**Zadanie 1.**

W przeszłości telefonu komórkowe posiadały funkcję kompozytora, która pozwala na wprowadzanie własnych melodii (dzwonków) za pomocą klawiatury telefonu. Napisz własny program, który odtworzy zakodowaną melodię za pomocą PC speaker. Przyjmij, że kolejne nuty melodii w kodzie będą oddzielone spacjami, a każda z nut będzie opisana wyrażeniem *WDO*, gdzie:

* W – kod długości trwania nuty: 1 oznacza całą nutę, 2 − półnutę, 4 − ćwierćnutę, 8 − ósemkę (uwaga: cyfra reprezentująca długość faktycznie jest dzielnikiem czasu trwania całej nuty);
* D – nazwa dźwięku z szeregu chromatycznego (nazwy i odpowiadające im częstotliwości dla oktawy razkreślnej podano w tabeli 1);
* O – oktawa tzn. 1 = razkreślna, 2 = dwukreślna, 3 = trzykreślna (podpowiedź: cyfra określająca oktawę to jednocześnie mnożnik częstotliwości).

W programie uwzględnij również kodowanie pauzy (przerwy w brzmieniu) za pomocą nazwy dodatkowego „głuchego” dźwięku P. Do uproszczenia zadania przyjmij czas trwania całej nuty równy 2 sekundy. Oto kod jego pierwszej melodii: 8C1, 8D1, 8E1, 8F1, 4G1, 4E1, 4G1, 4E1, 8E1, 8E1, 8D1, 8D1, 2C1, 8C1, 8D1, 8E1, 8F1, 4G1, 4E1, 4G1, 4E1, 8E1, 8E1, 8D1, 8D1, 2C1.

Tabela 1

|  |  |
| --- | --- |
| **Dźwięk** | **Częstotliwość, Hz** |
| C | 261,6 |
| D | 293,7 |
| E | 329,6 |
| F | 349,2 |
| G | 391,9 |
| A | 440,0 |
| H | 493,9 |

**Rozwiązanie 1**

W pętli należy odegrać każdy kolejny dźwięk. Częstotliwość dźwięku należy zdekodować na podstawie tabeli 1 i pomnożyć razy oktawę. Czas trwania dźwięku należy wyliczyć poprzez podzielenie czasu trwania całej nuty (2 sekundy) przez wspomniany w treści zadania dzielnik czasu trwania *W*. Na potrzeby funkcji Beep() czas trwania dźwięku należy przeliczyć na milisekundy.

Listing 1 zawiera przykładowy kod napisany w C++, który odgrywa powyższą melodię.

Listing 1

#include "stdafx.h"

#include <iostream>

using namespace std;

#include <windows.h>

float Czestotliwosc(char dzwiek, int oktawa);

int main()

{

char DzwiekNuty;

int Wartosc, Oktawa;

float DlugoscCalejNuty = 2000;

std:string Melodia[30] = {"8C1", "8D1", "8E1", "8F1", "4G1", "4E1", "4G1", "4E1", "8E1", "8E1", "8D1", "8D1", "2C1", "8C1", "8D1", "8E1", "8F1", "4G1", "4E1", "4G1", "4E1", "8E1", "8E1", "8D1", "8D1", "2C1"};

for (int i = 0; i < 30; i++)

{

Wartosc = Melodia[i][0] - '0'; DzwiekNuty = Melodia[i][1]; Oktawa = Melodia[i][2] - '0';

Beep(Czestotliwosc(DzwiekNuty, Oktawa), DlugoscCalejNuty / Wartosc);

}

return 0;

}

float Czestotliwosc(char dzwiek, int oktawa)

{

float czestotliwoscBazowa = 0;

switch (dzwiek)

{

case 'C':

czestotliwoscBazowa = 261.6;

break;

case 'D':

czestotliwoscBazowa = 293.7;

break;

case 'E':

czestotliwoscBazowa = 329.6;

break;

case 'F':

czestotliwoscBazowa = 349.2;

break;

case 'G':

czestotliwoscBazowa = 391.9;

break;

case 'A':

czestotliwoscBazowa = 440;

break;

case 'H':

czestotliwoscBazowa = 493.9;

break;

case 'P':

czestotliwoscBazowa = 0;

break;

