu zanim funkcja zostanie wywołana. Przekształcenia czasami mogą prowadzić do błędów, jeżeli reguły promocji nie zostaną zachowane. Reguły te określają jak typy danych mogą zostać przekształcone do innych typów bez utraty danych, np. gdybyśmy do funkcji max() przekazali double jako argument ułamkowa, część double byłaby obcięta. Reguły promocji stosuje się też do wyrażeń mieszanych, czyli takich, które zwierają dwa lub więcej typów danych. Oto tabela hierarchii promocji:

long double
double
float
unsigned long
long
unsigned int
int
unsigned short
short
unsigned short
short
char

Przekształcanie z wyższego typu danych w hierarchii do niższego, może prowadzić do błędów.

Definicja funkcji

Funkcja max jest wywoływana w funkcji main w linii:

```
cout<<”Najwieksza z tych liczb to :”<<max(x,y,z)<<endl;
```

Funkcja ta otrzymuje kopie wartości x, y, z w parametrach a, b, c. Następnie porównuje ona ze sobą argumenty i zwraca największy z nich. Wynik jest przesyłany do main(), gdzie funkcja była wywołana. Definicja funkcji max() jest w liniach:

```
 //definicja funkcji max
int max(int a,int b,int c)
{
  int max=a;
  if(b>max)
   max=b;
  if(c>max)
   max=c;
  return max;
}
```

Linia:

```
int max(int a,int b,int c)
```

Wyraz int najbardziej z lewej strony oznacza, że funkcja zwraca wynik całkowity (integer). Po nim następuje identyfikator oznaczający nazwę funkcji, w nawiasie podane są parametry, jakich oczekuje funkcja oraz ich nazwy. Definicja funkcji ma więc postać:

```
typ_zwracanej_wartości nazwa_funkcji (lista_parametrów)
{
ciało funkcji;
}
```

typ_zwracanej_wartości jest typem danych zwracanych przez funkcję. Typ void oznacza, że funkcja nie zwraca wartości. W C++ typ wartości zwracanej musi być zawsze określony (nawet jeśli funkcja nic nie zwraca, wówczas musimy zasygnalizować to kompilatorowi słowem kluczowym void). nazwa_funkcji jest dowolnym dozwolonym identyfikatorem. Lista_ parametrów jest to lista oddzielonych przecinkami deklaracji parametrów otrzymywanych przez funkcję. Jeśli funkcja nie otrzymuje żadnych argumentów lista_parametrów jest typu void lub jest pusta, np.:

```
void max(void)
void max()
```

Typ parametru musi być wyraźnie napisany przed każdym parametrem. Po nawiasach () nie umieszczamy średnika. Są trzy sposoby na opuszczenie ciała funkcji:
a. jeśli funkcja nie zwraca wartości, sterowanie jest zwracane wtedy, gdy osiągnięty zostanie prawy nawias kończący funkcję,
b. jeśli funkcja nie zwraca wartości, sterowanie jest zwracane przez wykonanie wyrażenia return,
c. jeśli funkcja zwraca wartość to wyrażenie return wyrażenie.

Argumenty domyślne

Programista może określić domyślną wartość argumentu. Kiedy argument jest pominięty w wywołaniu, jego wartość jest wstawiana przez kompilator i przekazywana w wywołaniu. Argumenty te powinny być położone jak najbardziej z prawej strony. Argumenty domyślne powinny być określone wraz z pierwszym wystąpieniem nazwy funkcji, z reguły będzie to prototyp.

```
#include <iostream>

using namespace std;
//prototyp funkcji poleProstokata

int poleProstokata( int=1,int=1 );

int main()
{
 int x, y;
 cout<<”Wprowadz dwie liczby calkowite :”<<endl;
 cin>>x>>y;
 cout<<”Pole prostokata wynosi :”<<poleProstokata(x,y)<<endl;
 cout<<”Pole prostokata z argumentami domyslnymi :”<<poleProstokata()<<’\n’
   <<”Pole prostokata z jednym argumentem domyslnym:”<<poleProstokata(x)<<endl;

 return 0;
}

//definicja funkcji poleProstokata
int poleProstokata(int a,int b)
{
```

```
 return a*b;
}
```

Klasy pamięci

Każdy identyfikator ma atrybuty obejmujące : klasę pamięci, zasięg i połączenia. Do określenia klasy pamięci w C++ służą specyfikatory klas pamięci. C++ zawiera cztery specyfikatory klas pamięci: auto, register, extern, static. Klasa pamięci identyfikatora określa czas, w którym identyfikator znajduje się w pamięci. Niektóre identyfikatory istnieją bardzo krótko, a inne przez cały czas wykonywania programu. Zasięg identyfikatora określa z jakiego miejsca w programie można się odwołać do identyfikatora. Do niektórych identyfikatorów można się odwołać z każdego miejsca w programie (np. zmienne globalne), a do innych tylko z niektórych części. Połączenia identyfikatorów określają czy w programach złożonych z wielu plików źródłowych identyfikator jest znany tylko w bieżącym pliku źródłowym, czy w każdym. Specyfikatory klas pamięci mogą być podzielone na dwie klasy:

1. statyczną,
2. automatyczną.

Tylko zmienne mogą być elementami automatycznych klas pamięci. Zmienne lokalne i parametry funkcji są zwykle elementami automatycznych klas pamięci. Specyfikator auto wyraźnie deklaruje zmienne automatycznej klasy pamięci. Zmienne tego typu są tworzone podczas wykonywania bloku, w którym są deklarowane i są niszczone, gdy następuje wyjście z niego. Zmienne lokalne są domyślnie automatycznej klasy pamięci, więc słowo kluczowe auto jest rzadko używane. Specyfikator klasy pamięci register może być umieszczony przed deklaracją zmiennej automatycznej. Specyfikator ten oznacza, że kompilator powinien umieszczać tą zmienną raczej w rejestrze procesora. Kompilator może ignorować deklaracje register.

Słowa kluczowe extern i static są używane do deklarowania zmiennych i funkcji statycznej klasy pamięci. Zmienne takie istnieją od momentu, w którym program rozpoczyna wykonywanie. Pamięć jest im przydzielana i inicjowana tylko raz. Globalne zmienne i funkcje mają domyślnie klasę pamięci extern. Są one tworzone poprzez umieszczenie ich poza jakąkolwiek funkcją. Do zmiennych tych i funkcji może się odwoływać każda funkcja w pliku. Zmienne używane tylko w określonej funkcji powinny być deklarowane jako zmienne lokalne. Deklarowanie zmiennych jako globalne utrudnia wykrywanie błędów... Zmienne lokalne zadeklarowane za słowa static są nadal znane tylko w funkcji, w której zostały zadeklarowane, inaczej od zmiennych automatycznych zachowują swoje wartości po wyjściu z funkcji. Następnym razem przy wywołaniu funkcji zmienne te mają taką wartość, jaką miały przy wyjściu z funkcji.

```
#include <iostream>

using namespace std;
//prototyp fukncji zmienneStatyczne
int zmienneStatyczne( );
int main()
{
 for(int i=0;i<10;i++)
 cout<<i<<". wywolanie funkcji zmienneStatyczne() "<<zmienneStatyczne()<<'\n';
 return 0;
}
//definicja funkcji zmienneStatyczne
int zmienneStatyczne( )
{
 static int a=0;
 a++;
 return a;
}
```

Reguły zasięgu

Reguły zasięgu określają z jakiego miejsca w pliku można się odwołać do danego identyfikatora. Pięcioma zasięgami dla identyfikatora są: zasięg funkcji, zasięg pliku, zasięg bloku, zasięg prototypu funkcji oraz zasięg klasy.
Identyfikator zadeklarowany poza jakąkolwiek funkcją ma zasięg pliku. Można do niego odwoływać się od miejsca, w którym został on zadeklarowany, aż do końca pliku. Etykiety są identyfikatorami mającymi zasięg funkcji. Mogą one być używane gdziekolwiek w funkcji, w której się pojawiają, ale nie można się do nich odwołać spoza ciała funkcji. Etykiety są używane w poleceniach switch i goto. Identyfikatory zadeklarowane wewnątrz bloku mają zasięg bloku. Zasięg bloku rozpoczyna się w miejscu deklaracji identyfikatora i kończy się po osiągnięciu nawiasu klamrowego - } - . Zasięg bloku mają zmienne lokalne zdefiniowane na początku funkcji, a także parametry funkcji.

Jedynymi identyfikatorami o zasięgu prototypu funkcji są identyfikatory użyte na liście jej parametrów.

