Organizer

1.0

Wygenerowano przez Doxygen 1.9.5

Rozdział 1

Indeks klas

1.1 Lista klas

Tutaj znajdują się klasy, struktury, unie i interfejsy wraz z ich krótkimi opisami:

Rozdział 2

Indeks plików

2.1 Lista plików

Tutaj znajduje się lista wszystkich plików z ich krótkimi opisami:

Rozdział 3

Dokumentacja klas

3.1 Dokumentacja struktury kontakt

Struktura kontaktu. Struktura ta składa się z 3 ciągów znaków: imienia, nazwiska i e-maila danego kontaktu.

Atrybuty publiczne

- str imie
- str nazwisko
- str email

3.1.1 Opis szczegółowy

Struktura kontaktu. Struktura ta składa się z 3 ciągów znaków: imienia, nazwiska i e-maila danego kontaktu.

Definicja w linii 50 pliku organizer.cpp.

3.1.2 Dokumentacja atrybutów składowych

3.1.2.1 email

```
str kontakt::email
```

Definicja w linii 54 pliku organizer.cpp.

3.1.2.2 imie

```
str kontakt::imie
```

Definicja w linii 52 pliku organizer.cpp.

3.1.2.3 nazwisko

```
str kontakt::nazwisko
```

Definicja w linii 53 pliku organizer.cpp.

Dokumentacja dla tej struktury została wygenerowana z pliku:

- C:/Users/Kamil/Desktop/Studia/PPK/Projekt_PPK/Organizer/organizer.cpp

3.2 Dokumentacja struktury zadanie

Struktura zadania. Struktura ta składa się z 5 ciągów znaków: daty, godziny, typu, statusu i opisu danego zadania.

Atrybuty publiczne

- str data
- str godzina
- str typ
- str status
- str opis

3.2.1 Opis szczegółowy

Struktura zadania. Struktura ta składa się z 5 ciągów znaków: daty, godziny, typu, statusu i opisu danego zadania.

Definicja w linii 62 pliku organizer.cpp.

3.2.2 Dokumentacja atrybutów składowych

3.2.2.1 data

```
str zadanie::data
```

Definicja w linii 64 pliku organizer.cpp.

3.2.2.2 godzina

```
str zadanie::godzina
```

Definicja w linii 65 pliku organizer.cpp.

3.2.2.3 opis

```
str zadanie::opis
```

Definicja w linii 68 pliku organizer.cpp.

3.2.2.4 status

```
str zadanie::status
```

Definicja w linii 67 pliku organizer.cpp.

3.2.2.5 typ

```
str zadanie::typ
```

Definicja w linii 66 pliku organizer.cpp.

Dokumentacja dla tej struktury została wygenerowana z pliku:

- C:/Users/Kamil/Desktop/Studia/PPK/Projekt_PPK/Organizer/organizer.cpp

Rozdział 4

Dokumentacja plików

4.1 Dokumentacja pliku C:/Users/Kamil/Desktop/Studia/PPK/Projekt_↩ PPK/Organizer/organizer.cpp

Aplikacja do zarządzania kontaktami i zadaniami.

```
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>
#include <cmath>
#include <fstream>
#include <cctype>
#include <regex>
```

Komponenty

- struct kontakt

  Struktura kontaktu. Struktura ta składa się z 3 ciągów znaków: imienia, nazwiska i e-maila danego kontaktu.
- struct zadanie

  Struktura zadania. Struktura ta składa się z 5 ciągów znaków: daty, godziny, typu, statusu i opisu danego zadania.

Definicje typów

- using str = string

  Definicja skrótu str dla typu danych string. Definicja skrótu dla wygody i czytelności.

Funkcje

- void WypiszVectorK (vector< kontakt > &vec)

  Funkcja wypisująca vector kontaktów.
- void WypiszVectorZ (vector< zadanie > &vec)

  Funkcja wypisująca vector zadań.
- vector< kontakt > sortujVecK (vector< kontakt > vec, int x)

  Funkcja sortująca vector kontaktów. Funkcje sortowania sortują alfabetycznie.
- vector< zadanie > sortujVecZ (vector< zadanie > vec, int x)

  Funkcja sortująca vector zadań. Funkcje sortowania sortują alfabetycznie.
- vector< kontakt > filtrujVecK (vector< kontakt > kontakty, int x)

  Funkcja filtrująca vector kontaktów.
- vector< zadanie > filtrujVecZ (vector< zadanie > zadania, int x)

  Funkcja filtrująca vector zadań.
- void DodajKontakt (vector< kontakt > &vec)

  Funkcja dodająca kontakt do vectora kontaktów. Funkcja ta przyjmuje od użytkownika wszystkie dane nowego kontaktu i zapisuje go do vectora kontaktów.
- void DodajZadanie (vector< zadanie > &vec)

  Funkcja dodająca zadanie do vectora zadań. Funkcja ta przyjmuje od użytkownika wszystkie dane nowego zadania i zapisuje go do vectora zadań.
- void UsunKontakt (vector< kontakt > &vec)

  Funkcja usuwająca kontakt z vectora kontaktów. Funkcja ta usuwa wybrany przez użytkownika kontakt z vectora kontaktów.
- void UsunZadanie (vector< zadanie > &vec)

  Funkcja usuwająca zadanie z vectora zadań. Funkcja ta usuwa wybrane przez użytkownika zadanie z vectora zadań.
- void ZmienStatus (vector< zadanie > &vec)

  Funkcja zmieniająca status zadania. Funkcja ta zmienia status zadania z wykonanego na niewykonane lub na odwrót (V -> X lub X -> V).
- vector< kontakt > CzytajKontakty (str nazwa_pliku_kontakty)

  Funkcja czytająca kontakty z pliku tekstowego. Funkcja ta czyta kontakty zapisane w pliku tekstowym i zapisuje je w vectorze kontaktów.
- vector< zadanie > CzytajZadania (str nazwa_pliku_zadania)

  Funkcja czytająca zadania z pliku tekstowego. Funkcja ta czyta zadania zapisane w pliku tekstowym i zapisuje je w vectorze zadań.
- void ZapiszWPlikach (str nazwa_pliku_kontakty, str nazwa_pliku_zadania, vector< kontakt > &vecK, vector< zadanie > &vecZ)

  Funkcja zapisująca kontakty i zadania w plikach tekstowych. Funkcja ta zapisuje kontakty i zadania z odpowiadających sobie vectorów do pliku tekstowego. Funkcja używa tych samych plików tekstowych co funkcje czytające dane z plików przy starcie programu.
- void Zapytanie ()