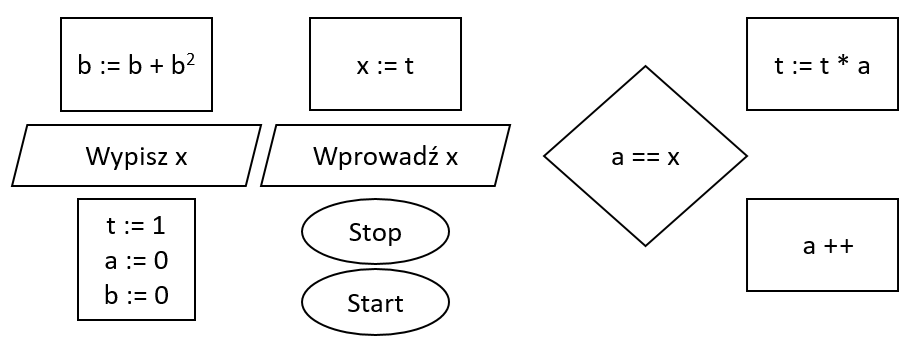
}

return czestotliwoscBazowa \* oktawa;

}

**Zadanie 2**

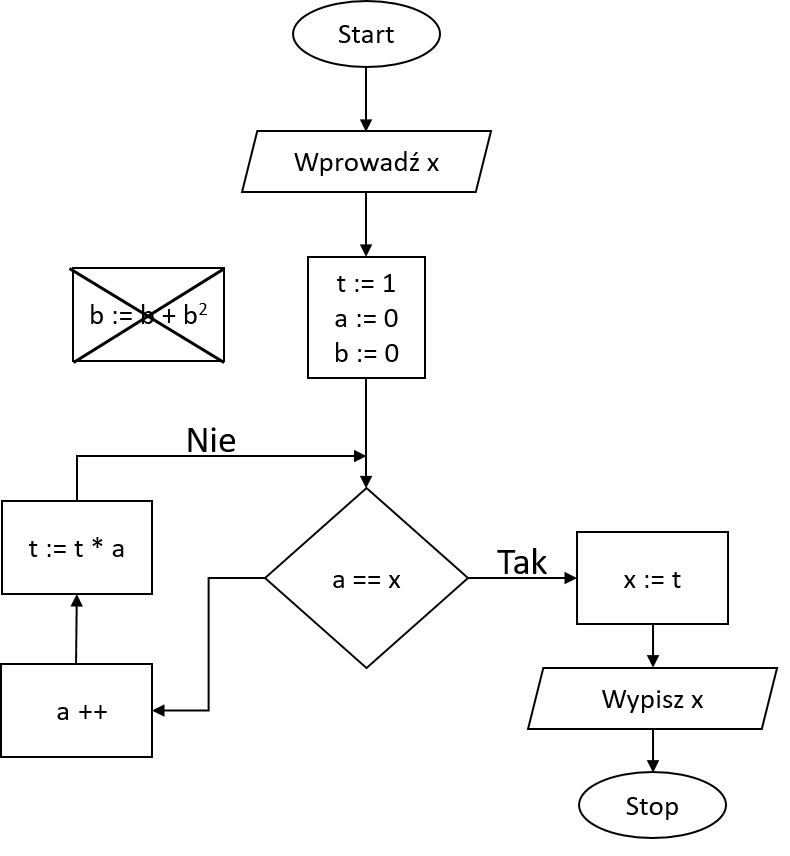
Ułóż prawidłowy schemat blokowy z elementów z rysunku 1. Jeden z elementów jest zbędny, wskaż który to element. Podpowiedź: działanie matematyczne reprezentowane przez algorytm z tego schematu składa się z jednej lub kilku liczb (nie więcej niż x), a wartość poszczególnych liczb jest mniejsza lub równa x.



Rys. 1 Bloki do wykorzystania

**Rozwiązanie 2**

Prawidłowo ułożony schemat to algorytm silni, tzn. n!. Zbędny element to b:= b + b2, który nie powinien brać udziału w schemacie blokowym. Rysunek 2 przedstawia prawidłowe rozwiązanie.



Rys. 2 Prawidłowo ułożony schemat blokowy

**Zadanie 3**

Na listingu 2 przedstawiono kod źródłowy znanego algorytmu stosowanego w matematyce. Niestety, gdzieś wkradł się błąd i w wyniku działania programu uzyskiwany jest zły wynik. Prawidłowe wyniki dla wybranych parametrów wejściowych przedstawia tabela 2. Popraw błąd w kodzie programu oraz określ jaki to algorytm.