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

```
//zmienna globalna
int a=1;
void funkcja();
int main()
{
 //zmienna lokalna w main
 int a=5;
 cout<<"Zmienna lokalna w main :"<<a<<'\n';
 //nowy blok
 {
  int a=10;
  cout<<"Zmienna lokalna w bloku :"<<a<<'\n';
 }
 funkcja();
 cout<<"Zmienna lokalna w main :"<<a<<'\n';
 return 0;
}
//definicja funkcji
void funkcja( )
{
 cout<<"Zmienna globalna to :"<<a<<'\n';
 return;
}
```

Rekurencja

Funkcja rekurencyjna jest to funkcja, która wywołuje bezpośrednio sama siebie lub pośrednio przez inną funkcję. Funkcja rekurencyjna jest wywoływana do rozwiązania określonego problemui z reguły wie, jak rozwiązać najprostszy przypadek. Jeśli wywoływana jest do problemu złożonego, dzieli go na dwa elementy: ten, który potrafi rozwiązać i ten, którego nie potrafi rozwiązać. Ten drugi element jest nieznacznie upraszczany. Funkcja wywołuje swoją kopię do pracy z tym uproszczonym elementem. Nazywane jest to krokiem rekurencji. Gdy problem zostaje uproszczony do przypadku podstawowego, wywołania rekurencyjne kończą się i funkcja zwraca wartość. Typowym problemem dla rekurencji może być silnia oraz szereg Fibonacciego. Silnia nieujemnej liczby całkowitej pisana jest n! (wymawiana jako “n silnia”). Jest to iloczyn:

n*(n-1)*(n-2)*...*1

przy czym 1! jest równe 1 i 0! jest równe 1. Dla przykładu 3!=3*2*1. Rekurencyjna silnia ma postać:

n!=n*(n-1)!

np.:

3!=3*(2!)
2!=2*(1!)
1!=1*(0!)
0!=1

Oto program obliczający silnię z liczby podanej przez użytkownika:

```
#include <iostream>
using namespace std;
//prototyp funkcji
long silniaRekurencyjnie(long);

int main()
{
 long a;
 cout<<”Wprowadz liczbe calkowita \n”;
 cin>>a;
 cout<<”Silnia liczby “<<a<<” wynosi :”<<silniaRekurencyjnie(a)<<endl;
 return 0;
}
//definicja funkcji
long silniaRekurencyjnie(long liczba)
{
 //przypadek podstawowy
 if(liczba<=1)
  return 1;
 else //przypadek złożony upraszczamy nieznacznie i funkcja wywołuje samą siebie
 return liczba*silniaRekurencyjnie(liczba-1);
}
```

Funkcja silniaRekurencyjnie() została zadeklarowana jako funkcja oczekująca parametru

typu long oraz zwracająca wynik typu long. Proponuję Wam, abyście zmodyfikowali tak ten program, aby było widać, jakie argumenty funkcja otrzymuje oraz jaką wartość zwraca.
Szereg Fibonacciego rozpoczyna się od 0 i 1 i charakteryzuje się tym, że kolejna liczba jest sumą dwóch poprzednich. Stosunek kolejnych liczb Fibonacciego jest równy w przybliżeniu 1.618. Liczba ta nazywana jest złotym podziałem. Szereg Fibonacciego może być przedstawiony rekurencyjnie:

fibonacci(0)=0
fibonacci(1)=1
fibonacci(n)=fibonacci(n-1)+fibonacci(n-2)

Oto program obliczający szereg Fibonacciego rekurencyjnie:

```
#include <iostream>

using namespace std;

//prototyp funkcji
long fibonacci(long);

int main()
{
 long a;
 cout<<”Wprowadz liczbe calkowita \n”;
 cin>>a;
 cout<<”fibonacci (“<<a<<”) wynosi :”<<fibonacci(a)<<endl;
 return 0;
}

//definicja funkcji
long fibonacci(long liczba)
{
 //przypadek podstawowy
 if(liczba<=0)
  return 0;
 if(liczba==1)
  return 1;
 else //przypadek złożony dzielimy na dwie części
```

```
 return fibonacci(liczba-1)+fibonacci(liczba-2);
}
```

Rekurencja ma wiele wad. Obliczenie 20 liczby Fibonacciego wymagać będzie 2^20 wywołań (ok. miliona wywołań !). Będzie to oczywiście znacznie obciążać zasoby komputera w efekcie może to doprowadzić do przepełnienia stosu. Dlatego rekurencje należy stosować wówczas, gdy liczba wywołań nie będzie zbyt wielka. Rekurencja ma też zalety, podstawową zaletą jest niejako “naturalny” sposób rozwiązywania problemów. W przypadku, gdy złożoność obliczeniowa naszego rekurencyjnego algorytmu jest zbyt duża, należy upraszczać rekurencję do wyrażenia nie będącego rekurencyjnym.
Drobny komentarz: Ciągu Fibonacciego nie da się obliczyć rekurencyjnie, gdyż razem ze wzrostem wartości tego ciągu dokładność będzie malała. Jedyną dokładną metodą jest metoda iteracyjna.

Referencje i parametry referencji

Funkcję można wywołać na dwa sposoby: wywołanie przez wartość (ang. call by value) oraz wywołanie przez referencje. Do tej pory funkcje wywoływaliśmy poprzez wartość. Gdy wywołujemy funkcję poprzez wartość, tworzona jest kopia argumentu, który przekazujemy w wywołaniu. Kopia ta jest przekazywana do funkcji. Wszystkie działania w ciele funkcji są wykonywane na tej kopii i nie oddziaływają na oryginalną wartość zmiennej. Dodatnią stroną tego typu wywołania jest to, że zapobiega to ujemnym skutkom ubocznym. Ujemną stroną jest to, że w przypadku dużych struktur danych kopiowanie zabiera dużo czasu i pamięci.
Dzięki wywołaniu przez referencje funkcja wywołująca przekazuje funkcji wywoływanej zdolność do bezpośredniego dostępu do danych. Nie jest tworzona kopia, a wszelkie modyfikacje są dokonywane na zmiennej bezpośrednio. Jeśli parametr ma być przekazywany do funkcji przez referencje po typie parametru, a przed identyfikatorem umieść znak ampersandu (&). Oto program demonstrujący różnicę między tymi dwoma wywołaniami:

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

//prototypy funkcji
int wywolaniePrzezWartosc (int);
void wywolaniePrzezReferencje(int &);

int main()
{
```

```
 int a;
 cout<<”Wprowadz liczbe calkowita \n”;
 cin>>a;
 cout<<”a przed wywołaniem funkcji przez wartość :”<<a<<’\n’;
 cout<<”Wartość zwracana przez funkcję
 wywołaniePrzezWartość:”<<wywolaniePrzezWartosc(a)<<’\n’;
 cout<<”a po wywołaniu funkcji przez wartość :”<<a<<’\n’;
 cout<<”a przed wywołaniem funkcji przez referencje :”<<a<<’\n’;
 cout<<”Wywołanie funkcji przez referencje \n”;
 wywolaniePrzezReferencje(a);
 cout<<”a po wywołaniu funkcji przez referencje :”<<a<<endl;
 return 0;
}
//definicja funkcji
int wywolaniePrzezWartosc(int liczba)
{
 liczba=liczba*5;
 return liczba;
}

void wywolaniePrzezReferencje(int &liczba1)
{
 liczba1=liczba1*5;
}
```

Przeciążanie funkcji

W C++ można definiować kilka funkcji o tej samej nazwie, przy czym muszą się one różnić parametrami. Jest to nazywane przeciążaniem funkcji. Na przykład: mamy funkcję obliczającą pole powierzchni prostokąta. Możemy zdefiniować kilka funkcji polePowierzchni. Będą one pobierały jako argumenty różne typy (np. jedna int, druga double itp.). Kiedy funkcja przeciążona jest wywoływana kompilator C++ wybiera właściwą poprzez sprawdzenie liczby typów i porządku argumentów w tym wywołaniu. Przeciążanie funkcji jest stosowane tam, gdzie przeprowadzane są takie same obliczenia na różnych typach danych. Oto podany przykład:

```
#include <iostream>
```

```
using namespace std;

//prototypy funkcji
int polePowierzchni(int bokX,int bokY);
double polePowierzchni(double bokX,double bokY);

int main()
{
 int a=5,b=4;
 double c=5.8,d=3.2;
 cout<<"Pole powierzchni z argumentami całkowitymi "<<polePowierzchni(a,b)<<'\n';
 cout<<"Pole powierzchni z argumentami rzeczywistymi "<<polePowierzchni(c,d)<<endl;
 return 0;
}
//definicja funkcji
int polePowierzchni(int bokX,int bokY)
{
 return bokX*bokY;
}

double polePowierzchni(double bokX,double bokY)
{
 return bokX*bokY;
}
```

Ćwiczenia

1. Jaki jest zasięg zmiennej x w main.
2. Napisz funkcję określającą dla pary liczb całkowitych, czy pierwsza jest wielokrotnością drugiej.
3. Napisz funkcję main zwracającą najmniejszą z trzech liczb całkowitych.
4. Liczba jest liczbą pierwszą jeżeli dzieli się tylko przez 1 i przez samą siebie. Napisz funkcję, która określi czy dana liczba jest liczbą pierwszą.
5. Czy main() może być wywołana rekurencyjnie? Sprawdź to.
6. Największy wspólny dzielnik x i y jest największą liczbą całkowitą przez którą x i y dzielą się bez reszty. Napisz funkcję, która będzie pobierała dwa argumenty i zwracała NWD.

Kurs część 4

Tablice

Tablica jest strukturą danych. Tablica to ciągła grupa komórek w pamięci. Grupa ta posiada wspólną nazwę. Komórki pogrupowane są w typ danych, który określamy przy definicji tablicy. Do poszczególnych elementów tablicy odwołujemy się poprzez określenie nazwy tablicy i numeru pozycji odpowiedniego elementu. Elementy tablicy numerujemy od zera (pierwszy jest element zerowy). Chcąc zdefiniować tablicę 20 liczb całkowitych, piszemy:

```
int tablicaInt[20];
```

Poszczególne elementy będziemy numerować od 0 do 19. Chcąc się odwołać do pierwszego elementu tablicy, piszemy:

```
tablicaInt[0];
```

Numer elementu ujmowany w nawiasy kwadratowe nazywany jest indeksem. Indeks musi być liczbą całkowitą lub wyrażeniem całkowitym.

Deklaracja tablic

Tablice zajmują miejsce w pamięci. Przy deklaracji tablicy należy określić typ elementów tablicy oraz ich ilość. Posiadając tą informację kompilator może zarezerwować odpowiednią ilość komórek pamięci. Deklaracja tablicy ma postać:

```
typ_elementów nazwa_Tablicy [ ilość_elementów ];
```

Kilka tablic tego samego typu może być deklarowanych w tej samej linii:

```
int tablicaInt1 [20], tablicaInt [50];
```

W linii tej deklarujemy dwie tablice typu int: jedną składającą się z 20 elementów, a drugą z 50 elementów.

Inicjowanie tablic

Elementy tablicy mogą być inicjowane w deklaracji tablicy przez umieszczenie po niej znaku równości i rozdzielonej przecinkami listy wartości inicjujących ujętej w nawiasy klamrowe.

```
int tablicaInt[5]={1,2,3,4,5};
```

Jeśli wartości inicjujących będzie mniej niż elementów tablicy, pozostałe elementy będą zainicjowane wartością zero.

```
int tablicaInt[5]={5};
```

Deklaruje tablicę typu int, która ma pięć elementów. Pierwszy element tej tablicy jest jawnie inicjowany wartością 5. Pozostałe elementy tej tablicy są inicjowane niejawnie wartością zero. Do inicjacji tablic możemy również wykorzystać pętle for.

```
#include <iostream>

using namespace std;
int main()
{
 const int wTab=100;
 int tablicaInt[wTab];

 for(int i=0; i<wTab; i++)
  tablicaInt[i]=i;

 cout<<"Tablica została zainicjowana liczbami :\n";

 for(int i=0; i<wTab; i++)
  cout<<"Numer elementu :"<<i<<" wartosc elementu "<<tablicaInt[i]<<'\n';

 return 0;
}
```

Linia:

```
const int wTab=100;
```

stosuje kwalifikator const w celu zadeklarowania stałej symbolicznej wTab, której wartość wynosi 100. Stałe takie muszą być inicjowane wyrażeniem stałym i nie mogą być później modyfikowane. Stałe symboliczne nazywane są również zmiennymi tylko do odczytu. Określając liczbę elementów tablicy kompilator oczekuje wyrażenia całkowitego i stałego, dlatego możemy użyć stałej symbolicznej. Wykorzystywanie stałych symbolicznych do określenia wielkości tablic, daje programom większą skalowalność. Nic nie stoi na przeszkodzie,

aby zamiast liczby 100 elementów typu int, inicjowane było 1000 elementów poprzez zmianę stałej symbolicznej. Gdybyśmy nie używali stałej symbolicznej, musielibyśmy zmienić program w 3 miejscach.

Wykorzystywanie tablic

Zostałeś poproszony o napisanie programu, który zliczałby odpowiedzi na ankietę zamieszczoną w pewnym piśmie. Oto pytania, jakie zadano czytelnikom:

Co Twoim zdaniem powinno się znaleźć w naszym piśmie ?
1. recenzje gier,
2. opisy sprzętu,
3. kursy programowania,
4. nowości ze świata komputerów.

A oto program realizujący to zadanie:

```
#include <iostream>

using namespace std;

int main()
{
 // inicjalizacja zmiennych
 // najpierw inicjalizujemy stałą symboliczną określającą wielkość tablicy
 // potem inicjalizujemy tablicę liczb typu int o nazwie tabOdp
 // tablica ma 4 elementy

 const int wTab=4;
 int tabOdp[wTab]={0}, licz;

 // wyświetlamy informacje dla użytkownika

 cout<<"Co Twoim zdaniem powinno się znalezc w naszym pismie ?\n"
                <<"1. recenzje gier\n2. opisy sprzętu,\n3. kursy programowania,\n"
                <<"4. nowości ze świata komputerów.\nWprowadz -1 aby zakończyć.";

 cin>>licz;
```

```
// pętla odpowiedzialna za zliczanie odpowiedzi
// 1. sprawdzamy jaka wartość została wprowadzona przez użytkownika
// -jeśli użytkownik wprowadził mniej niż 1 lub więcej niż 4 to przerywamy pętlę

while(licz>0 && licz<5)
{
  cout<<"Co Twoim zdaniem powinno się znalezc w naszym pismie ?\n"
  <<"1. recenzje gier\n2. opisy sprzętu,\n3. kursy programowania,\n"
  <<"4. nowości ze świata komputerów.\nWprowadz -1 aby zakończyć."<<endl;

  // zliczenie odpowiedzi indeks tablicy jest obliczany na przez wyrażenie licz-1
  // zwiększenie elementu tablicy przez operator ++
  // wprowadzenie kolejnej odpowiedzi

  ++tabOdp[licz-1];
  cin>>licz;
}

// wyświetlamy wyniki

cout<<"Wynik ankiety :\n"
                <<"1. recenzje gier :"<<tabOdp[0]<<"\n2. opisy sprzętu :"<<tabOdp[1]
                <<"\n3. kursy programowania :"<<tabOdp[2]
                <<"\n4. nowości ze świata komputerów :"<<tabOdp[3]<<endl;
 return 0;
}
```

Generowanie liczb losowych

Do generowania liczb losowych służy funkcja rand(). Funkcja ta generuje liczbę całkowitą między 0 i RAND_MAX (stałą symboliczną zdefiniowaną w pliku nagłówkowym cstdlib). Jeśli funkcja ta wybiera liczby naprawdę losowo, to każda z nich ma taką samą szansę bycia trafioną. Zakres wartości wytwarzanych przez funkcję rand() jest bardzo duży. Zwykle nie potrzebujemy aż tylu wartości. Na przykład program, który będzie symulował rzuty kostką będzie potrzebował sześć wartości. Chcąc wytworzyć liczby całkowite w zakresie od 0 do 5 używamy dzielenia modulo (%) w połączeniu z funkcją rand:

```
1+rand()%6
```

Nazywamy to skalowaniem. Liczba 6 jest czynnikiem skalowania. A liczba 1 jest wartością przesunięcia. Wartość przesunięcia to liczba równa pierwszej liczbie w pożądanym zakresie kolejnych liczb całkowitych.

Oto program symulujący rzut kostką 20 razy:

```
#include <iostream>
#include <cstdlib>

using namespace std;

int main()
{
        for(int i=1; i<20; i++)
                cout<<(1+rand()%6)<<'\n';

        cout<<endl;

        return 0;
}
```

Jeśli uruchomisz ten program kilka razy zobaczysz, że wyświetlana jest taka sama sekwencja wartości. Jest to spowodowane tym, że funkcja rand() generuje liczby pseudolosowe. Konieczne w tym wypadku jest losowanie, które jest realizowane przez funkcje srand(). Funkcja srand() pobiera jako argument liczbę unsigned int i wywołuje funkcję rand() do rozpoczęcia wytwarzania różnych sekwencji liczb losowych. Jeżeli za każdym razem chcemy losować bez konieczności wprowadzania liczby bazowej do generowania liczb losowych, możemy użyć wyrażenia:

```
srand(time(0));
```

Wyrażenie to odczytuje zegar, aby otrzymać wartość bazową dla generowania liczb losowych. Funkcja time z argumentem 0 zwraca czas jaki upłynął w sekundach od 1 stycznia 1970 roku. Oto program zliczający rzut kostką 5000 razy. Wykorzystujący w tym celu tablicę:

```
#include <iostream>
#include <cstdlib>
```

```
using namespace std;

int main()
{
        const int wTab=7;
        int czestosc[wTab]={0};
        srand(time(0));
        //ignoruj element zero w tablicy
        for(int rzut=1; rzut<=5000; rzut++)
                ++czestosc[1+rand()%6];

        cout<<"Scianka : "<<" czestosc wystepowania \n";
        for(int scianka=1;scianka<wTab;scianka++)
                cout<<scianka<<" "<<czestosc[scianka]<<endl;

        cin.get();
        return 0;
}
```

Przekazywanie tablic do funkcji

Chcąc przekazać tablicę do funkcji, piszemy nazwę tablicy bez nawiasów, np.:

```
#include <iostream>

void wyswietlTablice (int [], int);

using namespace std;

int main()
{

        const int wTab=10;
        int tab[ wTab ]={1,4,7,5,100,150,9,34};

        cout<<"Wywolanie funkcji wyswietlTablice"<<endl;

        wyswietlTablice(tab, wTab);
        cin.get();
```

```
        return 0;
}

void wyswietlTablice (int tablica [], int wielkosc)
{
 for(int i=0; i<wielkosc; i++)
   cout<<"Element tablicy : "<<(i+1)
     <<" Wartosc elementu :"<<tablica[i]<<endl;
}
```

Przekazując tablicę do funkcji, przekazuj jednocześnie jej rozmiar. Dzięki temu funkcja może przetworzyć odpowiednią ilość elementów. W przeciwnym wypadku musielibyśmy informacje o wielkości tablicy umieścić wewnątrz funkcji lub określić jako zmienną globalną, co jest niezgodne z zasadą najmniejszego przywileju.
C++ automatycznie przekazuje tablice do funkcji przez referencję (a jak się przekonamy później wskaźnik), w związku z tym funkcja może modyfikować elementy tablicy bezpośrednio. Jest to spowodowane względami wydajnościowymi. Jeśli tablicę przekazywalibyśmy przez wartość, kopiowanie wszystkich elementów tablicy zabierałoby dużo czasu i potrzebowałoby dużo pamięci. Poszczególne elementy tablicy są przekazywane są przez wartość, w związku z tym funkcja nie modyfikuje elementu bezpośrednio, a operuje na kopii wartości bezpośrednio.

Poniższy program demonstruje przekazywanie tablic i poszczególnych elementów do tablic:

```
 #include <iostream>

void wyswietlTablice (int [], int);
void modyfikujTablice (int [], int);
void modyfikujElement (int );

using namespace std;

int main()
{

  const int wTab=10;
  int tab[ wTab ]={1,4,7,5,100,150,9,34};

  cout<<"Wywolanie funkcji wyswietlTablice przed modyfikacja\n";
  wyswietlTablice(tab, wTab);
```

```
 modyfikujTablice(tab,wTab);

 cout<<”Tablica po modyfikacji :\n”;
 wyswietlTablice(tab, wTab);

 cout<<”Element 3 tablicy przed wywolaniem funcji modyfikujElement :”<<tab[3]<<endl;
 modyfikujElement(tab[3]);

 cout<<”Element 3 tablicy po wywolaniu funcji modyfikujElement : “<<tab[3]<<endl;
 cin.get();
  return 0;
}

void wyswietlTablice (int tablica [], int wielkosc)
{
        for(int i=0; i<wielkosc; i++)
                cout<<”Element tablicy : “<<(i+1)<<” Wartosc elementu :
“<<tablica[i]<<endl;
}
void modyfikujTablice (int tablica [], int wielkosc )
{
        for(int i=0; i<wielkosc; i++)
                tablica[i]=tablica[i]+10;
}

void modyfikujElement (int a)
{
        a*=2;
        cout<<”Wewnatrz funkcji modyfikujElement :”<<a<<endl;
}
```

Sortowanie tablic

Sortowanie danych jest jednym z najważniejszych zadań obliczeniowych. Powstało wiele wydajnych algorytmów, których zadaniem jest sortowanie tablic. Tutaj przedstawię dwa najprostsze algorytmy sortowania: bąbelkowe i przez selskcję.

Sortowanie bąbelkowe

Sortowanie bąbelkowe wymaga kilku przejść przez całą tablicę. W każdym przejściu tablicy kolejne pary są porównywane. Jeżeli para znajduje się w porządku rosnącym lub wartości są identyczne, ich wartości są pozostawione. Jeśli para znajduje się w porządku malejącym, wartości zamieniane są miejscami.

```
#include <iostream>

void wyswietlTablice (int [], int);
void sortBab(int [], int);

using namespace std;
int main ()
{
        const int wTab=10;
        int tab[ wTab ]={1,4,7,5,100,150,9,34,-1,120};

        cout<<”Tablica przed posortowaniem :\n”;
        wyswietlTablice(tab,wTab);
        sortBab(tab,wTab);
        cout<<”Tablica po posortowaniu :\n”;
        wyswietlTablice(tab,wTab);
        cin.get();
        return 0;
}

void wyswietlTablice (int tablica [], int wielkosc)
{
        for(int i=0; i<wielkosc; i++)
                cout<<”Element tablicy : “<<(i+1)<<” Wartosc elementu :
“<<tablica[i]<<endl;
}

void sortBab (int tablica [], int wielkosc )
{
        int temp;
```

```
        for (int i=0; i<wielkosc-1; i++)
        {
                for (int j=0; j<wielkosc-i; j++)
                {
                        if(tablica[j]>tablica[j+1])
                        {
                                temp=tablica[j];
                                tablica[j]=tablica[j+1];
                                tablica[j+1]=temp;
                        }
                } // for(j=0..
        } // for(i=0..
}
```

Sortowanie przez selekcję

Algorytm ten jest następujący:
1. należy znaleźć najmniejszy element i zmienić go miejscem z pierwszym,
2. znaleźć drugi element w kolejności i zamienić go miejscem w drugim,
3. kontynuować krok drugi, aż do posortowania.

```
#include <iostream>

void wyswietlTablice (int [], int);
void sortSel(int [], int);

using namespace std;

int main ()
{

        const int wTab=10;
        int tab[ wTab ]={1,4,7,5,100,150,9,34,-1,120};

        cout<<”Tablica przed posortowaniem :\n”;
        wyswietlTablice(tab,wTab);
        sortSel(tab,wTab);
        cout<<”Tablica po posortowaniu :\n”;
        wyswietlTablice(tab,wTab);
        cout<<”Nacisnij ENTER”<<endl;
```

```
        cin.get();
        return 0;
}

void wyswietlTablice (int tablica [], int wielkosc)
{
        for(int i=0; i<wielkosc; i++)
        cout<<"Element tablicy : "<<(i+1)<<" Wartosc elementu : "<<tablica[i]<<endl;
}
void sortSel (int tablica [], int wielkosc )
{
 void zmien( int [], int, int);
 int min;

 for (int i=0; i<wielkosc; i++)
        {
        min=i;
         for(int j=i+1; j<wielkosc; j++)
                {
                if(tablica[j]<tablica[min])
                        {
                                min=j;
                        }
                }
         zmien(tablica,i,min);
 }
}

void zmien (int tablica [], int indeks, int indeks1)
{
        int temp;
        temp=tablica[indeks];
        tablica[indeks]=tablica[indeks1];
        tablica[indeks1]=temp;
}
```

Tablice wielowymiarowe

W C++ tablice mogą być wielowymiarowe. Typowym ich zastosowaniem jest przedstawienie informacji pogrupowanych w wierszach i kolumnach. W tablicach wielowymiarowych, chcąc wskazać element tablicy, musimy określić dwa indeksy: pierwszy oznacza wiersz, a drugi kolumnę.
Tablice wymagające dwóch indeksów do zidentyfikowania określonego elementu są nazywane tablicami dwuwymiarowymi. Mogą mieć one więcej niż dwa indeksy. Kompilatory C++ dopuszczają 12 indeksów tablicy. Wielowymiarowa tablica może być zainicjowana w podobny sposób do tablicy jednowymiarowej. Chcąc utworzyć dwuwymiarową tablicę typu int, mającą dwa wiersze i dwie kolumny, napiszemy (jednocześnie incjując):

```
int tabInt[2][2]={{1,1},{2,2}};
```

Wartości w nawiasach klamrowych zgrupowane są według wierszy. Chcąc przekazać tablicę wielowymiarową do funkcji, musimy określić wielkość drugiego i ewentualnie następnych indeksów. Kompilator używa tych wielkości do określenia położenia w pamięci elementów wielowymiarowej tablicy. Przykładowo: chcąc przekazać do funkcji wyswTab() tablicę tabInt, musielibyśmy tę funkcję zadeklarować:

```
void wyswTab(int [][2],int );
```

A pierwsza linia definicji funkcji wyglądałaby tak:

```
void wyswTab(int tablica[][2], int wielkosc)
```

Ćwiczenia

1. Sortowanie bąbelkowe jest mało wydajne - dopracuj ten algorytm. Aby poprawić wydajność (przecież dane w tablicy mogą być już w całości lub w części posortowane), zmodyfikuj sortowanie tak, aby na końcu każdego przebiegu sprawdzano czy zostały dokonane jakieś zmiany, jeżeli nie - dane są już uporządkowane i należy zakończyć sortowanie.
2. Pewne przedsiębiorstwo płaci swoim sprzedawcom wynagrodzenie zależnie od wartości sprzedaży. Sprzedawcy otrzymują 690 zł + 10% swojej sprzedaży brutto w tym miesiącu. Napisz program obliczający pensję sprzedawcy w zależnośći od sprzedaży.
3. Napisz algorytm sortowania przez selekcję w sposób rekurencyjny.
4. Napisz program symulujący rzuty dwiema kostkami 100 razy. Powinien on wykorzystywać funkcję rand() w połączeniu z funkcją srand(). Suma tych 36 (od 2 do 12) kombinacji winna być

wyświetlona w wierszach i kolumnach jak poniżej.

Źródła:

- Andrzej Stasiewicz, C++ Ćwiczenia praktyczne, Helion, Gliwice, 2010.
- http://cpp0x.pl/
- http://www.ithelpdesk.pl/
- http://pl.wikibooks.org/wiki/C++

Delphi

Delphi – język programowania, którego można używać w środowiskach firmy Borland, Embarcadero, Microsoft (Delphi Prism) oraz w środowisku Lazarus. Narzędzia te są zintegrowanymi środowiskami programistycznymi typu RAD.

Dawniej język Delphi był nazywany Object Pascalem, lecz w świadomości społecznej programistów na całym świecie Delphi zaczęło być postrzegane z czasem jako osobny język programowania. Przy okazji premiery Delphi 6 w roku 2002 r., w oficjalnej dokumentacji programu, została użyta po raz pierwszy nazwa „Delphi language". Standard języka Delphi obejmuje wiele bogatych funkcjonalnie klas, których nie ma w standardzie Object Pascala, a ponadto umożliwia programowanie wizualne, czyli Object-Oriented Design. Object Pascal jest rozszerzeniem języka Pascal.

Programy tworzone w Delphi dla Win32 muszą zostać skompilowane do postaci kodu binarnego przed pierwszym wykonaniem. Niektóre komponenty wizualne działają już podczas tworzenia projektu, umożliwiając oglądanie efektów pracy. Delphi dla Win32 zapisuje informacje o właściwościach obiektów, udostępniając je programiście. Informacje te umożliwiają zmianę ich wartości przez programistę bez pisania kodu programu oraz są używane podczas pracy programu – technika ta zwana jest RTTI. Tworzone programy pracują na zasadzie obsługi zdarzeń, każde polecenie (np. kliknięcie myszką) generuje zdarzenie, które poprzez wewnętrzne mechanizmy programu są przesyłane do odpowiedniego komponentu, a rolą programisty jest tylko dołączenie odpowiedniego kodu umożliwiającego obsługę tego zdarzenia. Natywne programy tworzone w Delphi pod Win32 umożliwiają w prosty sposób stworzenie wydajnej aplikacji, sam język jest przyjazny użytkownikowi i podobny do języka C# dla platformy .NET

Odpowiednik Delphi produkcji firmy Borland dla Linuksa nosił nazwę Kylix. Jego produkcja została zaniechana, gdyż charakteryzował się niską jakością. Obecnie rozwijane jest na licencji GNU GPL zintegrowane środowisko programistyczne (IDE) o nazwie Lazarus, wzorowane na Delphi. Lazarus jest środowiskiem wieloplatformowym, dostępnym dla różnych systemów operacyjnych.

W przeciwieństwie do języka Pascal, Delphi nie było tworzone dla celów edukacyjnych, lecz biznesowych, jako język, który miał połączyć prostotę i przejrzystość języka Pascal z łatwym i wygodnym tworzeniem aplikacji. Delphi umożliwia również niskopoziomowe programowanie poprzez możliwość wstawiania części kodu napisanego w języku assembler. Jest językiem obiektowym.

Cechy i funkcjonalność:

- wspomaganie dla obsługi relacyjnych systemów bazodanowych,
- obsługa standardowych mechanizmów windowsowych,
- szeroki zestaw gotowych do użycia komponentów,
- rozszerzalność środowiska (zarówno palety komponentów, jak i samego IDE),
- budowa wizualnej części aplikacji za pomocą techniki drag and drop,

• zawiera wiele elementów mających na celu uproszczenie tworzenia aplikacji związanych z Internetem,
• szybki, efektywny kompilator,
• zawiera wiele dodatkowych narzędzi wspomagających programistów.

Środowisko Delphi wraz z dołączonymi narzędziami może być uznane za język czwartej generacji.

Środowisko użytkowników

Delphi cieszy się w Polsce stosunkowo dużą popularnością, w głównej mierze ze względu na prostotę i powszechność różnego rodzaju poradników dla początkujących.

Ważne uwagi

1. Podczas pisania programu źródłowego w Delphi wiele linii kodu generuje samo środowisko Delphi.
2. W systemie Turbo Pascal plik źródłowy ma rozszerzenie *.pas.
3. Program napisany w języku Delphi jest nazywany projektem. Środowisko Delphi dla każdego projektu tworzy wiele plików. Nazwa pliku składa się z dwóch elementów: nazwy nadanej projektowi i jego modułom oraz predefiniowanego rozszerzenia stosowanego przez Delphi.
4. Pliki źródłowe napisane w języku Delphi mają różne rozszerzenia (nie tylko *.pas).
5. Podczas tworzenia nowego projektu w trybie okienkowym Delphi zakłada 6 - 9 plików, m. in.:
- plik źródłowy projektu, który zawiera kod wykonywany podczas uruchamiania aplikacji,
- moduł formularza głównego, zawierający deklarację i definicję klasy formularza głównego,
- plik zasobów formularza głównego,
- plik zasobów projektu.

Pojęcie projektu

Projekt jest to zestaw plików, których przetworzenie przez Delphi daje w efekcie gotowy program (plik wykonywalny). Skompilowanie i połączenie plików projektu może dać w efekcie jeszcze inne obiekty, np. bibliotekę DLL. Najważniejszymi elementami projektu Delphi (okienkowego) są pliki źródłowe modułów, będących głównymi składnikami funkcjonalnymi programu (np. Unit1.pas oraz Unit1.dfm).

Start systemu Delphi7

1. Wybrać: Start | Programy | Borland Delphi 7 | Delphi 7
2. Wejść do katalogu C:\Program Files \ Borland \ Delphi7 \ Bin i wybrać plik delphi32.exe
3. Na pulpicie utworzyć ikonę skrótu do programu.

Okno programu Delphi

Po uruchomieniu programu na ekranie pojawia się zestaw okien i innych elementów tworzących tzw. zintegrowane środowisko robocze – IDE (Integrated Development Environment). Układ okien, charakterystyczny dla narzędzi RAD firmy Borland, odzwierciedla samą metodę tworzenia aplikacji – zamiast okna edytora, w którym wpisuje się tekst programu, centralne miejsce na ekranie zajmuje formularz, na którym ustawia się komponenty.

Elementy IDE:

• okno zatytułowane Form1 – główne pole robocze – formularz główny (main form) programu (aplikacji), reprezentuje okno aplikacji i stanowi „podłoże", na którym umieszcza się komponenty, wybierane z palety komponentów i umieszczane na formularzu poprzez klikanie myszą; proste programy zawierają zwykle pojedynczy formularz (formularz główny),
• paleta komponentów, zawierająca kilka kart obiektowych, np. Standard, Additional, Win32, System, Dialogs, komponenty przeznaczone do obsługi baz danych, komponenty internetowe, itd.,