  Funkcja pytająca użytkownika o chęć kontynuacji działania z programem. Funkcja ta pyta użytkownika czy chce on powrócić do menu głównego (1) czy też zakończyć działanie z programem (2).
- int WedlugCzegoK (string flag)

  Funkcja pytająca użytkownika według jakiej właściwości kontaktu posortować lub pofiltrować kontakty. Funkcja ta pyta użytkownika według jakiej właściwości kontaktu chce sortować lub filtrować kontakty i zwraca jego decyzję w postaci liczby całkowitej.
- int WedlugCzegoZ (string flag)

  Funkcja pytająca użytkownika według jakiej właściwości zadania posortować lub pofiltrować zadania. Funkcja ta pyta użytkownika według jakiej właściwości zadania chce sortować lub filtrować zadania i zwraca jego decyzję w postaci liczby całkowitej.
- bool CzySortowac ()

  Funkcja pytająca użytkownika czy sortować kontakty lub zadania. Funkcja ta pyta użytkownika czy chce sortować kontakty lub zadania i zwraca prawdę lub fałsz w zależności od jego decyzji.
- bool CzyFiltrowac ()

Funkcja pytająca użytkownika czy filtrować kontakty lub zadania. Funkcja ta pyta użytkownika czy chce filtrować kontakty lub zadania i zwraca prawdę lub fałsz w zależności od jego decyzji.

- bool TylkoLitery (string nazwa)

  Funkcja sprawdzająca czy dany ciąg znaków zawiera tylko litery. Funkcja ta przyjmuje ciąg znaków i sprawdza czy zawiera on tylko znaki alfanumeryczne za pomocą wyrażenia regularnego po czym zwraca prawdę lub fałsz w zależności od wyniku.
- bool CzyEmail (string email)

  Funkcja sprawdzająca czy dany ciąg znaków jest adresem e-mail. Funkcja ta przyjmuje ciąg znaków i sprawdza czy jest on prawidłowym e-mailem za pomocą wyrażenia regularnego po czym zwraca prawdę lub fałsz w zależności od wyniku.
- bool CzyData (string data)

  Funkcja sprawdzająca czy dany ciąg znaków jest datą w foramcie RRRR.MM.DD. Funkcja ta przyjmuje ciąg znaków i sprawdza czy jest on prawidłową datą w formacie RRRR.MM.DD za pomocą wyrażenia regularnego po czym zwraca prawdę lub fałsz w zależności od wyniku.
- bool CzyGodzina (string godzina)

  Funkcja sprawdzająca czy dany ciąg znaków jest godziną w formacie GG:MM. Funkcja ta przyjmuje ciąg znaków i sprawdza czy jest on prawidłową godziną w formacie GG:MM za pomocą wyrażenia regularnego po czym zwraca prawdę lub fałsz w zależności od wyniku.
- void Start (vector< kontakt > &kontakty, vector< zadanie > &zadania)

  Funkcja uruchamiająca główne menu programu i wykonująca podprogramy. Funkcja ta przyjmuje vector kontaktów oraz vector zadań i za ich pomocą uruchamia odpowiedni podprogram wybrany przez użytkownika.
- int main (int argc, char *args[])

  Funkcja main. Funkcja ta jest funkcją główną. Wykonuje ona ustawienia wstępne, uruchamia funkcję Start() oraz zajmuje się wywołaniem funkcji ZapiszWPlikach().

Zmienne

- bool DzialanieProgramu = true

  Na podstawie wartości tej zmiennej program kończy swoje działanie. Gdy zmienna ma wartość true program kontynuuje swoje działanie, a gdy false zostaje zakończony.
- bool show = true

  Na podstawie tej zmiennej wyświetlane jest podsumowanie. Gdy zmienna ma wartość true zostanie pokazane podsumowanie, a gdy false nie.

4.1.1 Opis szczegółowy

Aplikacja do zarządzania kontaktami i zadaniami.

Autor

Kamil Nol

Wersja

1.0

Data

2022-12-30

Copyright

Definicja w pliku organizer.cpp.

4.1.2 Dokumentacja definicji typów

4.1.2.1 str

```
using str = string
```

Definicja skrótu str dla typu danych string. Definicja skrótu dla wygody i czytelności.

Definicja w linii 31 pliku organizer.cpp.

4.1.3 Dokumentacja funkcji

4.1.3.1 CzyData()

```
bool CzyData (
            string data )
```

Funkcja sprawdzająca czy dany ciąg znaków jest datą w foramcie RRRR.MM.DD. Funkcja ta przyjmuje ciąg znaków i sprawdza czy jest on prawidłową datą w formacie RRRR.MM.DD za pomocą wyrażenia regularnego po czym zwraca prawdę lub fałsz w zależności od wyniku.

Parametry

data	Sprawdzany ciąg znaków.

Zwraca

true Wartość zawracana gdy okaże się że podany ciąg znaków jest prawidłową datą.

false Wartość zawracana gdy okaże się że podany ciąg znaków nie jest prawidłową datą.

Definicja w linii 1257 pliku organizer.cpp.

```
01258 {
01259     regex wzorzec("[0-9]{4}.[0-9]{2}.[0-9]{2}");
01260     if (regex_match(data, wzorzec))
01261     {
01262         if (data[5] == '0' && data[6] == '0')
01263             return false;
01264         if (data[8] == '0' && data[9] == '0')
01265             return false;
01266         return true;
01267     }
01268     else
01269         return false;
01270 }
```

4.1.3.2 CzyEmail()

```
bool CzyEmail (
            string email )
```

Funkcja sprawdzająca czy dany ciąg znaków jest adresem e-mail. Funkcja ta przyjmuje ciąg znaków i sprawdza czy jest on prawidłowym e-mailem za pomocą wyrażenia regularnego po czym zwraca prawdę lub fałsz w zależności od wyniku.

Parametry

email	Sprawdzany ciąg znaków.

Zwraca

true Wartość zawracana gdy okaże się że podany ciąg znaków jest prawidłowym e-mailem.

false Wartość zawracana gdy okaże się że podany ciąg znaków nie jest prawidłowym e-mailem.

