Skrypt do nauki języka c++

Na podstawie i z wykorzystaniem przykładów z książek:

1. **Symfonia c++ Standard – Jerzy Grębosz**
2. **Od podstaw Visual C++ 2005 – Ivor Horton**

Z własnymi ćwiczeniami

T. Roszczyk

Spis treści

1 Środowisko programu

Środowisko programistyczne Visual C++. Okno IDE składa się z:

- Okna Solution Explorer (Menedżera Rozwiązań),
- Edytora kodu
- okna Output, które pokazuje co dzieje się aktualnie w naszym programie. Właśnie w oknie Output wyświetlają się komendy debuggera w tym komunikaty o błędach w kodzie naszego programu.

Dwuklik na komunikacie błędu przenosi nas do okna edytora, gdzie wskazany jest wiersz, w którym kompilator napotkał błąd.

Można włączyć numerowanie linijek kodu.

Tools>Options>Text Editor>C/C++>Line Numbers

Edytor koloruje wprowadzany tekst dzięki czemu od razu możemy się zorientować jeśli popełnimy jakieś błędy.

Zielony – kolor komentarzy

Brązowy – kolor tekstu

Niebieski – słowa kluczowe

2 Nowy projekt

Spróbuj założyć pierwszy projekt w Visual c++.

File>**New**>**Project**>**Win32**>**Win32 Console Application**>

W **Location** wybierz **Browse** i wskaż **SWÓJ** katalog na dysku lokalnym.

Określ nazwę (**Name**) Twojego projektu.

Przejdź (**Next**) do **Application Settings**.

Tam wybierz **Console Application** i **Empty Project**. > **FINISH**

W Solution Explorer (po lewej) kliknij prawym na **Source Files**>**Add**>**New Item**>**Code**>**C++ File(.cpp)**

Określ nazwę w polu **Name**.>>**Add**

W okienku program.cpp będziesz pisał swoje programy, które można skompilować i uruchomić poprzez kliknięcie znaczka Play (Zielony trójkącik na pasku narzędzi) albo **Build**>**Build**

SWÓJ - to katalog założony przez Ciebie na HDD, który ma nazwę zgodną z Twoim nazwiskiem lub loginem.

3 Piszemy kod źródłowy

Pierwszy program może wyglądać tak:

```
#include <iostream>
int main()
{
	std::cout << "Witam";
	system ("pause");
}
```

Include – załącza do naszego projektu standardową bibliotekę input output stream (operacje wejścia/wyjścia)

Int main () – to rozpoczęcie kodu programu

{ - rozpoczyna ciało programu, czyli właściwy kod źródłowy. Na końcu procedury klamra powinna być zamknięta - }

Std – standard – informuje kompilator, że chcemy skorzystać ze standardowego zestawu poleceń

Dwa dwukropki otwierają zestaw poleceń i następuje polecenie z tego zestawu cout (czyt. Si-out) skrót od console output. Polecenie to służy do wyprowadzenia tekstu na ekran.

Znak << to łącznik, który pozwala wyświetlić po sobie tekst i zmienne.

System(„pause”) pozwala nam zobaczyć efekty działania naszego programu. Gdybyśmy nie kazali systemowi się zatrzymać okno aplikacji zamknęło by się bardzo szybko po wykonaniu ostatniej komendy i nic nie zdążylibyśmy zobaczyć.

Na końcu każdej linijki musi znaleźć się średnik - ;

4 Znaki specjalne

Znaki specjalne pomagają nam przede wszystkim formatować tekst w oknie konsoli.

Są to:

\b – cofnięcie (backspace)

\f – nowa strona

\n – nowa linia

\r – powrót karetki

\t – tabulator poziomy

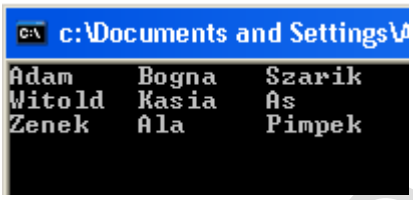
\v – tabulator pionowy

\a – sygnał dźwiękowy

\ - ten znak stosujemy dla zapisania znaku, który jest zwykle znakiem sterującym.

Zadanie: Wykonaj program, który wypisze na ekranie w trzech kolumnach kolejno imiona męskie, żeńskie i zwierzęce.

Przykład:



Rozwiązanie

5 Komentarze

Komentarze ułatwiają zorientowanie się w kodzie źródłowym napisanym przez inną osobę bądź nam samym przypomnienie przeznaczenia fragmentu kodu po jakimś czasie.

Komentarz w linijce kodu możemy napisać stosując dwa ukośniki.

//komentarz

Kompilator nie interpretuje kodu, który znajdzie po dwu ukośnikach do końca linijki

Można również wyłączyć z kompilacji całe bloki tekstu w kodzie źródłowym używając znaków

/*

Np.

```
/* To jest komentarz do programu i kompilator
nie będzie go analizował. Komentarze są kolorowane

przez edytor na zielono.*/
```

Dzięki takim wyłączeniom możemy zarówno umieścić dłuższy komentarz jak i pomóc sobie podczas testowania programu wyłączając z niego całe bloki kodu.

6 Zmienne

Zmienne deklarujemy za pomocą komendy, która od razu definiuje zakres zmiennej. Komend tych jest więcej:

NAZWA TYPU Danych	ZAKRES WARTOŚCI
int	**Zależne od systemu**
unsigned int	**Zależne od systemu**
__int8	**–128 do 127**
__int16	**–32,768 do 32,767**
__int32	**–2,147,483,648 do 2,147,483,647**
__int64	**–9,223,372,036,854,775,808 do 9,223,372,036,854,775,807**
char	**–128 do 127**
unsigned char	**0 do 255**
short	**–32,768 do 32,767**
unsigned short	**0 do 65,535**
long	**–2,147,483,648 do 2,147,483,647**
unsigned long	**0 do 4,294,967,295**
enum	**Takie samo jak int**
float	**3.4E +/- 38 (7 cyfr)**
double	**1.7E +/- 308 (15 cyfr)**
long double	**1.2E +/- 4932 (19 cyfr)**

W naszych programach najczęściej będziemy używali zmiennej typu INT, która zapamiętuje liczby całkowite z przedziału -32768 do 32768. Oczywiście zakres zmiennej zależy od użytego kompilatora.

Zmienne powinno się deklarować na początku programu, zaraz za main. Jest jednak również możliwe zdefiniowane zmiennej w locie. Przykład deklaracji zmiennej o zakresie int.

```
Int zak
```

Jest do deklaracja zmiennej o nazwie zak i zakresie int.

Możemy zadeklarować w jednej linijce więcej zmiennych o takim samym zakresie.

Np.

```
int a, x, zmienna, zak;
```

Dzięki takiemu zapisowi w jednej linijce zadeklarowaliśmy cztery zmienne używając tylko raz polecenia int.

Nazwą zmiennej w c++ może być dowolnie długi ciąg liter, cyfr oraz znaków podkreślenia. Nie może się jednak zaczynać od cyfry. Nazwa nie może być także żadnym ze słów kluczowych języka C++.

7 Wprowadzanie wartości zmiennych

Zmienne możemy wprowadzić z ekranu za pomocą:

Std::cin – Console Input

Oczywiście zanim pozwolimy wprowadzić zmienną należy ją zadeklarować.