Tabela 2

|  |  |  |  |
| --- | --- | --- | --- |
| p | q | k | Wynik |
| 1 | 2 | 6 | 63 |
| 3 | 2 | 7 | 381 |
| 6 | 3 | 5 | 726 |

Listing 2

#include "stdafx.h"

#include <iostream>

using namespace std;

int main() {

double p, q, w; int k;

cout << "Podaj p:" << endl;

cin >> p;

cout << "Podaj q:" << endl;

cin >> q;

cout << "Podaj k:" << endl;

cin >> k;

if (q == 1)

w = p \* k;

w = p \* ((1 - atan2l(q, k)) / (1 - q));

cout << "wynik: " << w << endl;

system("pause");

return 0; }

**Rozwiązanie 3**

Algorytm to suma k wyrazów ciągu geometrycznego, gdzie p to pierwszy element ciągu, q to iloraz ciągu oraz k-ty wyraz ciągu. Dla ułatwienia wzór na sumę k-tego wyrazu ciągu został zapisany w kodzie o postaci bliskiej wzorowi w matematyce, wystarczy funkcję arcus tanges zastąpić funkcją potęgi, co przedstawia listing 3.

Listing 3

#include "stdafx.h"

#include <iostream>

using namespace std;

int main() {

double p, q, w; int k;

cout << "Podaj p:" << endl;

cin >> p;

cout << "Podaj q:" << endl;

cin >> q;

cout << "Podaj k:" << endl;

cin >> k;

if (q == 1)

w = p \* k;

w = p \* ((1 - **pow**(q, k)) / (1 - q));

cout << "wynik: " << w << endl;

system("pause");

return 0; }

**Zadanie 4**

Unipolarne silniki krokowe wykorzystywane są często w drukarkach atramentowych. Przykładowy silnik tego typu ma 5 wyprowadzeń, z czego środkowe zawsze powinno być podłączone do logicznego zera (GND), natomiast logiczna jedynka (VCC) na jednym z pozostałych wyprowadzeń powoduje wykonanie jednego kroku przez silnik.

Załóż, że silnik sterowany jest przez rejestr 8-bitowego mikrokontrolera, a wyprowadzenia silnika są podłączone do najmniej znaczących bitów tego rejestru (drugi bit to podłączone na stałe logiczne zero).

Następująca sekwencja zapisana kolejno do rejestru spowoduje obrót osi silnika w lewą stronę (przeciwnie do wskazówek zegara): XXX00Z01, XXX00Z10, XXX01Z00, XXX10Z00, XXX00Z01 itd., gdzie Z to w rzeczywistości wartość 0 (w przykładzie zastąpione literą Z aby wskazać bit, który zawsze będzie logicznym zerem), X to bit bez znaczenia.

Napisz program, który rozpoczynając od stanu spoczynku (XXX00Z00 według powyższego zapisu) wykona następujący ruch osią silnika: 3 kroki w lewo, 2 kroki w prawo, 5 kroków w lewo, 9 kroków w prawo. Kolejne wartości rejestru wypisz od nowego wiersza na konsolę w postaci binarnej. Pamiętaj, że zapisanie wartości niezerowej do rejestru powoduje ruch osi silnika.

**Rozwiązanie 4**

Sterowanie silnikiem krokowym można zrealizować poprzez przesunięcia bitowe. Należy zabezpieczyć program przed zapisaniem logicznej jedynki do bitu b2 (zawsze powinno być zero) oraz przed przesunięciem jedynki poza bity sterujące. Po wystąpieniu jednego z dwóch wymienionych przypadków należy ustawić jedynkę na odpowiednim bicie.

Listing 4 zaweira przykładowy kod, zapisanie wartości niezerowej do zmiennej Rejestr powoduje wykonanie obrotu osi silnika. Na rysunku 3 przestawiony jest wynik z konsoli.



Rys. 3. Wynik wypisany w konsoli, przedstawiający przesunięcia w rejestrze.

Listing 4

#include "stdafx.h"

#include <iostream>

#include <bitset>

using namespace std;

unsigned char Rejestr;

int \_tmain(int argc, \_TCHAR\* argv[])

{

Rejestr = 0;

WykonajKrok(3, 0);

WykonajKrok(2, 1);

WykonajKrok(5, 0);

WykonajKrok(9, 1);

std::getchar();

return 0;

}

/// Kierunek - 0 w lewo, 1 w prawo

void WykonajKrok(int iloscKrokow, bool kierunek)

{

for (int i = 0; i < iloscKrokow; i++)