Definicja w linii 1248 pliku organizer.cpp.

```
01249 {
01250     regex wzorzec("[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\\.[a-zA-Z]{2,}");
01251     if (regex_match(email, wzorzec))
01252         return true;
01253     else
01254         return false;
01255 }
```

4.1.3.3 CzyFiltrowac()

```
bool CzyFiltrowac ( )
```

Funkcja pytająca użytkownika czy filtrować kontakty lub zadania. Funkcja ta pyta użytkownika czy chce filtrować kontakty lub zadania i zwraca prawdę lub fałsz w zależności od jego decyzji.

Zwraca

true Wartość zwracana gdy użytkownik chce filtrować.

false Wartość zwracana gdy użytkownik nie chce filtrować.

Definicja w linii 1059 pliku organizer.cpp.

```
01060 {
01061     string wejscie;
01062     char x;
01063     do
01064     {
01065         do
01066         {
01067             cout « "Czy filtrować? T/N" « endl;
01068             cin » wejscie;
01069             if (cin.fail() || wejscie.size() > 1)
01070                 cout « "Błąd konwersji. Wejście najprawdopodobniej nie jest znakiem. Spróbuj
    ponownie." « endl;
01071         } while (cin.fail() || wejscie.size() > 1);
01072         x = wejscie[0];
01073         if (x != 'T' && x != 'N' && x != 't' && x != 'n')
01074             cout « "Nie ma takiej opcji. Spróbuj ponownie." « endl;
01075     } while (x != 'T' && x != 'N' && x != 't' && x != 'n');
01076
01077     if (x == 'T' || x == 't')
01078         return true;
01079     else
01080         return false;
01081 }
```

4.1.3.4 CzyGodzina()

```
bool CzyGodzina (
            string godzina )
```

Funkcja sprawdzająca czy dany ciąg znaków jest godziną w formacie GG:MM. Funkcja ta przyjmuje ciąg znaków i sprawdza czy jest on prawidłową godziną w formacie GG:MM za pomocą wyrażenia regularnego po czym zwraca prawdę lub fałsz w zależności od wyniku.

Parametry

godzina	Sprawdzany ciąg znaków.

Zwraca

true Wartość zawracana gdy okaże się że podany ciąg znaków jest prawidłową godziną.

false Wartość zawracana gdy okaże się że podany ciąg znaków nie jest prawidłową godziną.

Definicja w linii 1272 pliku organizer.cpp.

```
01273 {
01274     regex wzorzec("[0-2]+[0-9]:[0-5]+[0-9]");
01275     if (regex_match(godzina, wzorzec))
01276     {
01277         regex wzorzec1("[4-9]");
01278         string s(1, godzina[1]);
01279         if (godzina[0] == '2' && regex_match(s, wzorzec1))
01280             return false;
01281         return true;
01282     }
01283     else
01284         return false;
01285 }
```

4.1.3.5 CzySortowac()

```
bool CzySortowac ( )
```

Funkcja pytająca użytkownika czy sortować kontakty lub zadania. Funkcja ta pyta użytkownika czy chce sortować kontakty lub zadania i zwraca prawdę lub fałsz w zależności od jego decyzji.

Zwraca

true Wartość zwracana gdy użytkownik chce sortować.

false Wartość zwracana gdy użytkownik nie chce sortować.

Definicja w linii 1034 pliku organizer.cpp.

```
01035 {
01036     string wejscie;
01037     char x;
01038     do
01039     {
01040         do
01041         {
01042             cout « "Czy sortować? T/N" « endl;
01043             cin » wejscie;
01044             if (cin.fail() || wejscie.size() > 1)
01045                 cout « "Błąd konwersji. Wejście najprawdopodobniej nie jest znakiem. Spróbuj
      ponownie." « endl;
01046         } while (cin.fail() || wejscie.size() > 1);
```

```
01047         x = wejscie[0];
01048         if (x != 'T' && x != 'N' && x != 't' && x != 'n')
01049             cout « "Nie ma takiej opcji. Spróbuj ponownie." « endl;
01050
01051     } while (x != 'T' && x != 'N' && x != 't' && x != 'n');
01052
01053     if (x == 'T' || x == 't')
01054         return true;
01055     else
01056         return false;
01057 }
```

4.1.3.6 CzytajKontakty()

```
vector< kontakt > CzytajKontakty (
             str nazwa_pliku_kontakty )
```

Funkcja czytająca kontakty z pliku tekstowego. Funkcja ta czyta kontakty zapisane w pliku tekstowym i zapisuje je w vectorze kontaktów.

Parametry

nazwa_pliku_kontakty	Nazwa pliku z którego mają zostać sczytane kontakty.

Zwraca

vector<kontakt> Vector kontaktów przechowujący sczytane kontakty.

Definicja w linii 900 pliku organizer.cpp.

```
00901 {
00902     fstream wejscie(nazwa_pliku_kontakty);
00903
00904     vector<kontakt> kontakty;
00905
00906     if (wejscie.good())
00907     {
00908         str imie, nazwisko, email;
00909
00910         while (wejscie » imie » nazwisko » email)
00911         {
00912             kontakt osoba;
00913             osoba.imie = imie;
00914             osoba.nazwisko = nazwisko;
00915             osoba.email = email;
00916             kontakty.push_back(osoba);
00917         }
00918     }
00919     wejscie.close();
00920     return kontakty;
00921 }
```

4.1.3.7 CzytajZadania()

```
vector< zadanie > CzytajZadania (
             str nazwa_pliku_zadania )
```

Funkcja czytająca zadania z pliku tekstowego. Funkcja ta czyta zadania zapisane w pliku tekstowym i zapisuje je w vectorze zadań.

Parametry

nazwa_pliku_zadania	Nazwa pliku z którego mają zostać sczytane zadania.

Zwraca

vector<zadanie> Vector zadań przechowujący sczytane zadania.

Definicja w linii 923 pliku organizer.cpp.

```
00924 {
00925     fstream wejscie(nazwa_pliku_zadania);
00926
00927     vector<zadanie> zadania;
00928
00929     str data, godzina, typ, status, opis;
00930
00931     if (wejscie.good())
00932     {
00933         while (wejscie » data » godzina » typ » status)
00934         {
00935             getline(wejscie, opis);
00936             opis.erase(0, 1);
00937             zadanie zad;
00938             zad.data = data;
00939             zad.godzina = godzina;
00940             zad.typ = typ;
00941             zad.status = status;
00942             zad.opis = opis;
00943             zadania.push_back(zad);
00944         }
00945     }
00946
00947     wejscie.close();
00948     return zadania;
00949 }
```

4.1.3.8 DodajKontakt()

```
void DodajKontakt (
            vector< kontakt > & vec )
```

Funkcja dodająca kontakt do vectora kontaktów. Funkcja ta przyjmuje od użytkownika wszystkie dane nowego kontaktu i zapisuje go do vectora kontaktów.