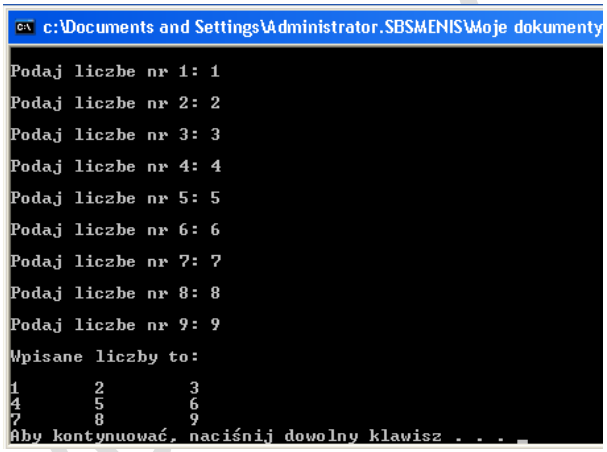
Przykład programu pytającego o zmienną i wypisujący jej wartość na ekranie.

```
#include <iostream>

int main()
{
	int x;
	std::cout << "\nPodaj liczbe: ";
	std::cin >> x;
	std::cout<< "\nWpisana liczba to " << x << "\n\n" ;

system ("pause");

}
```

Zadanie 1 Wprowadzanie wartości zmiennych

Wykonaj program, który zapyta użytkownika o dziewięć liczb a następnie wyświetli je na ekranie w trzech kolumnach i trzech wierszach.

Przykład:



Rozwiązanie

Zadanie 2

Wykonaj program, który zapyta użytkownika o dwie liczby a następnie wyliczy ich iloczyn i wyświetli na ekranie.

Zadanie 3

Wykonaj program, który będzie przeliczał cale na centymetry. 1cal = 2,54cm.

8 Instrukcje sterujące

Najważniejsza instrukcja warunkowa IF.

Przykład:

```
#include <iostream>
int main()
{
int i;
std::cout << "Podaj liczbe: ";
std::cin >> i;
if(i>4)
{
	std::cout << "zmienna i > 4";
}
else
{
	std::cout << "zmienna i <= 4\n\n";
}
system ("pause");

}
```

Sprawdzamy wartość zmiennej i, która jest pobierana z klawiatury od użytkownika programu.

Jeśli wartość jest większa niż 4 to prawdziwe jest stwierdzenie przy funkcji if (i>4) i wtedy wykona się pierwsze polecenie. Jeśli wprowadzona liczba będzie mniejsza lub równa 4 to wykona się polecenie po

```
else
```

czyli

```
	std::cout << "zmienna i <= 4\n\n";
```

Jeśli chcesz wielokrotnie sprawdzać różne liczby możesz zastosować najprostszą pętlę

```
goto
```

Za pomocą tej funkcji można wskazać, którą część kodu ma od tego momentu wykonywać program. W miejscu, w które chcemy skierować wykonywanie programu wstawiamy znacznik. Może to wyglądać tak:

```
#include <iostream>
int main()
{
int i;

start:

std::cout << "Podaj liczbe: ";
std::cin >> i;
if(i>4)
{
	std::cout << "zmienna i > 4\n\n";
}
else
{
	std::cout << "zmienna i <= 4\n\n";
```

```
}
system ("pause");

goto start;

}
```

W powyższym programie wstawiono znacznik o nazwie:

```
start:
```

po sprawdzeniu wartości liczby i wyświetleniu komunikatu, za pomocą instrukcji

```
goto start;
```

kierujemy dalsze wykonywanie programu do linijki, w której znajduje się znacznik start.

Program będzie sprawdzał bez końca kolejne wpisywane liczby. Jego działanie można przerwać zamykając okno konsoli programu.

Można także stosować polecenie else if w przypadku jeśli chcemy sprawdzić kolejne warunki powiązane z if np. w sytuacji kiedy rozpoznajemy kilka zakresów dla liczby. Patrz zadanie 1. Alternatywnym rozwiązaniem jest zagnieżdżenie funkcji if w innej funkcji if.

Zadanie 1: Wykonaj program, który sprawdzi jaką liczbę wprowadził użytkownik. Niech program rozpoznaje trzy przedziały:

1. <10
2. 10 do 100
3. >100

Niech na ekranie pojawiają się stosowne komunikaty w zależności od wielkości liczby.

Rozwiązanie

Zadanie 2: Przyjmij od użytkownika dwie liczby. Pomnóż przez 10 mniejszą z liczb wpisanych przez użytkownika. Wyświetl wynik mnożenia oraz liczbę, która została użyta do mnożenia.

Rozwiązanie

Zadanie 3: Przyjmij od użytkownika trzy liczby. Wypisz na ekranie najmniejszą liczbę.

Rozwiązanie

Zadanie 4: Przyjmij od użytkownika trzy liczby. Wypisz liczby na ekranie w kolejności od najmniejszej do największej.

Rozwiązanie (za pomocą sortowania bąbelkowego).

Rozwiązanie (sortowanie bąbelkowe realizacja - Piotr Wacławczyk)

9 Zmienne logiczne jako warunek.

Często w programie sprawdzamy wielokrotnie tą samą zmienną. Każde sprawdzenie wiąże się z użyciem procesora i za każdym razem wykonywane są te same operacje. Można zatem wykonać sprawdzenie raz a wynik zapisać w innej zmiennej.

Załóżmy, że sprawdzamy czy użytkownik jest pełnoletni. Wiek użytkownika zapiszemy do zmiennej x. W każdym miejscu programu musielibyśmy sprawdzać wartość x np. x<18. Lepiej jest zatem na początku programu sprawdzić warunek a jego rezultat zapisać w zmiennej o małym zakresie np. bool.

```
#include<iostream>
int main()
{
	int x;
	bool niepelnoletni; //definicja obiektu

	std::cout << "Podaj ile masz lat: ";
	std::cin >> x;

	niepelnoletni = (x<18); //zapamietanie rezultatu

	std::cout << "wartosc niepelnoletni to: " << niepelnoletni << "\n\n";
	system ("pause");
}
```

Zmienna niepełnoletni będzie miała wartość 1 jeśli wyrażenie x<18 będzie prawdziwe, a jeśli wyrażenie będzie nieprawdziwe to wartość zmiennej będzie równa 0.

10 Pętle

10.1 Pętla While

Pętla ta jest wykonywana tak długo dopóki spełniony jest jakiś warunek. W poniższym programie wyświetlamy zadaną przez użytkownika ilość wykrzykników.

```
#include<iostream>
int main()
{
	int ile;
	bool niepelnoletni; //definicja obiektu

	std::cout << "Ile wykrzyknikow wyswietlic?: ";
	std::cin >> ile;
	std::cout << "Wyswietlamy: " << ile << " : ";

while (ile)//pętla wyświetlająca wykrzykniki
{
	std::cout << "!";
	ile=ile-1;
}
//wyświetlamy sobie wartość ile, aby sprawdzić czy program miał prawo zakończyć pętlę
std::cout << "\n teraz zmienna ile ma wartosc: " << ile << "\n\n";
system ("pause");
}
```

Pętla while sprawdza, czy zmienna ile jest większa od zera.

10.2 Pętla Do while

Po polsku do while oznacza „rób dopóki”. Od pętli while różni się tym, że uwarunkowana pętlą instrukcja wykona się co najmniej raz. Dopiero później zostanie sprawdzony warunek i od tego sprawdzenia uzależnione będzie ponowne wykonywanie instrukcji.

Poniższy program sprawdza jaką literę użytkownik wpisał z klawiatury. Jeśli zostanie wpisana litera K to program zostanie zakończony. Proszę zwrócić uwagę na to, że zmienna została zadeklarowana jako char, czyli zmienna tekstowa.

```
#include <iostream>
int main()
{
	char litera;
	do {
		std::cout << "Napisz jakas litere: ";
		std::cin >> litera;
		std::cout << "\n Napisales: " << litera << " \n";
	}while(litera != 'K');
		// `1

	std::cout << "\n Skoro Napisales K to konczymy !";
}
```

10.3 Pętla for

Jest to pętla, która wykonuje się określoną z góry ilość razy.

```
#include <iostream>
int main()
{
	std::cout << "Stewardzie, ilu leci pasazerów ? ";

	int   ile;              // liczba pasażerów
	std::cin >> ile;

	int i;                  // licznik obiegów pętli`1
	for(i = 1 ; i <= ile ; i = i + 1)        //`2
	{
		std::cout   << "Pasazer nr " << i
                            << " prosze zapiac pasy ! \n";
	}

	std::cout << "Skoro wszyscy juz zapieli, to ladujemy. ";
}
```

Program ten wypisuje polecenia zapięcia pasów dla każdego z pasażerów. Pętla wykonuje się „ile" razy.