• okno hierarchii komponentów Object TreeView – prezentuje ono w formie graficznej wszystkie komponenty zawarte na formularzu oraz zależności między nimi,
• okno inspektora obiektów Object Inspector – zawierające dwie karty: kartę Properties (lista właściwości – ich nazwy i wartości) i kartę Events (lista zdarzeń i procedur ich obsługi),
• okno edytora kodu źródłowego o nazwie Unit1.pas zawierające kod źródłowy modułu o nazwie Unit1, który to moduł jest częścią składową całego projektu (treść programu),
• pasek tytułu – „Delphi 7 – Project1",
• pasek menu (menu główne),
• paleta narzędzi – przyciski realizujące najczęściej wykonywane operacje:

górny rząd

New items - utwórz nowy projekt
Open - otwórz plik
Save - zapisz plik
Save all - zapisz wszystkie pliki
Open Project - otwórz projekt
Add file to Project - dodaj plik do projektu
Remove file from Project - usuń plik z projektu
Help contents - pomoc

dolny rząd:

View Unit - wyświetl listę modułów
View Form - wyświetl listę formularzy
Toggle Form / Unit - przełącz tekst/formularz

New Form - utwórz nowy formularz
Run - uruchom program
Pause - wstrzymaj program
Trace into - wykonaj następną instrukcję
Step over - wykonaj następną procedurę

Pojęcie komponentu

Komponent jest to element programu, realizujący konkretne zadanie (umożliwiający wprowadzanie lub wyprowadzanie danych, obsługujący połączenie z serwerem WWW, odmierzający czas, wyświetlający zawartość folderu, itd.).

Zapisywanie plików w trybie okienkowym

1. IDE proponuje różne katalogi w zależności od sposobu uruchomienia (Start | Programy | … czy ikona skrótu):
C:\Program Files \ Borland \ Delphi7 \ Projects lub
C:\Program Files \ Borland \ Delphi7 \ Bin
2. Delphi nie zapamiętuje nazwy katalogu użytego do zapisania ostatniego projektu; każdorazowo trzeba wskazywać katalog docelowy.
3. Zapisywanie projektów Delphi odbywa się dwuetapowo:
Save Unit1 As
Save Project1 As
w sumie min. 6 plików:
Unit1.pas – tekst programu w Pascalu,
Unit1.dfm – opisuje układ i właściwości formularza i wszystkich zawartych w nim obiektów
pliki zawierające informacje o ustawieniach projektu i tzw. zasobach

Project1.dof
Project1.dpr
Project1.cfg
Project1.res

Najważniejszymi elementami projektu są pliki źródłowe modułów, będących głównymi składnikami funkcjonalnymi programu.

Pasek menu (menu główne)

File (plik) – manipulowanie plikami i projektami
New
Application – utworzenie nowego programu
……..

Other – przejście do aplikacji konsolowej
Open – otwórz plik
Open Project – otwórz projekt
Reopen - lista ostatnio otwieranych plików
Save - zapisz plik
Save As - zapisz jako
Save Project As - zapisz projekt jako
Save All – zapisz wszystkie otwarte pliki
Close
Close All
Use Unit
Print
Exit

Edit (edycja) - typowe funkcje służące do edycji tekstu programu i zawartości formularza
Undelete – cofnięcie operacji
Redo – powtórzenie operacji
Cut - wycinanie
Copy - kopiowanie
Paste - wklejanie
Delete -- usuwanie
Select All – wybieranie elementów
polecenia zawarte w dolnej części menu służą do manipulowania komponentami umieszczonymi na formularzu
Align to Grid
……
Lock Controls

Search (szukaj)
Find – wyszukiwanie tekstu
Find in Files – przeszukiwanie kilku wybranych plików
Replace – zamiana tekstu
Incremental Search – wyszukiwanie tekstu pasującego do kolejno wpisywanych znaków

View (widok) – zarządzanie oknami wchodzącymi w skład IDE
Project Manager - wyświetla zawartość projektu
Object Inspector
Object TreeView

Alignment Palete – umożliwia manipulowanie komponentami na formularzu
Component List
Window List – wyświetla listę otwartych okien
Additional Message Info
Debug Windows – udostępnia obszerne menu okien podsystemu uruchomieniowego (debuggera)
Desktops
Toggle Form / Unit – przełącza pomiędzy oknami formularza i edytora kodu
Units – wyświetla listę modułów zawartych w projekcie
Forms – wyświetla listę formularzy zawartych w projekcie
New Edit Window
Toolbars – konfiguruje zwartość okna głównego

Project (projekt) – polecenia służące do zarządzania projektami
Add to Project – dodaj wybrany moduł do projektu
Remove from Project– usuń wybrany moduł z projektu
Import Type Library
Add to Repository
View Source
polecenia umożliwiające tworzenie tzw. grup składających się z kilku powiązanych ze sobą projektów
Add New Project Add Existing Project
polecenia pozwalające przetworzyć tekst źródłowy programu na postać wykonywalną
(polecenia kompilacji)
Compile Project1
Build Project1
Syntax check Project1 – sprawdzanie poprawności tekstu źródłowego polecenia kompilacji wszystkich projektów wchodzących w skład grupy
Compile All Project
Build All Project
Options – okno umożliwiające zmianę ustawień projektu

Run (uruchomienie)
Run – uruchom program
Parameters
Step Over - wykonaj następną procedurę
Trace Into - wykonaj następną instrukcję
Trace to Next Source Line

Run to Cursor
Run Until Return
Show Execution Point
Program Pause – wstrzymuje pracę programu
Program Reset – przerywa wykonanie programu
Evaluate / Modify
Add Watch
Add Breakpoint

Component (komponent) - zarządzanie komponentami
New Component
Install Component
Import ActiveX Control
Create Component Template
Install Packages
Configure Palette

Tools (narzędzia) – udostępnia funkcje konfiguracyjne oraz programy narzędziowe
Environment Options – zmiana ustawień całego IDE
Editor Options – zmiana ustawień samego edytora kodu
Debugger Options
Repository
Configure Tools
Image Editor

Window (okno) – spis aktualnie otwartych okien

Help (pomoc)

System Delphi 7 - aplikacje konsolowe - możliwość tworzenia tradycyjnych programów pracujących w trybie tekstowym:

Opis postępowania
1. Uruchomić program Delphi
np. Start | Programy | Borland Delphi 7 | Delphi 7.
2. Pojawia się standardowe okno programu Delphi.
Przejść do trybu tekstowego (aplikacji konsolowej)
3. Wybrać polecenie File | New | Other ….

Na ekranie pojawi się okno New Items, reprezentujące tzw. składnicę obiektów (Object Repository).
Składnica obiektów jest w Delphi magazynem zawierającym kreatory i szablony programów oraz ich składników (formularzy, bibliotek, okien, modułów danych oraz innych elementów związanych z bazami danych i Internetem). Służy ona jako punkt wyjścia do tworzenia projektów i zawiera kilka kart (New, Project1, Forms, Dialogs, Projects).
4. Przejść na kartę New, skupiającą najpopularniejsze elementy aplikacji i narzędzia. Interesuje nas tzw. aplikacja tekstowa lub konsolowa (console application).
5. Aby utworzyć odpowiedni projekt należy na karcie New kliknąć dwukrotnie ikonę Console Application.
6. Utworzony w ten sposób projekt nie zawiera formularza, posiada tylko okno edytora, a okna Object TreeView i Object Inspector są puste. Na pasku tytułu pojawia się nazwa Project2.dpr. W oknie edytora pojawia się następująca treść programu:

```
program Project2;

{$APPTYPE CONSOLE}
        (dyrektywa określająca rodzaj aplikacji - tekstowa lub okienkowa)

uses SysUtils;                  (lista modułów wykorzystywanych przez program)


begin
  { TODO –oUser –cConsole Main : Insert code here }
end.
```

Powyższe 10 linijek to kompletny program w Pascalu, który co prawda nie robi nic, ale daje się uruchomić (np. klawiszem F9) – wyświetla przez chwilę puste okno.
Komentarz zaczynający się od słowa TODO (do zrobienia) można usunąć.
Pomiędzy słowa kluczowe begin i end wpisać instrukcje w języku Pascal (program źródłowy w języku Pascal).
7. Zapisać projekt w swoim katalogu:
polecenie File | Save Project As ….
pod wybraną, opisową nazwą.
8. Skompilować i uruchomić program:
- polecenie Run | Run,
- klawisz F9,
- przycisk uruchomienia - zielona strzałka na palecie narzędzi (dolny pasek narzędziowy).

9. Kompilację i uruchomienie programu można przeprowadzić dwuetapowo:
- polecenie Project | Compile nazwa_projektu, <Ctrl + F9>
- polecenie Run | Run <F9>
10. Zamknąć aplikację
- polecenie File | Exit,
- przycisk zamknięcia głównego okna Delphi „X”,
- klawisze Alt+F4
11. Otwarcie istniejącego pliku *.dpr przygotowanego w trybie tekstowym powoduje automatyczne przejście programu Delphi do trybu aplikacji konsolowej.

W aplikacji konsolowej (tekstowej) najważniejszym plikiem jest plik *.dpr, zawierający kod źródłowy programu Pascal.

W systemie Turbo Pascal standardowo tworzony był jeden plik (*.pas), zawierający kod źródłowy.

Podczas tworzenia nowego projektu w trybie tekstowym Delphi zakłada 4 pliki:
1. Plik źródłowy projektu (*.dpr), który zawiera kod wykonywany podczas uruchamiania aplikacji.
2. Wykonywalny program wynikowy (*.exe).
3. Plik konfiguracyjny projektu (*.cfg), w którym zawarte są główne ustawienia kompilatora i konsolidatora.
4. Plik opcji projektu (*.dof), w którym zawarte są bieżące ustawienia wszystkich opcji projektu.

Zapisywanie plików w trybie tekstowym

1. IDE proponuje różne katalogi w zależności od sposobu uruchomienia (Start | Programy | … czy ikona skrótu):

C:\Program Files \ Borland \ Delphi7 \ Projects lub
C:\Program Files \ Borland \ Delphi7 \ Bin

2. Delphi nie zapamiętuje nazwy katalogu użytego do zapisania ostatniego projektu; każdorazowo trzeba wskazywać katalog docelowy.
3. Zapisywanie projektu - polecenie:

File | Save Project As

Przykładowe materiały do wykorzystania na lekcjach z przedmiotu „Programowanie strukturalne i obiektowe" w technikum informatycznym. Materiały zostały opracowane na przykładzie języka programowania Delphi.

Przedmiot: Programowanie strukturalne i obiektowe
Temat: Przyciski opcji
Czas: 2 jednostki lekcyjne
Cele lekcji:
- zapoznanie z nowymi komponentami,
- zapamiętanie informacji o ich cechach, przeznaczeniu i możliwościach wykorzystania,
- ćwiczenie umiejętności samodzielnego tworzenia programów z wykorzystaniem poznanych komponentów,
- wyrabianie i kształtowanie umiejętności samodzielnego uczenia się.

Przycisk opcji (ang. radio button) jest to widżet, element graficznego interfejsu użytkownika programów komputerowych.
Przycisk opcji posiada dwa stany: włączony i wyłączony, a „naciśnięcie" go lewym klawiszem myszy zawsze spowoduje jego włączenie. W momencie aktywacji deaktywuje on pozostałe przyciski radiowe z grupy - widżet ten to grupa przycisków wzajemnie wykluczających się.
Zdarzenia generowane są spod każdego przycisku; zawsze generowane jest zdarzenie aktywacji przycisku, bywa że jest też generowane zdarzenie deaktywacji wcześniej aktywnego przycisku. Jeśli biblioteka jest wyposażona w mechanizm sygnałów i slotów, to zwykle podłącza się wszystkie sygnały aktywacji każdego przycisku z grupy do jednego slotu, gdyż wszystkie przyciski zwykle są stowarzyszone z jednym stanem w używającym je oknie.
Widżet [wym. łidżet] to podstawowy element graficznego interfejsu użytkownika (GUI), np. okno, pole edycji, suwak lub przycisk. Termin ten jest szczególnie popularny wśród użytkowników systemów operacyjnych z rodziny UNIX, natomiast użytkownicy systemów MS Windows używają w tym kontekście terminu kontrolka lub element kontrolny. W pewnych kontekstach synonimem widżetu jest okno.
Graficzny interfejs użytkownika (ang. Graphical User Interface, GUI) często nazywany też środowiskiem graficznym – ogólne określenie sposobu prezentacji informacji przez komputer oraz interakcji z użytkownikiem, polegającego na rysowaniu i obsługiwaniu widgetów.

Zadanie 1
Przeczytaj uważnie powyższe definicje. Zastanów się, czy spotkałeś się już z przyciskami opcji w programach komputerowych. Zanotuj w zeszycie temat lekcji oraz sporządź notatkę z definicją przycisku opcji. (10 minut)

RadioButton – przycisk opcji w Delphi
Przyciski budowane za pomocą komponentu RadioButton służą do wyboru jednego spośród kilku elementów. Możemy je umieszczać na formie, aczkolwiek najlepiej to zrobić w oknie komponentu GroupBox.

Uwaga! Przy korzystaniu z komponentu RadioButton zwróć uwagę na nową właściwość Checked. Przybiera ona wartość True (gdy komponent jest zaznaczony) lub False.

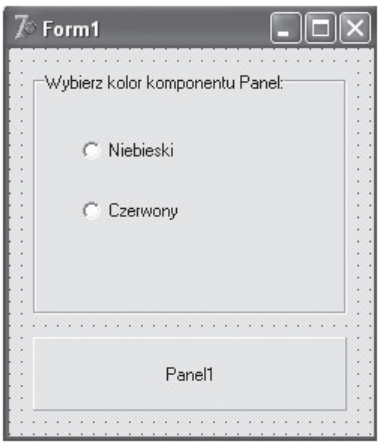
RadioGroup – grupa przycisków opcji
Komponent RadioGroup pozwala na generowanie przycisków opcji. Nie musimy ich umieszczać na formie w sposób ręczny. Do tego zadania służy narzędzie String List Editor, które uaktywnia się po kliknięciu pola właściwości Items.
Przy obsłudze korzystamy ze zdarzenia OnClick dla RadioGroup i wykorzystujemy właściwość ItemIndex, podającą numer wybranego przycisku. Kolejne przyciski w grupie są numerowane od 0, a więc pierwszy element ma numer 0, drugi ma numer 1 itd. Zapisując operacje związane z wyborem poszczególnych opcji, możemy wykorzystać instrukcję case. W danej chwili może być wybrana co najwyżej jedna opcja. Gdy żadna opcja nie jest wybrana, to ItemIndex = -1. Komponent nie reaguje na kliknięcie, jeśli wybieramy opcję już zaznaczoną. Aby móc ponownie wybrać tę samą opcję, możemy programowo zmienić wartość ItemIndex: RadioGroup1. ItemIndex:= -1; // usuwa zaznaczenie opcji.

Zadanie 2
Przeczytaj powyższy tekst. Odszukaj opisane komponenty i wstaw je na formularzu. Przejrzyj ich właściwości, starając się zwrócić szczególną uwagę na wyżej opisane (plus columns) oraz te, które dotyczyły komponentów, które już znasz. Uruchom utworzony program i przetestuj zachowanie się przycisków opcji. Uzupełnij notatkę o definicje wymienionych komponentów oraz listę istotnych ich cech. (15 minut)

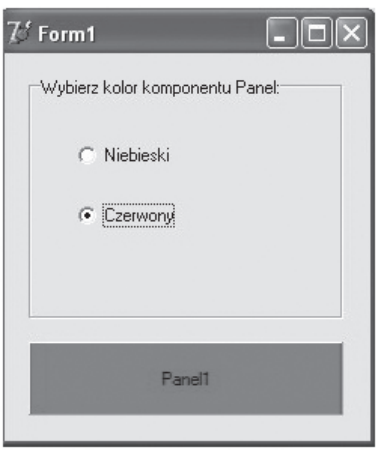
Zadanie 3
Utwórz program, który zawiera dwa przyciski opcji umieszczone w komponencie GroupBox oraz komponent Panel na zakładce Standard. Przyciski opisz dowolnymi nazwami kolorów (np. niebieski i czarny) - po wyborze jednego z przycisków kolor komponentu Panel będzie się zmieniał. Zadanie możesz wykonać samodzielnie lub skorzystać z podanego rozwiązania. Jeżeli uznasz to za słuszne uzupełnij notatkę w zeszycie. Staraj się dobrze zrozumieć zasadę działania programu oraz wykorzystanych komponentów i instrukcji. (15 minut)

1. Umieść na formie komponent GroupBox oraz dwa przyciski opcji RadioButton.
2. Zmień napis na przyciskach opcji na nazwy wybranych przez siebie kolorów.

3. Umieść na formie komponent Panel. Wszystkie komponenty umieść tak, jak na rysunku:



4. Podepnij pod przyciski opcji instrukcje według wzoru:
Panel1.Color:=ClBlue; {Po znaku równości wpisz nazwę koloru}
5. Sprawdź działanie aplikacji (Rysunek 2)



Zadanie 4

Napisz aplikację, która będzie zawierać dwa przyciski opcji, komponent przycisku oraz etykietę. Po naciśnięciu na etykiecie będzie wyświetlać się nazwa zaznaczonego przycisku opcji. Zadanie możesz wykonać samodzielnie lub skorzystać z podanego rozwiązania. Jeżeli uznasz to za słuszne, uzupełnij notatkę w zeszycie. (10 minut)

1. Wstaw na formę dwa przyciski opcji, przycisk oraz komponent Label (etykietę).
2. Ustaw właściwość Checked komponentu RadioButton1 na True.
3. Podepnij pod przycisk instrukcję warunkową:


```
if RadioButton1.Checked then
label1.Caption:=RadioButton1.name
else {przed tą instrukcją nie wpisujemy znaku średnika}
label1.Caption:=RadioButton2.name;
```

Powyższą instrukcję określa się mianem instrukcji warunkowej (if ... then ... else ...).

4. Sprawdź działanie programu.

Zadanie 5

Napisz aplikację, która będzie zawierać pole RichEdit (karta Win32) oraz grupę przycisków opcji opisanych nazwami 5 kolorów. Przyciskami opcji wybieramy kolor czcionki w oknie RichEdit. Nazwy wybranych kolorów wyświetlane są w etykiecie.
Podpowiedź: Wykorzystaj cechę Items do wyświetlania nazw kolorów (Radiogroup1. Items[RadioGroup1.Itemindex]) (15 minut)

Zadanie 6

W sklepie z wyposażeniem wnętrz sprzedawany jest parkiet z różnego rodzaju drewna: sosna, buk, dąb, w dwóch gatunkach: pierwszym i drugim. Cena parkietu w drugim gatunku stanowi 70% ceny gatunku pierwszego. Ustal ceny dla gatunku pierwszego. Zbuduj grupy przycisków opcji do wyboru rodzaju i gatunku parkietu. Przycisk Wyświetl cenę uruchamia procedurę wyświetlającą w oknie komunikatu cenę jednego metra wybranego parkietu. (15 minut)

Zadanie 7

Zapisz wszystkie wykonane przez siebie zadania w katalogu nazwanym swoim nazwiskiem i imieniem oraz skopiuj go na serwer (sbs2005) do folderu Programowanie 3Ti. (5 minut)

Po lekcji powinieneś umieć:

- scharakteryzować poznane komponenty,
- wykorzystać poznane komponenty w programach komputerowych,
- samodzielnie tworzyć notatki oraz budować programy komputerowe w oparciu o nowe wiadomości.

Przedmiot: Programowanie strukturalne i obiektowe
Temat: Przyciski wyboru
Czas: 1 jednostka lekcyjna
Cele lekcji:

- zapoznanie z nowymi komponentami,
- zapamiętanie informacji o ich cechach, przeznaczeniu i możliwościach wykorzystania,
- ćwiczenie umiejętności samodzielnego tworzenia programów z wykorzystaniem poznanych komponentów,
- wyrabianie i kształtowanie umiejętności samodzielnego uczenia się.

Przycisk wyboru (ang. check box) jest w działaniu bardzo podobny do przycisku przełączania, z tym tylko, że znacznie różni się od niego wyglądem.
Przycisk zwykle jest skojarzony z odpowiednią etykietą (niektóre biblioteki, np. MS Windows API, i tak implementują napis jako stan każdego widżetu), natomiast jeśli chodzi o wygląd to jest to zwykle mały prostokącik, wielkości mniej więcej jednego-dwóch znaków, który w zależności od stanu albo jest „czysty", albo jest na nim narysowany znaczek „tick" (v z wydłużonym prawym ramieniem), ewentualnie niektóre style stosują krzyżyk (zdarza się też mały prostokącik, który wygląda na wypukły lub wklęsły). Po prawej lub lewej stronie tego prostokąciku znajduje się etykieta przycisku (innych sposobów umieszczania etykiety się raczej nie spotyka).
Przycisk wyboru jest zwykle trójstanowy (oczywiście tylko od konfiguracji zależy, czy będzie on się zachowywał jak dwustanowy lub trójstanowy). W „trzecim stanie" przycisk zaznaczania wygląda zazwyczaj tak jak w stanie „zaznaczony", tyle tylko że rysunek „tick" -
a jest wyszarzony.
Przycisk ten można tylko kliknąć lewym klawiszem myszy, po czym zmienia on stan na „następny" (w przypadku przycisków dwustanowych jest to odwrócenie stanu, w przypadku trójstanowych jest to cykliczne przechodzenie w inne stany).
Zdarzenia są niemal identyczne, jak w przypadku przycisku - generowanie zdarzenia następuje w momencie zwolnienia przycisku myszy, pod warunkiem oczywiście, że zwolnienie nastąpi w momencie, gdy kursor myszy znajduje się nad prostokątem lub etykietą przycisku.

Zadanie 1

Przeczytaj uważnie powyższą definicję. Zastanów się, czy spotkałeś się już z przyciskami wyboru w programach komputerowych. Zanotuj w zeszycie temat lekcji oraz sporządź notatkę z definicją przycisku wyboru. (10 minut)

CheckBox – przycisk wyboru w Delphi
Przyciski wyboru tworzy się za pomocą komponentu CheckBox. Służą one do zaznaczania elementów grupy. W odróżnieniu od przycisków RadioButton, jednocześnie może być tu wybrany więcej niż jeden składnik.

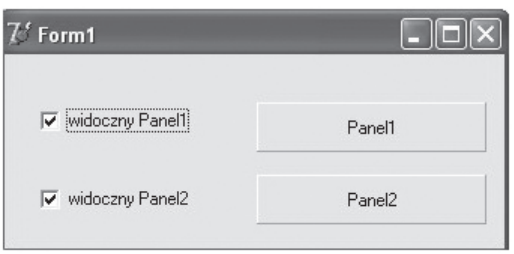
Zadanie 2

Przeczytaj powyższy tekst. Odszukaj opisany komponent i wstaw go na formularzu. Przejrzyj jego właściwości. Uruchom utworzony program i przetestuj zachowanie się przycisków wyboru. Uzupełnij notatkę o definicję tego komponentu oraz listę istotnych jego cech. (5 minut)

Zadanie 3

Utwórz program, który za pomocą komponentów CheckBox będzie uwidaczniał komponenty typu Panel. Zadanie możesz wykonać samodzielnie lub skorzystać z podanego rozwiązania. Jeżeli uznasz to za słuszne uzupełniaj notatkę w zeszycie. (10 minut)

1. Wstaw na formę dwa komponenty CheckBox oraz Panel.
2. Zmień etykiety komponentów CheckBox na widoczny Panel1 oraz widoczny Panel2.
3. Zmień właściwość Visible komponentów Panel1 oraz Panel2 na False.
4. Podepnij pod przyciski wyboru odpowiednie instrukcje:
• Dla pierwszego komponentu CheckBox:
if panel1.Visible=false then panel1.Visible:=true else panel1.Visible:=false;
• Dla drugiego komponentu CheckBox:
if panel2.Visible=false then panel2.Visible:=true else panel2.Visible:=false;
5. Sprawdź działanie programu.



Zadanie 4

Napisz aplikację, która będzie zawierać trzy przyciski wyboru oraz panel. Przyciski wyboru opisz nazwami kolorów: czerwony, zielony, niebieski. Po wyborze przez użytkownika kolorów oraz naciśnięciu przycisku „Pokoloruj panel", nastąpi malowanie panelu kolorem wygenerowanym przez makro RGB. (15 minut)

Podpowiedź: Makro RGB (r, g, b: byte) generuje kolor o podanych składowych r, g, b (red, green, blue). Wartości parametrów od 0 do 255 oznaczają intensywność określonej składowej w kolorze wynikowym. Powinieneś zadeklarować zmienne r, g i b w programie i w zależności od tego czy użytkownik zaznaczy wybrany kolor zmiennym, tym przypisywać wartość 0 lub 255. Później należy przemalować panel instrukcją: Panel1.Color:=RGB (r, g, b);

Zadanie 5

Zapisz wszystkie wykonane przez siebie zadania w katalogu nazwanym swoim nazwiskiem i imieniem oraz skopiuj go na serwer (sbs2005) do folderu Programowanie 3Ti. (5 minut)

Po lekcji powinieneś umieć:

- scharakteryzować poznane komponenty,
- wykorzystać poznane komponenty w programach komputerowych,
- samodzielnie tworzyć notatki oraz budować programy komputerowe w oparciu o nowe wiadomości.

Przedmiot: Programowanie strukturalne i obiektowe
Temat: Przyciski wyboru - ćwiczenia
Czas: 1 jednostka lekcyjna
Cele lekcji:

- zapoznanie z nowymi komponentami,
- zapamiętanie informacji o ich cechach, przeznaczeniu i możliwościach wykorzystania,
- ćwiczenie umiejętności samodzielnego tworzenia programów z wykorzystaniem poznanych komponentów,
- wyrabianie i kształtowanie umiejętności samodzielnego uczenia się.

Zadanie 1

Przypomnij sobie ostatnio poznane komponenty związane z przyciskami opcji i wyboru. Jak się one nazywają w Delhi? Jaka jest różnica w ich działaniu? Odpowiedź wraz z tematem zapisz w zeszycie. (5 minut)

Zadanie 2

Utwórz program, który za pomocą komponentów CheckBox pozwoli zmieniać styl czcionki w polu RichEdit. Dostępne style czcionek to: pogrubiona, pochylona, podkreślona i przekreślona. Po zaznaczeniu odpowiednich przycisków wyboru nastąpi zmiana stylu czcionki w polu RichEdit. (15 minut)

Podpowiedź: Dostępne style czcionki dla pola RichEdit to: fsBold – pogrubiona, fsItalic – podkreślona, fsUnderline – podkreślona, fsStrikeout – przekreślona, np.:

- dodanie do istniejącego stylu czcionki pogrubienia: RichEdit1.Font.Style:= RichEdit1.Font.Style+[fsBold];
- usunięcie z istniejącego stylu czcionki pochylenia: RichEdit1.Font.Style:= RichEdit1.Font.Style-[fsItalic];

Pamiętaj, że nazwy poszczególnych stylów należy podawać w nawiasach kwadratowych.

Zadanie 3

W oparciu o poprzedni program, dodaj do aplikacji dwie grupy przycisków opcji, ustalające w polu RichEdit kolor wyświetlanego tekstu i kolor tła. Użyj właściwości RichEdit1.Font.Color oraz RichEdit1.Color oraz stałych opisujących wybrane kolory. (15 minut)

Zadanie 4

Zapisz wszystkie wykonane przez siebie zadania w katalogu nazwanym swoim nazwiskiem i imieniem oraz skopiuj go na serwer (sbs2005) do folderu Programowanie 3Ti. (5 minut)

Po lekcji powinieneś umieć:

- scharakteryzować poznane komponenty,
- wykorzystać poznane komponenty w programach komputerowych,
- samodzielnie tworzyć notatki oraz budować programy komputerowe w oparciu o nowe wiadomości.