Parametry

vec	Referencja do vectora kontaktów do którego zostanie dodany nowy kontakt.

Definicja w linii 661 pliku organizer.cpp.

```
00662 {
00663     str imie, nazwisko, email;
00664     do
00665     {
00666         cout « "Podaj imię: A-anuluj" « endl;
00667         cin » imie;
00668         if (imie == "A"){
00669             show = false;
00670             return;
00671         }
00672         if (!TylkoLitery(imie))
00673             cout « "Błędne imie. Spróbuj ponownie." « endl;
00674     } while (!TylkoLitery(imie));
00675     do
00676     {
```

```
00677         cout « "Podaj nazwisko: A-anuluj" « endl;
00678         cin » nazwisko;
00679         if (nazwisko == "A"){
00680             show = false;
00681             return;
00682         }
00683         if (!TylkoLitery(nazwisko))
00684             cout « "Błędne nazwisko. Spróbuj ponownie." « endl;
00685     } while (!TylkoLitery(nazwisko));
00686     do
00687     {
00688         cout « "Podaj email: A-anuluj" « endl;
00689         cin » email;
00690         if (email == "A"){
00691             show = false;
00692             return;
00693         }
00694         if (!CzyEmail(email))
00695             cout « "Błędny E-Mail. Spróbuj ponownie." « endl;
00696     } while (!CzyEmail(email));
00697
00698     kontakt NowaOsoba;
00699     NowaOsoba.imie = imie;
00700     NowaOsoba.nazwisko = nazwisko;
00701     NowaOsoba.email = email;
00702
00703     vec.push_back(NowaOsoba);
00704 }
```

4.1.3.9 DodajZadanie()

```
void DodajZadanie (
            vector< zadanie > & vec )
```

Funkcja dodająca zadanie do vectora zadań. Funkcja ta przyjmuje od użytkownika wszystkie dane nowego zadania i zapisuje go do vectora zadań.

Parametry

vec	Referencja do vectora zadań do którego zostanie dodane nowe zadanie.

Definicja w linii 744 pliku organizer.cpp.

```
00745 {
00746     str data, godzina, typ, status, opis;
00747     do
00748     {
00749         cout « "Podaj datę (RRRR.MM.DD): A-anuluj" « endl;
00750         cin » data;
00751         if (data == "A"){
00752             show = false;
00753             return;
00754         }
00755         if (CzyData(data) == false)
00756             cout « "Błędna data. Spróbuj ponownie." « endl;
00757     } while (!CzyData(data));
00758     do
00759     {
00760         cout « "Podaj godzinę (GG:MM): A-anuluj" « endl;
00761         cin » godzina;
00762         if (godzina == "A"){
00763             show = false;
00764             return;
00765         }
00766         if (CzyGodzina(godzina) == false)
00767             cout « "Błędna godzina. Spróbuj ponownie." « endl;
00768     } while (!CzyGodzina(godzina));
00769     do
00770     {
00771         cout « "Podaj typ: A-anuluj" « endl;
00772         cin » typ;
00773         if (typ == "A"){
00774             show = false;
```

```
00775             return;
00776         }
00777         if (TylkoLitery(typ) == false)
00778             cout « "Błędny typ. Spróbuj ponownie." « endl;
00779     } while (!TylkoLitery(typ));
00780     do
00781     {
00782         cout « "Podaj status(X - niewykonane, V - wykonane): A-anuluj" « endl;
00783         cin » status;
00784         if (status == "A"){
00785             show = false;
00786             return;
00787         }
00788         if (status != "X" && status != "V" && status != "x" && status != "v")
00789             cout « "Błędny status. Spróbuj ponownie." « endl;
00790     } while (status != "X" && status != "V" && status != "x" && status != "v");
00791     if (status == "x")
00792         status = "X";
00793     else if (status == "v")
00794         status = "V";
00795     cout « "Podaj opis: A-anuluj" « endl;
00796     cin.clear();
00797     cin.sync();
00798     getline(cin, opis);
00799
00800     if (opis == "A"){
00801             show = false;
00802             return;
00803         }
00804
00805     zadanie NoweZadanie;
00806     NoweZadanie.data = data;
00807     NoweZadanie.godzina = godzina;
00808     NoweZadanie.typ = typ;
00809     NoweZadanie.status = status;
00810     NoweZadanie.opis = opis;
00811
00812     vec.push_back(NoweZadanie);
00813 }
```

4.1.3.10 filtrujVecK()

```
vector< kontakt > filtrujVecK (
            vector< kontakt > kontakty,
            int x )
```

Funkcja filtrująca vector kontaktów.

Parametry

kontakty	Vector kontaktów który ma zostać pofiltrowany.
x	Zmienna oznaczająca to według której właściwości kontaktów zostanie pofiltrowany vector. Kolejno: x = 1 - filtruj według imion, x = 2 - filtruj według nazwisk, x = 3 - filtruj według E-maili.

Zwraca

vector<kontakt> Pofiltrowany vector kontaków według wybranej właściwości kontaktów

Definicja w linii 494 pliku organizer.cpp.

```
00495 {
00496     vector<kontakt> PofiltrowaneKontakty;
00497     switch (x)
00498     {
00499     case 1:
00500     {
00501         str imie;
00502         do
00503         {
```

```
00504             cout « "Podaj imie: " « endl;
00505             cin » imie;
00506             if (!TylkoLitery(imie))
00507                 cout « "Błędne imie. Spróbuj ponownie." « endl;
00508         } while (!TylkoLitery(imie));
00509
00510         for (int i = 0; i < kontakty.size(); i++)
00511         {
00512             if (kontakty[i].imie == imie)
00513                 PofiltrowaneKontakty.push_back(kontakty[i]);
00514         }
00515         break;
00516     }
00517     case 2:
00518     {
00519         str nazwisko;
00520         do
00521         {
00522             cout « "Podaj nazwisko: " « endl;
00523             cin » nazwisko;
00524             if (!TylkoLitery(nazwisko))
00525                 cout « "Błędne nazwisko. Spróbuj ponownie." « endl;
00526         } while (!TylkoLitery(nazwisko));
00527
00528         for (int i = 0; i < kontakty.size(); i++)
00529         {
00530             if (kontakty[i].nazwisko == nazwisko)
00531                 PofiltrowaneKontakty.push_back(kontakty[i]);
00532         }
00533         break;
00534     }
00535     case 3:
00536     {
00537         str email;
00538         do
00539         {
00540             cout « "Podaj E-Mail: " « endl;
00541             cin » email;
00542             if (!CzyEmail(email))
00543                 cout « "Błędny E-Mail. Spróbuj ponownie." « endl;
00544         } while (!CzyEmail(email));
00545
00546         for (int i = 0; i < kontakty.size(); i++)
00547         {
00548             if (kontakty[i].email == email)
00549                 PofiltrowaneKontakty.push_back(kontakty[i]);
00550         }
00551         break;
00552     }
00553     default:
00554         break;
00555     }
00556     return PofiltrowaneKontakty;
00557 }
```

4.1.3.11 filtrujVecZ()

```
vector< zadanie > filtrujVecZ (
            vector< zadanie > zadania,
            int x )
```

Funkcja filtrująca vector zadań.