Są trzy argumenty funkcji for. Zapisuje się je wewnątrz nawiasu zwykłego i oddziela średnikami.

- pierwszy argument to nadanie wartości początkowej jakiejś zmiennej (najczęściej licznika, od którego zależna będzie ilość przebiegów pętli) Wykonywany jest jednokrotnie
- drugi to warunek, który musi być spełniony aby pętla się wykonywała. Warunek jest sprawdzany w każdym przebiegu pętli.
- Trzeci to instrukcja kroku pętli wykonywana na zakończenie każdego przebiegu pętli. Najczęściej zwiększamy lub zmniejszamy zmienną, aby kontrolować ilość kroków wykonania pętli.

Zadanie 1

Napisz program z wykorzystaniem pętli, który będzie wyświetlał kolejne potęgi zadanej liczby (od potęgi zerowej do potęgi dziesiątej).

Opracuj komunikaty w taki sposób, aby użytkownik programu wiedział co program liczy.

Przykład wyświetlenia poszczególnych potęg liczby 2.

2 do potęgi 0 = 1

2 do potęgi 1 = 2

2 do potęgi 2 = 4

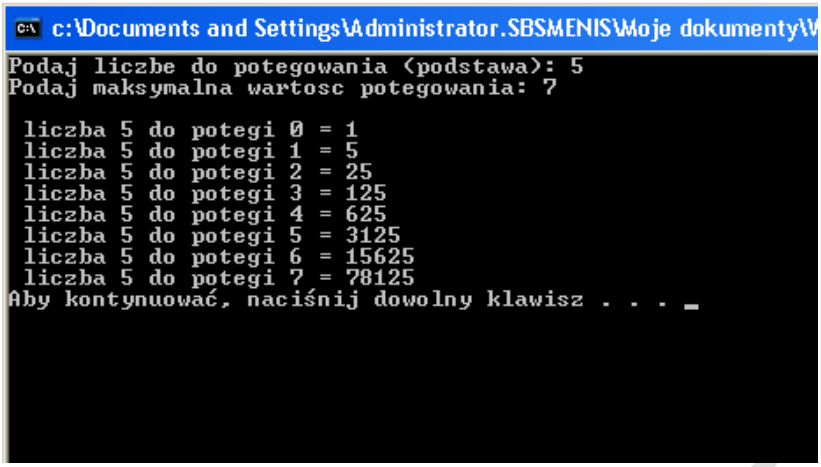
2 do potęgi 3 = 8

2 do potęgi 4 = 16

Zadanie 2

Zmodyfikuj program z poprzedniego zadania tak aby użytkownik nie tyko podawał liczbę, która będzie podstawą potęgowania, ale także maksymalny wykładnik potęgowania, na którym pętla zakończy operację.

Przykład:



Rozwiązanie

Rozwiązanie z załączeniem biblioteki math.h

Zadanie 3

Wykonaj program, który będzie obliczał pola powierzchni kół o promieniu powiększającym się o 1, w zakresie podanym przez użytkownika. $P=Pi * r^2$. Liczbę PI możemy przyjąć jako 3,14.

Rozwiązanie

Zadanie 4

Wykonaj program, który obliczy sumę liczb całkowitych z przedziału od **a** do **b**, gdzie **a** i **b** podaje użytkownik. Niech program sprawdzi czy **b**>**a** i jeśli warunek nie będzie spełniony niech program nie wykonuje obliczeń tylko ponownie zapyta o wartość **b** albo ponownie zapyta o obie wartości **a** i **b**.

Rozwiązanie

11 Switch

Funkcja switch pozwala na podjęcie wielowariantowej decyzji.

Oczywiście możemy użyć funkcji if, ale często użycie switch case będzie wygodniejsze.

Poniższy program pozwala wybrać jakiego typu obliczenie chcemy wykonać.

Przykład 1

```
#include <iostream>
int main()
{
	int stopy;
	int metry;
	double przelsm = 0.3;
	double przelms = 3.33;
	int spr;

start:
	std::cout << "Przeliczanie: \n 1) ze stop na metry, \n 2) z metrow na
stopy \n";
	std::cin >> spr;

	if (spr < 2)
	{
		std::cout << "Wartosc w stopach = ";
		std::cin >> stopy;
		metry = stopy * przelsm;
		std::cout << "To jest " << metry << "m. \n";
	}
	else
	{
		std::cout << "Wartosc w metrach = ";
		std::cin >> metry;
		stopy = metry * przelms;
		std::cout << "To jest " << stopy << "stop. \n";
	}
system ("pause");
std::cout << "\n \n";
goto start;
}
```

To samo możemy zrealizować za pomocą switch case:

Przykład 2

```
#include <iostream>
int main()
{
	double stopy, metry;
	double przelsm = 0.3;
	double przelms = 3.33;
	int spr;

start:
```

```
	std::cout << "Przeliczanie: \n 1) ze stop na metry, \n 2) z metrow na
stopy \nCo liczymy? :";
	std::cin >> spr;

	switch (spr)
	{
	case 1:
		std::cout << "Wartosc w stopach = ";
		std::cin >> stopy;
		metry = stopy * przelsm;
		std::cout << "To jest " << metry << "m. \n";
		break;
	case 2:
		std::cout << "Wartosc w metrach = ";
		std::cin >> metry;
		stopy = metry * przelms;
		std::cout << "To jest " << stopy << "stop. \n";
		break;
	default:
		std::cout << "Nie ma takiego wyboru!\n\n";
		break;
	}

system ("pause");
std::cout << "\n \n";
goto start;
}
```

Funkcja break pozwala na wyjście z klamry zawierającej warianty switch-case. Break nie jest obowiązkowe. Jeśli je pominiemy to po wykonaniu instrukcji przewidzianych dla naszego przypadku będą się wykonywały wszystkie kolejne instrukcje do końca klamry.

Jak widać instrukcję switch lepiej wybrać jeśli chcemy stworzyć menu programu, kiedy użytkownik powinien wybrać jedną z wielu opcji i używa do tego liczb całkowitych. Funkcja if będzie się lepiej sprawdzała przy sprawdzaniu przedziałów liczbowych i liczb niecałkowitych.

Wykonaj test. Uruchom pierwszy przykład i wpisz wybór = 4. Jak widzisz wykonuje się obliczenie numer 2 zgodnie z nierównością w funkcji if. W drugim przykładzie program poinformuje nas, że nie możemy dokonać takiego wyboru.

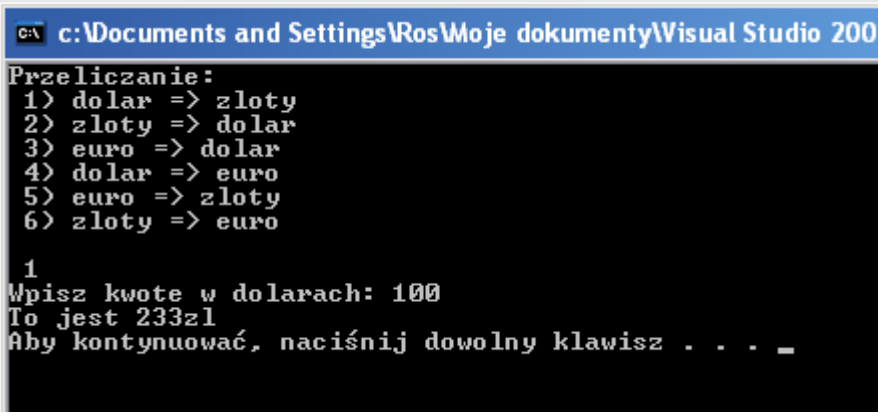
Zadanie 1

Wykonaj program, który będzie przeliczał kwoty pieniędzy pomiędzy różnymi walutami, w oparciu o zdefiniowane w nim kursy.