{

unsigned char kopiaRejestru = ejestr; // potrzebna jest kopia rejestru, ponieważ zapisanie do niego niezerowej wartości powoduje ruch osi silnika

kierunek ? kopiaRejestru >>= 1 : kopiaRejestru <<= 1; // przesuń kopię rejestru

if (kopiaRejestru & 0x4) // wykrywanie czy jedynka nie ustawiła się na b2 (który ma być zawsze zerowy)

kierunek ? kopiaRejestru >>= 1 : kopiaRejestru <<= 1; // jeśli tak po przesuń jedynkę z b2 w odpowiednią stronę

if ((kopiaRejestru >= 0x20) || (kopiaRejestru == 0)) // wykrywanie czy jedynka nie wyszła poza bity sterujące

{

if (kierunek)

Rejestr = 16; // ustawianie jedynki na b4

else

Rejestr = 1; // ustawianie jedynki b0

}

else

{

Rejestr = kopiaRejestru; // przypisanie wartości wynikającej z przesunięcia

}

std::cout << std::bitset<8>(Rejestr); // wypisanie na konsolę w postaci binarnej

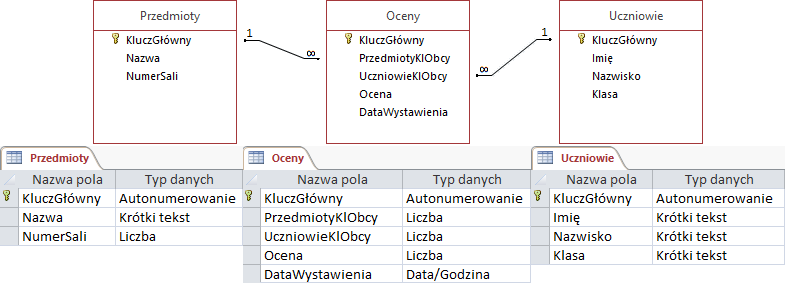
cout << endl;

}

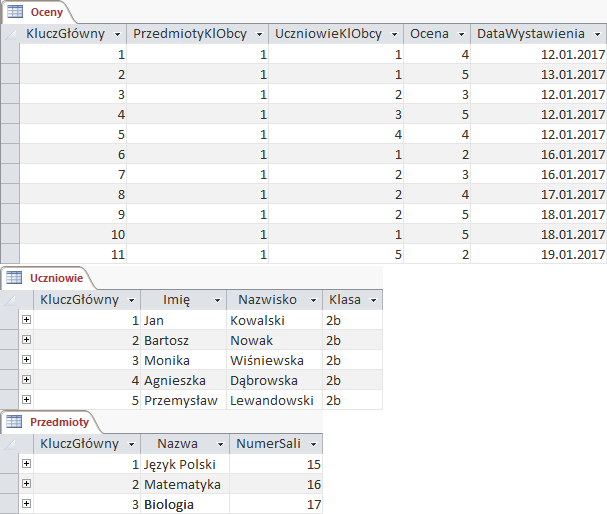
}

**Zadanie 5**

Elektroniczny dziennik szkolny można stworzyć wykorzystując Microsoft Access. Stwórz kwerendę, która wyświetli średnie ocen dla wszystkich uczniów z przedmiotu Język Polski. Wykorzystaj strukturę danych przedstawioną na rysunku 4. Dane uzupełnij zgodnie z rysunkiem 5. Wynik kwerendy posortuj według nazwiska ucznia. Postaraj się wykorzystać język SQL.



Rys. 4. Struktura elektronicznego dziennika



Rys. 5 Dane z elektronicznego dziennika

**Rozwiązanie 5**

Rozwiązanie polega na utworzeniu bazy danych zgodnie z przedstawionymi rysunkami, a następnie przygotowaniu kwerendy w języku SQL (patrz rys. 6 i listing 5), która powinna zawierać klauzule *select, from, inner join, where, group by, order by.*



Rys 6. Wynik kwerendy uruchomionej w bazie danych elektronicznego dziennika

Listing 5

-- Zapytanie w środowisku Microsoft Access

SELECT Uczniowie.Imię, Uczniowie.Nazwisko, AVG(Oceny.Ocena) AS Średnia

FROM Uczniowie INNER JOIN (Przedmioty INNER JOIN Oceny ON Przedmioty.KluczGłówny=Oceny.[PrzedmiotyKlObcy]) ON Uczniowie.KluczGłówny=Oceny.[UczniowieKlObcy]

WHERE Przedmioty.Nazwa = "Język Polski"

GROUP BY Uczniowie.Imię, Uczniowie.Nazwisko

order by Uczniowie.Nazwisko