Parametry

zadania	Vector zadań który ma zostać pofiltrowany.
x	Zmienna oznaczająca to według której właściwości zadań zostanie pofiltrowany vector. Kolejno: x = 1 - filtruj według daty, x = 2 - filtruj według godziny, x = 3 - filtruj według typu, x = 4 - filtruj według statusu, x = 5 - filtruj według opisu.

Zwraca

vector<zadanie> Pofiltrowany vector zadań według wybranej właściwości zadań.

Definicja w linii 559 pliku organizer.cpp.

```
00560 {
00561     vector<zadanie> PofiltrowaneZadania;
00562 
00563     switch (x)
00564     {
00565     case 1:
00566     {
00567         str data;
00568         do
00569         {
00570             cout « "Podaj datę: " « endl;
00571             cin » data;
00572             if (!CzyData(data))
00573                 cout « "Błędna data. Spróbuj ponownie." « endl;
00574         } while (!CzyData(data));
00575 
00576         for (int i = 0; i < zadania.size(); i++)
00577         {
00578             if (zadania[i].data == data)
00579                 PofiltrowaneZadania.push_back(zadania[i]);
00580         }
00581 
00582         break;
00583     }
00584     case 2:
00585     {
00586         str godzina;
00587         do
00588         {
00589             cout « "Podaj godzinę: " « endl;
00590             cin » godzina;
00591             if (!CzyGodzina(godzina))
00592                 cout « "Błędna godzina. Spróbuj ponownie." « endl;
00593         } while (!CzyGodzina(godzina));
00594 
00595         for (int i = 0; i < zadania.size(); i++)
00596         {
00597             if (zadania[i].godzina == godzina)
00598                 PofiltrowaneZadania.push_back(zadania[i]);
00599         }
00600 
00601         break;
00602     }
00603     case 3:
00604     {
00605         str typ;
00606         do
00607         {
00608             cout « "Podaj typ: " « endl;
00609             cin » typ;
00610             if (!TylkoLitery(typ))
00611                 cout « "Błędny typ. Spróbuj ponownie." « endl;
00612         } while (!TylkoLitery(typ));
00613 
00614         for (int i = 0; i < zadania.size(); i++)
00615         {
00616             if (zadania[i].typ == typ)
00617                 PofiltrowaneZadania.push_back(zadania[i]);
00618         }
00619 
00620         break;
00621     }
00622     case 4:
00623     {
00624         str status;
00625         do
00626         {
00627             cout « "Podaj status: (X lub V)" « endl;
00628             cin » status;
00629             if (status.size() > 1 || status != "X" && status != "x" && status != "V" && status != "v")
00630                 cout « "Błędny status. Spróbuj ponownie." « endl;
00631         } while (status.size() > 1 || status != "X" && status != "x" && status != "V" && status !=
    "v");
00632 
00633         for (int i = 0; i < zadania.size(); i++)
00634         {
00635             if (zadania[i].status == status)
00636                 PofiltrowaneZadania.push_back(zadania[i]);
00637         }
00638 
```

```
00639         break;
00640     }
00641     case 5:
00642     {
00643         str opis;
00644         cout « "Podaj opis: " « endl;
00645         getline(cin, opis);
00646 
00647         for (int i = 0; i < zadania.size(); i++)
00648         {
00649             if (zadania[i].opis == opis)
00650                 PofiltrowaneZadania.push_back(zadania[i]);
00651         }
00652         break;
00653     }
00654     default:
00655         break;
00656     }
00657 
00658     return PofiltrowaneZadania;
00659 }
```

4.1.3.12 main()

```
int main (
              int argc,
              char * args[] )
```

Funkcja main. Funkcja ta jest funkcją główną. Wykonuje ona ustawienia wstępne, uruchamia funkcję Start() oraz zajmuje się wywołaniem funkcji ZapiszWPlikach().

Parametry

argc	Parametr określający ilość argumentów podanych w linii poleceń przy uruchamianiu programu.
args	Tablica wskaźników do argumentów podanych w linii poleceń przy uruchamianiu programu.

Zwraca

int Liczba całkowita odzwierciedlająca to jak przebiegła praca z programem.

Definicja w linii 271 pliku organizer.cpp.

```
00272 {
00273     setlocale(LC_ALL, "pl_PL");
00274 
00275     system("chcp 65001");
00276     system("cls");
00277 
00278     string nazwa_pliku_wejsciowego_kontakty;
00279     string nazwa_pliku_wejsciowego_zadania;
00280     try
00281     {
00282         if (string(args[1]) == "-k" && string(args[3]) == "-z")
00283         {
00284             nazwa_pliku_wejsciowego_kontakty = args[2];
00285             nazwa_pliku_wejsciowego_zadania = args[4];
00286         }
00287         else if (string(args[3]) == "-k" && string(args[1]) == "-z")
00288         {
00289             nazwa_pliku_wejsciowego_kontakty = args[4];
00290             nazwa_pliku_wejsciowego_zadania = args[2];
00291         }
00292     }
00293     catch (exception &e)
00294     {
00295         printf("%s", "Nieprawidłowe wejście!");
00296         return 1;
00297     }
00298 
```

```
00299     vector<kontakt> kontakty = CzytajKontakty(nazwa_pliku_wejsciowego_kontakty);
00300     vector<zadanie> zadania = CzytajZadania(nazwa_pliku_wejsciowego_zadania);
00301
00302     while (DzialanieProgramu)
00303         Start(kontakty, zadania);
00304
00305     ZapiszWPlikach(nazwa_pliku_wejsciowego_kontakty, nazwa_pliku_wejsciowego_zadania, kontakty,
      zadania);
00306
00307     return 0;
00308 }
```

4.1.3.13 sortujVecK()

```
vector< kontakt > sortujVecK (
            vector< kontakt > vec,
            int x )
```

Funkcja sortująca vector kontaktów. Funkcje sortowania sortują alfabetycznie.