Waluty, które będziemy przeliczać to Euro, Dolar i Złoty.

Na początku działania programu ma pojawić się menu, z którego wybierzemy jakiego przeliczenia chcemy dokonać.

Przykład:



Rozwiązanie

Rozwiązanie z podejściem strukturalnym

Zadanie 2

Wykonaj program, w którym możliwe będzie przeliczanie odległości pomiędzy różnymi miarami. Na początku program ma zapytać co użytkownik chce przeliczać.

1. Ze stóp na metry (1 stopa=30,48cm) i z metrów na stopy.
2. Z km na mile (1 mila=1,609km) i z mil na km

Sprawdzian nr 1 - podstawy

Link do sprawdzianów na końcu skryptu

12 Dyrektywa using

Za pomocą dyrektywy możemy powiązać przestrzeń nazw z naszym programem, a zatem poinformować kompilator, że jeśli jakaś nazwa jest mu nieznana to ma sprawdzić w przestrzeni nazw zadeklarowanej w dyrektywie.

Przykład:

Dyrektywa

```
using namespace std
```

spowoduje, że będziemy mogli używać nazwy cout i cin bez deklaracji przestrzeni nazw. Zatem w kodzie programu zamiast pisać:

```
std::cout
```

napiszemy

```
cout
```

Możemy zadeklarować także użycie pojedynczych funkcji a nie całej przestrzeni nazw.

Przykład

```
using std::cout
using std::cin
```

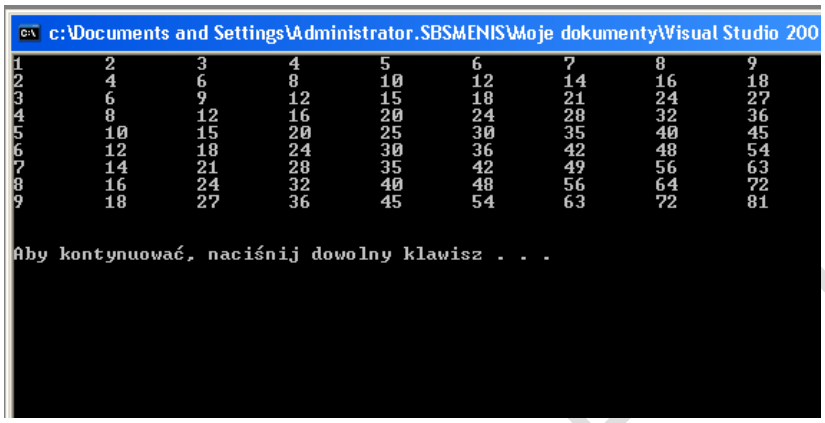
spowoduje, że nazw cin i cout będziemy mogli używać bez deklarowania przestrzeni nazw jednak inne nazwy z przestrzeni nazw std będziemy musieli deklarować jak dotychczas.

13 Zagnieżdżanie pętli for (pętle wielokrotne)

Zadanie

Za pomocą znaków specjalnych i pętli wyświetl tabliczkę mnożenia do 9*9. Po wyświetleniu tabelki ma się odezwać sygnał dźwiękowy.

Tabela: Tabliczka mnożenia do 9*9



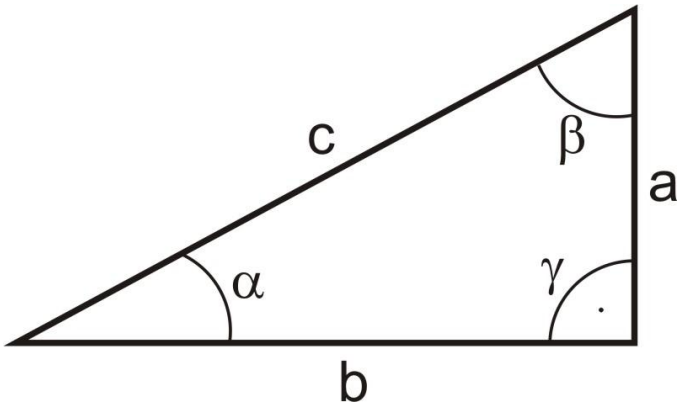
Rozwiązanie exe

Kod źródłowy

Zadanie

Wykonaj program, który będzie obliczał tangensy kąta α dla kolejnych powiększających się długości boków. Wzór na tangens α to:

$$tg\alpha = \frac{a}{b}$$



Zapytaj użytkownika o długość boku a oraz o wielkość x, która będzie stanowiła przedział dla wartości boku b (od –x do +x). Np. użytkownik podaje x= 10 to program ma obliczyć tangensy dla b = -10, b = -9 itd. Aż do b=10.

Dla boku b równego 0 (zero) obliczenie tangensa zwróci błąd obliczenia. Wymyśl takie rozwiązanie aby uniknąć wyświetlenia tego błędu.

Rozwiązanie niezbyt dobre

Rozwiązanie prawidłowe

14 Funkcje

Przykład programu zawierającego funkcję o nazwie kukulka.
Funkcja ta została zdefiniowana na końcu kodu.
W programie została wywołana poleceniem kukulka
Zmienna m została użyta po to aby stwierdzić, że funkcja została wywołana.

Skopiuj poniższy kod do edytora Vcpp i prześledź działanie.

```
//*****************************************************
// Program z paragrafu  5.0 (str 132)
//*****************************************************
// Sprawdzony na Linuksie, kompilator: GNU gcc version 3.3.3 (SuSE Linux)
// Sprawdzony na Windows XP, kompilator: Microsoft Visual C++ 6.0

#include <iostream>
using namespace std ;

int kukulka(int ile);          //
/************************************************/
int   main()
{
 int m = 20 ;
 cout << "Zaczynamy" << endl ;

 m = kukulka(5) ;            //
 cout << "\nNa koniec m = " << m ;          //


}
/*****************************************************/
int kukulka(int ile)                    //
{                                  //
 int i ;
 for(i = 0 ; i < ile ; i++)
 {
  cout << "Ku-ku ! " ;
 }
 return 77 ;                        //
}                                   //

/************************************************************

************************************************************/
```

Zadania
Zmodyfikuj program zawierający funkcję kukułka.
1. niech Ku-ku! wyświetli się 50 razy zamiast pięciu.
2. Niech m ma na końcu wartość 187 zamiast 77.

3. Napisz jeszcze jedna funkcje oprócz funkcji kukulka. Niech funkcja nazywa się pies i po jej wywołaniu na ekranie pokazuje się napis "hau-hau".
Program powinien działać w ten sposób, że na początku zapyta użytkownika o liczbę a następnie podaną liczbę razy wykona funkcję kukulka, następnie taką samą liczbę razy wykona funkcję pies, a następnie znowu wykona taką samą liczbę razy funkcję kukulka.