Parametry

vec	Vector kontaktów który ma zostać posortowany.
x	Zmienna oznaczająca to według której właściwości kontaktów zostanie posortowany vector. Kolejno: x = 1 - sortuj według imion, x = 2 - sortuj według nazwisk, x = 3 - sortuj według E-maili.

Zwraca

vector<kontakt> Posortowany vector kontaków według wybranej właściwości kontaktów.

Definicja w linii 340 pliku organizer.cpp.

```
00341 {
00342     vector<string> PosortowaneImiona;
00343     vector<string> PosortowaneNazwiska;
00344     vector<string> PosortowaneEmaile;
00345     vector<kontakt> PosortowaneKontakty;
00346
00347     for (auto elem : vec)
00348     {
00349         string klucz1 = elem.imie;
00350         string klucz2 = elem.nazwisko;
00351         string klucz3 = elem.email;
00352         if (!count(PosortowaneImiona.begin(), PosortowaneImiona.end(), klucz1))
00353             PosortowaneImiona.push_back(elem.imie);
00354         if (!count(PosortowaneNazwiska.begin(), PosortowaneNazwiska.end(), klucz2))
00355             PosortowaneNazwiska.push_back(elem.nazwisko);
00356         if (!count(PosortowaneEmaile.begin(), PosortowaneEmaile.end(), klucz3))
00357             PosortowaneEmaile.push_back(elem.email);
00358     }
00359
00360     switch (x)
00361     {
00362     case 1:
00363         sort(PosortowaneImiona.begin(), PosortowaneImiona.end());
00364         for (auto elem : PosortowaneImiona)
00365         {
00366             for (auto elem1 : vec)
00367             {
00368                 if (elem1.imie == elem)
00369                     PosortowaneKontakty.push_back(elem1);
00370             }
00371         }
00372         break;
00373     case 2:
00374         sort(PosortowaneNazwiska.begin(), PosortowaneNazwiska.end());
00375         for (auto elem : PosortowaneNazwiska)
00376         {
00377             for (auto elem1 : vec)
```

```
00378             {
00379                 if (elem1.nazwisko == elem)
00380                     PosortowaneKontakty.push_back(elem1);
00381             }
00382         }
00383         break;
00384     case 3:
00385         sort(PosortowaneEmaile.begin(), PosortowaneEmaile.end());
00386         for (auto elem : PosortowaneEmaile)
00387         {
00388             for (auto elem1 : vec)
00389             {
00390                 if (elem1.email == elem)
00391                     PosortowaneKontakty.push_back(elem1);
00392             }
00393         }
00394         break;
00395
00396     default:
00397         break;
00398     }
00399     return PosortowaneKontakty;
00400 }
```

4.1.3.14 sortujVecZ()

```
vector< zadanie > sortujVecZ (
            vector< zadanie > vec,
            int x )
```

Funkcja sortująca vector zadań. Funkcje sortowania sortują alfabetycznie.

Parametry

vec	Vector zadań który ma zostać posortowany.
x	Zmienna oznaczająca to według której właściwości zadań zostanie posortowany vector. Kolejno: x = 1 - sortuj według daty, x = 2 - sortuj według godziny, x = 3 - sortuj według typu, x = 4 - sortuj według statusu, x = 5 - sortuj według opisu.

Zwraca

vector<zadanie> Posortowany vector zadań według wybranej właściwości zadań.

Definicja w linii 402 pliku organizer.cpp.

```
00403 {
00404     vector<string> PosortowaneDaty;
00405     vector<string> PosortowaneGodziny;
00406     vector<string> PosortowaneTypy;
00407     vector<string> PosortowaneStatusy;
00408     vector<string> PosortowaneOpisy;
00409     vector<zadanie> PosortowaneZadania;
00410
00411     for (auto elem : vec)
00412     {
00413         string klucz1 = elem.data;
00414         string klucz2 = elem.godzina;
00415         string klucz3 = elem.typ;
00416         string klucz4 = elem.status;
00417         string klucz5 = elem.opis;
00418         if (!count(PosortowaneDaty.begin(), PosortowaneDaty.end(), klucz1))
00419             PosortowaneDaty.push_back(elem.data);
00420         if (!count(PosortowaneGodziny.begin(), PosortowaneGodziny.end(), klucz2))
00421             PosortowaneGodziny.push_back(elem.godzina);
00422         if (!count(PosortowaneTypy.begin(), PosortowaneTypy.end(), klucz3))
00423             PosortowaneTypy.push_back(elem.typ);
00424         if (!count(PosortowaneStatusy.begin(), PosortowaneStatusy.end(), klucz4))
00425             PosortowaneStatusy.push_back(elem.status);
```

```
00426         if (!count(PosortowaneOpisy.begin(), PosortowaneOpisy.end(), klucz5))
00427             PosortowaneOpisy.push_back(elem.opis);
00428     }
00429
00430     switch (x)
00431     {
00432     case 1:
00433         sort(PosortowaneDaty.begin(), PosortowaneDaty.end());
00434         for (auto elem : PosortowaneDaty)
00435         {
00436             for (auto elem1 : vec)
00437             {
00438                 if (elem1.data == elem)
00439                     PosortowaneZadania.push_back(elem1);
00440             }
00441         }
00442         break;
00443     case 2:
00444         sort(PosortowaneGodziny.begin(), PosortowaneGodziny.end());
00445         for (auto elem : PosortowaneGodziny)
00446         {
00447             for (auto elem1 : vec)
00448             {
00449                 if (elem1.godzina == elem)
00450                     PosortowaneZadania.push_back(elem1);
00451             }
00452         }
00453         break;
00454     case 3:
00455         sort(PosortowaneTypy.begin(), PosortowaneTypy.end());
00456         for (auto elem : PosortowaneTypy)
00457         {
00458             for (auto elem1 : vec)
00459             {
00460                 if (elem1.typ == elem)
00461                     PosortowaneZadania.push_back(elem1);
00462             }
00463         }
00464         break;
00465     case 4:
00466         sort(PosortowaneStatusy.begin(), PosortowaneStatusy.end());
00467         for (auto elem : PosortowaneStatusy)
00468         {
00469             for (auto elem1 : vec)
00470             {
00471                 if (elem1.status == elem)
00472                     PosortowaneZadania.push_back(elem1);
00473             }
00474         }
00475         break;
00476     case 5:
00477         sort(PosortowaneOpisy.begin(), PosortowaneOpisy.end());
00478         for (auto elem : PosortowaneOpisy)
00479         {
00480             for (auto elem1 : vec)
00481             {
00482                 if (elem1.opis == elem)
00483                     PosortowaneZadania.push_back(elem1);
00484             }
00485         }
00486         break;
00487
00488     default:
00489         break;
00490     }
00491     return PosortowaneZadania;
00492 }
```

4.1.3.15 Start()

```
void Start (
            vector< kontakt > & kontakty,
            vector< zadanie > & zadania )
```

Funkcja uruchamiająca główne menu programu i wykonująca podprogramy. Funkcja ta przyjmuje vector kontaktów oraz vector zadań i za ich pomocą uruchamia odpowiedni podprogram wybrany przez użytkownika.

Parametry

kontakty	Referencja do vectora kontaktów przechowującego sczytane z pliku kontakty.
zadania	Referencja do vectora zadań przechowującego sczytane z pliku zadania.

Definicja w linii 1083 pliku organizer.cpp.

```
01084 {
01085     int x = 0;
01086     do
01087     {
01088         cout « "Witaj w Organizerze! Co chciał(a)byś zrobić?" « endl;
01089         cout « "1.Dodaj kontakt \n 2.Usuń kontakt \n 3.Dodaj zadanie \n 4.Usuń zadanie \n 5.Wyświetl
      kontakty \n 6.Wyświetl zadania \n 7.Zmień status zadania \n 8.Zakończ działanie programu" « endl;
01090         cin » x;
01091         if (cin.fail())
01092         {
01093             x = 0;
01094             cout « "Błąd konwersji. Najprawdopodobniej wejście nie jest liczbą. Spróbuj ponownie." «
      endl;
01095             cin.clear();
01096             str ignore;
01097             cin » ignore;
01098             continue;
01099         }
01100         else if (x != 1 && x != 2 && x != 3 && x != 4 && x != 5 && x != 6 && x != 7 && x != 8)
01101         {
01102             x = 0;
01103             cout « "Nie ma takiej opcji. Spróbuj ponownie." « endl;
01104         }
01105     } while (x < 1 || x > 8);
01106
01107     switch (x)
01108     {
01109     case 1:
01110         DodajKontakt(kontakty);
01111         if(show){
01112             cout « endl
01113              « "Podsumowanie:" « endl;
01114             WypiszVectorK(kontakty);
01115         }
01116         show = true;
01117         Zapytanie();
01118         break;
01119     case 2:
01120         UsunKontakt(kontakty);
01121         if(show){
01122             cout « endl
01123              « "Podsumowanie:" « endl;
01124             WypiszVectorK(kontakty);
01125         }
01126         show = true;
01127         Zapytanie();
01128         break;
01129     case 3:
01130         DodajZadanie(zadania);
01131         if(show){
01132             cout « endl
01133              « "Podsumowanie:" « endl;
01134             WypiszVectorZ(zadania);
01135         }
01136         show = true;
01137         Zapytanie();
01138         break;
01139     case 4:
01140         UsunZadanie(zadania);
01141         if(show){
01142             cout « endl
01143              « "Podsumowanie:" « endl;
01144             WypiszVectorZ(zadania);
01145         }
01146         show = true;
01147         Zapytanie();
01148         break;
01149     case 5:
01150     {
01151         vector<kontakt> PosortowaneKontakty;
01152         vector<kontakt> PofiltrowaneKontakty;
01153
01154         if (CzyFiltrowac())
01155         {
01156             int kryterium = WedlugCzegoK("Pofiltrować");
01157             PofiltrowaneKontakty = filtrujVecK(kontakty, kryterium);
```

```
01158             if (CzySortowac())
01159             {
01160                 int kryterium = WedlugCzegoK("Posortować");
01161                 PosortowaneKontakty = sortujVecK(PofiltrowaneKontakty, kryterium);
01162                 WypiszVectorK(PosortowaneKontakty);
01163             }
01164             else
01165             {
01166                 WypiszVectorK(PofiltrowaneKontakty);
01167             }
01168         }
01169         else
01170         {
01171             if (CzySortowac())
01172             {
01173                 int kryterium = WedlugCzegoK("Posortować");
01174                 PosortowaneKontakty = sortujVecK(kontakty, kryterium);
01175                 WypiszVectorK(PosortowaneKontakty);
01176             }
01177             else
01178                 WypiszVectorK(kontakty);
01179         }
01180         Zapytanie();
01181         break;
01182     }
01183     case 6:
01184     {
01185         vector<zadanie> PosortowaneZadania;
01186         vector<zadanie> PofiltrowaneZadania;
01187 
01188         if (CzyFiltrowac())
01189         {
01190             int kryterium = WedlugCzegoZ("Pofiltrować");
01191             PofiltrowaneZadania = filtrujVecZ(zadania, kryterium);
01192             if (CzySortowac())
01193             {
01194                 int kryterium = WedlugCzegoZ("Posortować");
01195                 PosortowaneZadania = sortujVecZ(PofiltrowaneZadania, kryterium);
01196                 WypiszVectorZ(PosortowaneZadania);
01197             }
01198             else
01199             {
01200                 WypiszVectorZ(PofiltrowaneZadania);
01201             }
01202         }
01203         else
01204         {
01205             if (CzySortowac())
01206             {
01207                 int kryterium = WedlugCzegoZ("Posortować");
01208                 PosortowaneZadania = sortujVecZ(zadania, kryterium);
01209                 WypiszVectorZ(PosortowaneZadania);
01210             }
01211             else
01212                 WypiszVectorZ(zadania);
01213         }
01214         Zapytanie();
01215         break;
01216     }
01217     case 7:
01218         ZmienStatus(zadania);
01219         if(show){
01220             cout « endl
01221              « "Podsumowanie:" « endl;
01222             WypiszVectorZ(zadania);
01223         }
01224         show = true;
01225         Zapytanie();
01226         break;
01227     case 8:
01228         DzialanieProgramu = false;
01229         break;
01230 
01231     default:
01232         break;
01233     }
01234 }
```

4.1.3.16 TylkoLitery()

```
bool TylkoLitery (
            string nazwa )
```

Funkcja sprawdzająca czy dany ciąg znaków zawiera tylko litery. Funkcja ta przyjmuje ciąg znaków i sprawdza czy zawiera on tylko znaki alfanumeryczne za pomocą wyrażenia regularnego po czym zwraca prawdę lub fałsz w zależności od wyniku.