Rozwiązanie

14.1 Zwracanie rezultatu przez funkcję

Wypróbuj poniższy kod:

```
//*************************************************
// Program z paragrafu  5.2 (str 136)
//*************************************************
// Sprawdzony na Linuksie, kompilator: GNU gcc version 3.3.3 (SuSE Linux)
// Sprawdzony na Windows XP,  kompilator: Microsoft Visual C++ 6.0
#include <iostream>
using namespace std;
long potega(int stopien, long liczba);
//*************************************************************
int main()
{
      int pocz, koniec;

      cout << "Program na obliczanie poteg liczb"
                  << "calkowitych\n"
                  << "z zadanego przedzialu \n"
                  << "Podaj poczatek przedzialu: ";
      cin >> pocz;

      cout << "\nPodaj koniec przedzialu: ";
      cin >> koniec;

      // pętla drukująca wyniki z danego przedziału
      for(int i = pocz ; i <= koniec ; i++)
      {
            cout << i
                        << " do kwadratu = "
                        << potega(2, i)
      // <-wywołanie funkcji
                        << " a do szescianu = "
                        << potega(3, i)                              //
<-wywołanie funkcji
                        << endl;
      }
system("pause");
}
//*************************************************************
long potega(int stopien, long liczba)
{
      long wynik = liczba;
      for(int i = 1 ; i < stopien ; i++)
      {
            wynik = wynik * liczba;
            // zwięźlej można zapisać to samo jako:  wynik *= liczba;
      }
```

```
      return wynik;     // `1
}
//************************************************************
```

Zwracani wartości funkcji odbywa się przez funkcję **return.** W powyższym przykładzie **return wynik**.

Dotychczas nie stosowaliśmy tej komendy mimo, że w funkcji main rozpoczynającej procedurę jest określony rezultat (int). Zatem funkcja main powinna zwracać jakiś wynik. Dzieję się tak dlatego, że kompilator napotykając klamrę zamykającą **main**, przyjmuje domyślnie funkcję **return 0**.

Zadanie

Zmodyfikuj powyższy program. Dopisz funkcję pierwiastek.

Podczas wyświetlania (drukowania) kwadratu i sześcianu niech wyświetli się również pierwiastek liczby.

rozwiązanie kod źródłowy

14.2 Przesyłanie argumentu funkcji przez wartość.

Na przykładzie prześledzimy przesyłanie argumentów funkcji przez wartość.

Załóżmy, że wewnątrz programu mamy funkcję „alarm”.

```
void alarm (int stopien, int wyjscie)
{
      cout <<  ”Alarm ” << stopien <<  “stopnia ” << “skierowac sie do
wyjscia nr “ << wyjscie << endl;
}
```

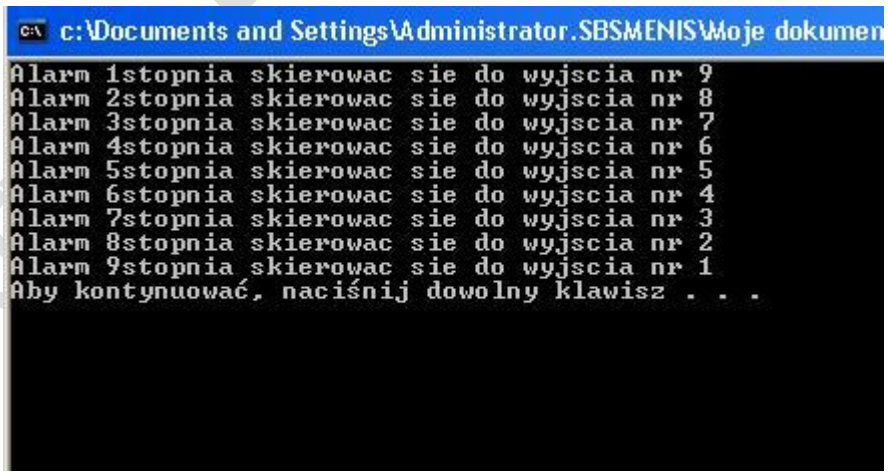
Załóżmy ,że w programie wywołujemy tę funkcję tak:

```
int a, m;
alarm (1,10);
alarm (a,m);
```

Stopien i wyjscie nazywamy argumentami formalnymi funkcji, natomiast 1, 10, a, m to tak zwane argumenty (parametry) aktualne, czyli te z którymi funkcja ma wykonać pracę (tzw argumenty wywołania funkcji).

Argumenty przesłane do funkcji nie zmieniają oryginału.

Ćwiczenie:
Na bazie powyższego przykładu stwórz program, który będzie przesyłał do funkcji alarm wartości zmieniające się o jeden, w ten sposób, że stopień alarmu będzie rósł od 1 do 9 a numer wyjścia będzie malał od 9 do 1.
Wykorzystaj argumenty a i m.
Wynikiem działania programu ma być następujący komunikat:



rozwiązanie

15 Klasyczne algorytmy iteracyjne

Wyszukiwanie najmniejszego elementu z ciągu liczb.

Liczba pierwsza czy złożona

16 Tablice

To ciąg obiektów tego samego typu, które zajmują ciągły obszar w pamięci. Tablice stosujemy w sytuacjach, w których potrzebujemy dużej ilości zmiennych o takim samym zakresie. Deklaracja tablicy jest znacznie wygodniejsza i mniej pracochłonna niż deklarowanie pojedynczych zmiennych.

Przykład deklaracji tablicowej zmiennych parametr o zawartości czternastu zmiennych (od 0 do 13):

```
int parametr[14]
```

Możemy od razu deklarując tablicę zdefiniować wartości zmiennych:

```
int parametr[14] = { 1, 6, 5, 6, 8, 9, 2, 4, 5, 1, 6, 6, 7, 8}
```

powyżej deklaracja czternastoelementowej tablicy, gdzie zdefiniowano wartości początkowe. Na przykład ósmy element tablicy „parametr" będzie równy cztery:

```
parametr[7] = 4
```

Można również przypisanie wykonać w taki sposób:

```
parametr[7] = 4
```

W miejsce numeru elementu tabeli możemy wstawić zmienną, której aktualna wartość będzie określała numer zmiennej w tabeli:

```
licznik = 7
parametr[licznik] = 4
```

powyższy kod przypisze zmiennej nr 7 (ósma zmienna) w tablicy wartość 4.

Zadanie 1: Załóżmy, że piszemy grę RPG. Nasz bohater opisany jest kilkunastoma różnymi wskaźnikami, które są zapisywane jako dane w tablicy. Naszym zadaniem jest zaprogramowanie wpływu na bohatera wydarzenia w grze. Niech to wydarzenie to będzie spotkanie ze smokiem, który zionął na naszą postać ogniem. W poniższej tabelce znajdziesz jakimi parametrami opisano bohatera oraz jak powinny się one zmienić pod wpływem smoczego ataku.

Lp	Nazwa parametru	zmiana pod wpływem ataku smoka przez zionięcie ogniem
1.	Poziom magii	+1
2.	Zdrowie	-3
3.	Siły witalne	-1
4.	Odwaga	-1
5.	Pewność siebie	-1
6.	Szybkość	-1
7.	Wytrzymałość	-1

8.	Siła fizyczna	-1
9.	Komfort psychiczny	-2
10.	Komfort fizyczny	-2
11.	Poczucie bezpieczeństwa	-2
12.	Zdolność bojowa	-2
13.	Spokój ducha	-2
14.	Radość z życia	-2

Początkowe parametry bohatera przyjmij dowolnie. Wyświetl je na ekranie przed atakiem smoka oraz po ataku. Sprawdź czy zdrowie spadło do zera lub poniżej. Jeśli tak, bohater ginie i powinien pojawić się odpowiedni komunikat.

Rozwiązanie proste z wykorzystaniem funkcji

Rozwiązanie kod źródłowy (wersja rozrywkowa)

Rozwiązanie exe

Zadanie 2: Rozbuduj zadanie 1. Walczymy ze smokiem!

Niech nasz bohater ma też możliwość działania. Możliwe działania to: atak na smoka i unik.

Smok niech też ma więcej możliwości: atak, unik, bezruch.

Niech smok zostanie opisany parametrami. Siła (Ss), zdrowie (Sz), szybkość (Ssz). Wielkość tych parametrów pozwól wpisać graczowi na początku, aby mógł wybrać z jak silnym smokiem chce się zmierzyć.

Niech działanie postaci zależy od gracza (wpisanie z klawiatury): A – atak, U-unik.

Działania smoka niech będą losowe.

W każdej turze i smok, i postać muszą wykonać jakieś działanie. Ta walka jest na śmierć i życie bo nie można uciec ani zrezygnować z walki, a kończy ją dopiero śmierć postaci lub smoka. Sprawdzamy w każdej turze poziom zdrowia postaci i smoka. Ginie ten, którego zdrowie będzie równe lub mniejsze niż zero.

Każdy z parametrów oprócz zdrowia musi mieć wartość większą lub równą jeden. W trakcie walki sprawdzamy te wartości i nie zmniejszamy ich jeśli są już równe jeden. Inaczej pojawi się błąd dzielenia przez zero oraz może być niemożliwe rozstrzygnięcie walki.

Tabelka możliwych akcji i punkty

smok	**Postać**	**Zmiany punktów**						
		Smoka			**Postaci**			
		zdrowie	**siła**	**Szybkość**	**Zdrowie**	**Szybkość**	**siła**	**Wytrzymałość**
		Sz	**Ss**	**Ssz**	**Pz**	**Psz**	**Ps**	**Pw**
Atak	Atak	Sz – (Psz*Ps*Pz) / (Ssz*Ss*Sz)	-1	-1	Pz - (Ssz*Ss*Sz) / (Pw*PS*Pz)	-1	-1	-1
Unik	Atak	0	-1	0	-1	0	0	-1
Atak	Unik	-1	-1	0	0	0	0	-1
Unik	Unik	0	-1	0	0	0	-1	0
Bezruch	Atak	Sz- (Psz*PS*Pz*Pw) / (Ssz*Ss*Sz)	-1	0	-1	0	0	-1
Bezruch	Unik	+1	+1	+1	0	0	-1	0

Pojedynek powinien być zakończony komunikatem:

- W przypadku zwycięstwa postaci – „Zwycięstwo - Zabiłeś Smoka!”
- W przypadku zwycięstwa smoka – „Zginąłeś! – SMOK wygrał!”

Oraz wyświetleniem wszystkich parametrów smoka i postaci.

16.1 Przygotowanie do sprawdzianu - Zadania z tablic.

Zadanie 1
Wykonaj program, który pobierze od użytkownika osiem zmiennych i wpisze je do tablicy. Następnie spytaj użytkownika o liczbę i przemnóż wszystkie elementy tabeli przez tę liczbę. Zamknij to działanie w pętlę tak aby możliwe było mnożenie początkowych (wpisanych do tablicy) liczb przez coraz to nowe mnożniki.
Zmienne wpisane pierwotnie do tablicy mają pozostać bez zmian.

Przykład:
Użytkownik wpisuje wartości: 1, 2, 3, 5, 8, 4, 16, 66
Te wartości program umieszcza w tablicy.
Następnie program pyta o mnożnik.
Użytkownik podaje: 3

Na ekranie powinny się wyświetlić liczby 3, 6, 9, 15, 24, 12, 48, 198
Natomiast wartości w tablicy powinny nadal być takie jak wpisano, czyli: 1, 2, 3, 5, 8, 4, 16, 66
Teraz program powinien znowu zapytać o mnożnik i wyświetlić na ekranie wynik mnożenia poszczególnych elementów tablicy przez ten mnożnik.

Rozwiązanie problemu: pobieranie danych w pętli

Zadanie 2
Zmień program z zadania 1 tak aby po przemnożeniu przez liczbę do tablicy zostały wpisane wartości po przemnożeniu. Daj możliwość wyboru działania, które wykonasz na wszystkich liczbach zapisanych w tablicy. Niech będzie możliwe także wielokrotne powtarzanie operacji na liczbach w tablicy.

Przykład:
Użytkownik wpisuje wartości: 1, 2, 3, 5, 8, 4, 16, 66
Te wartości program umieszcza w tablicy.
Następnie program pyta o działanie za pomocą menu.
Załóżmy, że użytkownik wybiera odejmowanie
Program pyta o liczbę.
Użytkownik podaje: 3
Wartości w tablicy powinny się zmienić na: -2, -1, 0, 2, 5, 1, 13, 63
Na ekranie powinny się wyświetlić liczby -2, -1, 0, 2, 5, 1, 13, 63
Teraz program powinien znowu zapytać o działanie a następnie o liczbę i ponownie zmienić wartości wszystkich elementów tabeli.
Podpowiedź do zadania 2

Zadanie 3
Wpisz kolejne wielokrotności podanej przez użytkownika liczby do tablicy. Zakończ wpisywanie do tablicy w momencie jeśli wartość wielokrotności przekroczy 1000.
Przykład:
Użytkownik podaje liczbę 6
W tablicy należy wpisać 6, 12, 18, 24 itd… Ostatnią liczbą wpisaną do tablicy powinna być 996, ponieważ następna wartość to 1002 > 1000.

Rozwiązanie:
Krok 1 – przerabiamy podpowiedź z zadania 2 tak aby wpisywać wielokrotności podanej liczby do tablicy.
Krok 2 – Zastanowić się ile miejsca powinno być w tablicy, aby wpisać wszystkie dodatnie wielokrotności liczby całkowitej.
Krok 3 – Jak kontrolować pętlę, żeby skończyła się jak liczba przekroczy odpowiednią wartość?
Zrobione poprzez zmianę pętli for. Można też zastosować pętlę while.

Zadanie 4
Rozszerz możliwości programu z zadania 3 tak aby wpisanie liczby ujemnej nie powodowało nieskończonej pętli. Wersje:

1. Wstawiaj do tabeli wartość bezwzględną liczby zamiast liczby podanej przez użytkownika

2. Sprawdź wprowadzoną liczbę i odrzuć ją jeśli jej wartość nie będzie się zawierała w przedziale od 1 do 1000.

16.2 Tablice wielowymiarowe

Tablice mogą mieć więcej wymiarów. Oto przykład wypełnienia tablicy trójwymiarowej o wielkości 3x3x3 elementy.

Przykład

16.3 Przeszukiwanie tablicy jednowymiarowej

16.4 Przeszukiwanie tablicy z wartownikiem

16.5 Zapisywanie liczb w różnych systemach liczbowych.

Przeliczanie

Zadanie 1

Wykonaj program przeliczający liczbę z systemu dziesiętnego na binarny.

Zadanie 2

Wykonaj program przeliczający liczbę z systemu dziesiętnego na dowolny inny system od 2 do 9. (Systemy o podstawie większej niż 9 są bardziej skomplikowane bo trzeba zdefiniować dodatkowe cyfry).

Rozwiązanie 1

Rozwiązanie 2

16.6 Sortowanie bąbelkowe elementów tablicy

Dotychczas sortowaliśmy elementy w sytuacji kiedy ich ilość była ściśle określona. Nauczmy się sortować tablice o dowolnej pojemności. Jeśli napiszemy uniwersalną funkcję sortowania to będziemy mogli przesłać jej zawartość dowolnej tablicy do posortowania.

Funkcja sortująca bąbelkowo może wyglądać tak:

```
void babelkowe(int *tab, int n)
{

  for (int i=0; i<n; i++)
    {
      for (int j=0; j<n-1; j++)
      {
        if (tab[j]>tab[j+1])
          {
            int tmp;
            tmp = tab[j];
            tab[j] = tab[j+1];
            tab[j+1] = tmp;
          }
      }
    }
}
```

Jak widać do funkcji należy przesłać tablicę oraz n – ilość elementów tablicy. Wywołanie takiej funkcji z programu może wyglądać tak:

```
babelkowe(tab, n);
```

Samo sortowanie wykonywane jest za pomocą zagnieżdżonych pętli, w których porównuje się sąsiadujące liczby dbając o to aby nie wyjść poza zakres n elementów tablicy.

Zadanie: Napisz program, który wykorzysta powyższą funkcję sortującą. Program powinien zapytać o ilość elementów tablicy, wprowadzić do tablicy elementy podane przez użytkownika a następnie wywołać funkcję sortującą i wyświetlić wynik sortowania, czyli wszystkie kolejne elementy tablicy.

Rozwiązanie

16.7 Sortowanie przez wybór.

Listing z podręcznika (5.19 na stronie 149) jest przeznaczony dla DevC++. W Visual C++ nie obsługiwana jest funkcja **time,** oraz kompilator nie przyjmuje niezdefiniowanej zmiennej **temp.** Program będzie działał prawidłowo po poprawkach:

1. Od razu po zadeklarowaniu zmiennej **temp** nadajemy jej wartość (definiujemy ją) np. int temp=0.
2. Linijkę zawierającą funkcję **time** kasujemy. Spowoduje to, że generowane liczby losowe będą zawsze takie same ponieważ funkcja `srand(time(NULL))` uzależnia losowo wpisywane liczby od czasu. Rozwiązaniem bardziej eleganckim będzie pozostawienie tej linijki i dołączenie biblioteki **ctime**, która zawiera funkcję **time**.

Listing programu poprawiony dla Visual C++.

16.8 Wykorzystanie tablic danych w typowych algorytmach ćwiczenia.

1. Napisz program, który będzie tworzył anagram prosty z podanego wyrazu.
2. Posortuj elementy tablicy pięcioelementowej metodą przez wybór.
3. Wypełnij tablicę dwuwymiarową (5x6) w ten sposób, że pierwszą liczbę każdej kolumny podasz z klawiatury a pozostałe elementy każdego wiersza niech będą wielokrotnościami podanej liczby.
4. Wylicz medianę wartości wpisanych do tablicy. Mediana to wartość środkowa w uporządkowanym zbiorze elementów. Przy parzystej liczbie elementów są dwa elementy środkowe. Wyświetl ten z niższym indeksem.
5. Wpisz losowe elementy do tablicy dwuwymiarowej (5x6). Następnie wyświetl całą tablicę i osobno wyświetl elementy najmniejsze w każdej kolumnie i każdym wierszu.
6. (zad na 6) Posortuj tablicę dwuwymiarową (5x6) dowolną metodą, tak aby liczby rosły w wierszach a następnie w kolumnach.
7. Wykonaj program, który pobierze adres IP w postaci dziesiętnej i wyświetli go w postaci binarnej.
8. Napisz program, który zamieni wszystkie samogłoski we wprowadzonym tekście na kropki.
9. Napisz program zliczający ilość znaków we wprowadzonym tekście.
10. Napisz program, który wypisze z tablicy dwuwymiarowej (5x6) wiersz, którego suma wprowadzonych liczb jest największa oraz poda jej wartość. Dla kontroli najlepiej jest wypisać na ekranie wszystkie elementy tablicy z sumami poszczególnych wierszy.

Sprawdzian teoria – algorytmy tablicowe

16.9 Wskaźniki

Wskaźniki to zmienne, które przechowują nie dane, ale ich adresy. Wskazują adres w pamięci komputera, gdzie znajduje się zmienna. Aby określić adres posługujemy się operatorem adresu & (ampersand).

Przykład:

```
#include <iostream>
using namespace std;
int main()
{
unsigned int zmienna1=9;
cout << "adres zmiennej w pamieci komputera to: " << &zmienna1 <<endl;
cout << "wartość wpisana do zmiennej to: " << zmienna1 << endl;
system ("pause");
}
```

Ćwiczenie:

Wykonaj program, który wpisze wartości pobrane od użytkownika do pięcioelementowej tablicy a następnie wypisze adresy przechowywania tych zmiennych tablicowych w pamięci wraz z ich zawartością.

Rozwiazanie

16.10 Struktury

Przykład struktury komp, która pozwala zdefiniować zapasy magazynowe komputerów w sklepie. Przykład Program zlicza również wartość wszystkich komputerów w sklepie.

Zadanie: Dokonaj przeróbki programu tak aby możliwe było wpisanie specyfikacji komputerów z klawiatury.

16.11 Unie

Podczas tworzenia rozbudowanych aplikacji na potrzeby zmiennych rezerwuje się bardzo duże ilości pamięci. Może to prowadzić do zbyt dużego zwiększenia zapotrzebowania na pamięć RAM przez aplikację. W celu ograniczenia tego zjawiska stosuje się unie i pola bitowe, które w znacznym stopniu umożliwiają zredukowanie zapotrzebowania na pamięć RAM.

Unia to postać danych, która potrafi przechować zmienne różnego typu, ale tylko jedną wartość w danej chwili.

Przykład unii o nazwie **bicykl.**

```
#include <iostream>
```

```
using namespace std;

union bicykl
{
int numer_pasazera;
char osoba;
};

int main()
{
bicykl rowerek, rowerek2;
cout << "Dwa rowery przewoza cyfry\n";
rowerek.numer_pasazera=12;
rowerek2.numer_pasazera=2;
cout << "Rowerkami jada: \n" << rowerek.numer_pasazera << "\n ";
cout << rowerek2.numer_pasazera << "\n";
cout << "Przesiadka \n";
rowerek.osoba='b';
rowerek2.osoba='a';
cout << "Teraz rowerkami jada: \n" << rowerek.osoba << "\n ";
cout << rowerek2.osoba << "\n";
cout << "Osoba b dalej jedzie pierwszym rowerkiem zatem: " <<
rowerek.numer_pasazera<< endl;
system ("pause");
}
```

Jak widać program na początku tworzy unię w ramach której potrafi przechować **numer_pasazera** o typie int, czyli liczbę całkowitą oraz **osoba** o typie char, czyli literę.

Jeśli w programie przypisujemy wartość do unii to musimy podać jeszcze element w ramach unii. Wygląda to tak:

```
rowerek.osoba='b';
```

Rowerek to element będący częścią unii **bicykl** zatem korzystamy tutaj z unii. Po kropce określamy element (tutaj **osoba)**, który ma określony typ (tutaj char). Zatem jeśli odwołujemy się do elementu unii o nazwie osoba to możemy zapisać typ char a jeśli odwołujemy się do elementu numer_pasazera to możemy zapisać liczbę całkowitą.

Na końcu naszego przykładu pokazano co się stanie jeśli po wpisaniu do unii elementu o typie char będziemy chcieli zobaczyć element liczbowy. W tym przykładzie do unii zostało wpisane „b”.

```
rowerek.osoba='b';
```

Program po wywołaniu elementu

cout << "Osoba b … zatem: " << rowerek.numer_pasazera

zwróci wartość 98 co stanowi wartość literki „b” w kodzie ASCII.

16.12 Pola bitowe

Przykład

17 Klasy

Przykład tworzenia klasy komputer dla programu analizującego wykorzystanie komputerów w dużej korporacji.

przykład

Zadanie:

Utwórz klasę postaci gry RPG. Sam wybierz jej specjalność. Pamiętaj o stworzeniu zarówno zmiennych jak i funkcji. Zastanów się, które zmienne mają pozostać prywatne, a które mogą być publiczne.

Przykład: Magik będzie miał np. funkcję rzucania czaru, której trzeba będzie przekazać jaki czar ma być rzucony. Oprócz tego trzeba będzie przechować w zmiennej jego poziom magii. Itd.

17.1 Konstruktor i destruktor

Konstruktor to specjalna funkcja składowa klasy, która ma taką samą nazwę jak nazwa klasy a jego zadaniem jest budowanie obiektu swojej klasy. Destruktor ma taką samą nazwę poprzedzoną znakiem tyldy „~”. Destruktor usuwa stworzony obiekt.

Zadanie:

W poniższym przykładzie z użyciem konstruktora tworzony jest nowy obiekt o wartościach zdefiniowanych „na sztywno” w programie. Zmodyfikuj program tak, aby pobierał dane do nowo tworzonego obiektu od użytkownika.

```
#include <iostream>
#include <string>
using namespace std;
class motocykl
{
private:
 string producent;
 int rocznik;
public:
 string model;
 motocykl();
 ~motocykl(); //deklaracja destruktora
 void wypisz();
};

motocykl::motocykl()
{
      cout << "Dotychczas nasz obiekt ma dane:\n" << "\nProducent: " <<
producent << "\nRocznik: " << rocznik << "\nModel: " << model;
   cout << "\n=======================================" << "\nTeraz
konstruktor nadaje poczatkowe wartosci danym obiektu\n";
      producent="Yamaha";
      model="YZF-R1";
```

```
      rocznik = 2008;
      cout << "Kontruktor nadal wartosci:\n";
      cout << "Producent: " << producent;
      cout << "\nRocznik  : " << rocznik;
      cout << "\nModel: " << model;
}
motocykl::~motocykl() // definicja destruktora
{
 cout <<"\n\nzniszczylem ERLANa";
}

int main()
{
 motocykl ERLAN;
 cout << "\n-------------------------";
 cout << "\n zmieniam model";
 ERLAN.model="R6";  //odwołanie się do składowej obiektu
 cout << "\n teraz model to:" << ERLAN.model;

 system("PAUSE");
 return(0);
}
```

Powyższy kod w notatniku

rozwiązanie

18 Operacje na plikach

Przykłady zastosowań:

```
==========================================
1. bezwzględność klasy ofstream - wywołanie kasuje plik
======================================
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
  int a=4;
  float b=8.5;
  ofstream wyj("ala.txt");   // (1) utworzenie obiektu wyj
  wyj << a << endl << b;  // (2) zapis do pliku
  wyj.close();    // zamknięcie pliku
  wyj.open("ala.txt");  // ponowne otwarcie pliku
  wyj.close();    // ponowne zamknięcie pliku
  wyj.open("zosia.txt");
  wyj << endl << endl << 99;
  wyj.close();
  return 0;
}



==========================================
2. możliwośc nazywania plików
======================================


#include <iostream>
#include <fstream>
using namespace std;
int main()
{
  char nazwa[50];   // tablica znaków z ograniczeniem do 50 znaków
  cout << "Podaj nazwe pliku jaki chcesz utworzyc ";
  cin >> nazwa;   // pobranie nazwy dla tworzonego pliku
  ofstream wyniki(nazwa);  // (1) utworzenie pliku
  wyniki << 's';   // do pliku wpiszemy literę s
  wyniki.close();
  return 0;
}



==========================================
3. sprawdzenie czy plik się otwarł
```

```
=======================================


#include <iostream>
#include <cstdio>
#include <fstream>
using namespace std;

int main()
{
  ofstream wyniki("out.txt");
  if (!wyniki)   // początek obsługi błędów
  {
    cout << "Pliku nie mozna otworzyc";
    getchar();   // umożliwia ci przeczytanie komunikatu błędu
    return 1;
  }    // koniec obsługi błędów
  wyniki.close();
  return 0;
}



=========================================
4. obliczenia z pliku z danymi
=======================================

#include <iostream>
#include <cstdio>
#include <fstream>
using namespace std;

int main()
{
  int a, b;
  ifstream wejscie("prostokat.txt");  // (1) powiązanie strumienia z
plikiem
  if (!wejscie)    // (2) badanie poprawnosci otwarcia
  {
    cout << "Nie mozna otworzyc pliku";
    getchar();     // (3) umożliwia przeczytanie komunikatu błędu
    return 1;
  }
  wejscie >> a >> b;   // (4) odczyt z pliku
  wejscie.close();
  cout << "Pole prostokata wynosi: " << a*b;
  getchar();
  return 0;
}


=========================================
5. sumowanie liczb z pliku
=======================================


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

```
int main()
{
  float a, suma = 0;
  char nazwa [100];
  cout << "Podaj nazwe pliku" << endl;
  cin >> nazwa;
  ifstream we(nazwa);
  if (!we)
  {
    cout << "Nie mozna otworzyc pliku";
    cin.ignore();
    getchar();
    return 1;
  }
  while (!we.eof())  // dopóki nieprawda że natrafiliśmy na koniec pliku
  {
    we >> a;  // odczyt z pliku
    if (we)        // jeśli strumień we nie jest pusty
      suma = suma+a;
  }
  we.close();
  cout << "Suma liczb w pliku " << nazwa << " wynosi: " << suma;
  cin.ignore();
  getchar();
  return 0;
}
```

Zadanie do przykładu 10.6 z podręcznika

1. Zmodyfikuj program (5. sumowanie liczb z pliku) tak, aby obliczał średnią ocen klasy z dołączonego pliku **oceny.txt**. (plik oceny.txt)

   rozwiązanie

2. Wykonaj program, który będzie się pytał użytkownika o oceny kolejnych uczniów w klasie i zapisywał je do pliku wg zasady, że oceny każdego ucznia zapisują się w osobnym wierszu i są oddzielone spacjami. Program powinien zapytać o oceny dziesięciu uczniów. Dla każdego ucznia 3 oceny.

   rozwiązanie

3. Zmodyfikuj program z punktu pierwszego aby obliczał i wyświetlał na ekranie średnią ocen dla każdego z uczniów osobno. Na końcu powinien wyświetlić średnią ocen całej klasy.

   rozwiązanie

Zadanie końcowe

4. Wykonaj program, który będzie realizował następujące funkcje wybierane przez użytkownika z menu:
   a. sumował liczby ze wskazanego pliku

b. tworzył plik z liczbami do sumowania (same liczby dodatnie, program kończy przyjmować liczby jeśli wpiszemy -1)

c. wypisywał liczby pobrane z pliku na ekranie.

19 Sprawdziany

19.1 Sprawdzian z podstaw

Sprawdzian do przeprowadzenia po zakończeniu działu Switch Case.

Sprawdzian nr 1 - podstawy

Rozwiązanie zadania nr 7

Rozwiązanie zadania nr 16

Sprawdzian poprawkowy

Rozwiązania zadań ze sprawdzianu poprawkowego

Rozwiązanie_zad.1

Rozwiązanie_zad.2

19.2 Sprawdzian z tablic i funkcji

Sprawdzian po dziale Tablice

Zadania

(UWAGA! Rozwiązania są do zadań z roku 2007, które minimalnie się różnią od zadań z 2008r.)

Rozwiązania

Rozwiązania cpp

Zadania na sprawdzian poprawkowy

19.3 Sprawdzian z teorii na koniec semestru (test)

Test na koniec semestru

Rozwiązania niektórych zadań z testu

Tablica dwunastoelementowa

Test poprawkowy

Test poprawkowy nr 2

19.4 Sprawdzian z pojęć programowania obiektowego

Sprawdzian z zakresu pojęć struktury, unie, pola bitowe, klasy i obiekty, definiowanie klas, składniki klasy, hermetyzacja konstruktor i destruktor, przeciążenie funkcji, przeciążenie operatorów, zapis i odczyt pliku, dziedziczenie, polimorfizm.

Sprawdzian

Sprawdzian – rozwiązanie

Sprawdzian poprawkowy

Sprawdzian poprawkowy rozwiązanie

Sprawdzian praktyczny

Sprawdzian praktyczny rozwiązanie