Parametry

nazwa	Sprawdzany ciąg znaków.

Zwraca

true Wartość zwracana gdy ciąg znaków zawiera tylko znaki alfanumeryczne.

false Wartość zwracana gdy ciąg znaków nie zawiera tylko znaków alfanumerycznych.

Definicja w linii 1236 pliku organizer.cpp.

```
01237 {
01238     for (char znak : nazwa)
01239     {
01240         if (!isalpha(znak))
01241         {
01242             return false;
01243         }
01244     }
01245     return true;
01246 }
```

4.1.3.17 UsunKontakt()

```
void UsunKontakt (
            vector< kontakt > & vec )
```

Funkcja usuwająca kontakt z vectora kontaktów. Funkcja ta usuwa wybrany przez użytkownika kontakt z vectora kontaktów.

Parametry

vec	Referencja do vectora kontaktów z którego zostanie usunięty kontakt.

Definicja w linii 706 pliku organizer.cpp.

```
00707 {
00708     str x;
00709     int numer = -1;
00710     WypiszVectorK(vec);
00711     do
00712     {
00713         cout « "Który kontakt usunąć? A - anuluj" « endl;
00714         cin » x;
00715         if (x == "A"){
00716             show = false;
00717             return;
00718         }
00719         for (char znak : x)
00720         {
00721             if (!isdigit(znak))
00722                 continue;
00723         }
00724         try
00725         {
00726             numer = stoi(x);
00727         }
00728         catch (exception &e)
00729         {
```

```
00730             cout « "Błąd konwersji. Najprawdopodobniej wejście nie jest liczbą. Spróbuj ponownie." «
      endl;
00731             numer = -1;
00732             continue;
00733         }
00734         numer = numer - 1;
00735         if (numer < 0 || numer > vec.size() - 1)
00736         {
00737             cout « "Nie ma takiego kontaktu. Spróbuj ponownie." « endl;
00738             continue;
00739         }
00740     } while (numer < 0 || numer > vec.size() - 1);
00741     vec.erase(vec.begin() + numer);
00742 }
```

4.1.3.18 UsunZadanie()

```
void UsunZadanie (
            vector< zadanie > & vec )
```

Funkcja usuwająca zadanie z vectora zadań. Funkcja ta usuwa wybrane przez użytkownika zadanie z vectora zadań.

Parametry

vec	Referencja do vectora zadań z którego zostanie usunięte zadanie.

Definicja w linii 815 pliku organizer.cpp.

```
00816 {
00817     str x;
00818     int numer = -1;
00819     WypiszVectorZ(vec);
00820     do
00821     {
00822         cout « "Które zadanie usunąć? A-anuluj" « endl;
00823         cin » x;
00824         if (x == "A"){
00825             show = false;
00826             return;
00827         }
00828         for (char znak : x)
00829         {
00830             if (!isdigit(znak))
00831                 continue;
00832         }
00833         try
00834         {
00835             numer = stoi(x);
00836         }
00837         catch (exception &e)
00838         {
00839             cout « "Błąd konwersji. Najprawdopodobniej wejście nie jest liczbą. Spróbuj ponownie." «
      endl;
00840             numer = -1;
00841             continue;
00842         }
00843         numer = numer - 1;
00844         if (numer < 0 || numer > vec.size() - 1)
00845             cout « "Nie ma takiego zadania. Spróbuj ponownie." « endl;
00846     } while (numer < 0 || numer > vec.size() - 1);
00847     vec.erase(vec.begin() + numer);
00848 }
```

4.1.3.19 WedlugCzegoK()

```
int WedlugCzegoK (
            string flag )
```

Funkcja pytająca użytkownika według jakiej właściwości kontaktu posortować lub pofiltrować kontakty. Funkcja ta pyta użytkownika według jakiej właściwości kontaktu chce sortować lub filtrować kontakty i zwraca jego decyzję w postaci liczby całkowitej.

Parametry

flag	Parametr będący stringiem który służy do tego by wyświetlić odpowiednie słowo (Posortuj lub Pofiltruj) w zależności od tego o co własnie pyta program.

Zwraca

int Liczba całkowita będąca odzwierciedleniem decyzji użytkownika o tym według jakiej właściwości kontaktu sortować lub filtrować kontakty. Kolejno: 1 - imie, 2 - nazwisko, 3 - e-mail.

Definicja w linii 991 pliku organizer.cpp.

```
00992 {
00993     int x = 0;
00994     do
00995     {
00996         do
00997         {
00998             cout « flag « " kontakty według: \n 1.Imienia \n 2.Nazwiska \n 3.E-Maila" « endl;
00999             cin » x;
01000             if (cin.fail())
01001                 cout « "Błąd konwersji. Najprawdopodobniej wejście nie jest liczbą. Spróbuj ponownie."
      « endl;
01002         } while (cin.fail());
01003         if (x != 1 && x != 2 && x != 3)
01004         {
01005             x = 0;
01006             cout « "Nie ma takiej opcji. Spróbuj ponownie." « endl;
01007         }
01008     } while (x < 1 || x > 3);
01009     return x;
01010 }
```

4.1.3.20 WedlugCzegoZ()

```
int WedlugCzegoZ (
              string flag )
```

Funkcja pytająca użytkownika według jakiej właściwości zadania posortować lub pofiltrować zadania. Funkcja ta pyta użytkownika według jakiej właściwości zadania chce sortować lub filtrować zadania i zwraca jego decyzję w postaci liczby całkowitej.

Parametry

flag	Parametr będący stringiem który służy do tego by wyświetlić odpowiednie słowo (Posortuj lub Pofiltruj) w zależności od tego o co własnie pyta program.

Zwraca

int Liczba całkowita będąca odzwierciedleniem decyzji użytkownika o tym według jakiej właściwości zadania sortować lub filtrować zadania. Kolejno: 1 - data, 2 - godzina, 3 - typ, 4 - status, 5 - opis.

Definicja w linii 1012 pliku organizer.cpp.

```
01013 {
```

```
01014     int x = 0;
01015     do
01016     {
01017         do
01018         {
01019             cout « flag « " zadania według: \n 1.Daty \n 2.Godziny \n 3.Typu \n 4.Statusu \n 5.Opisu"
      « endl;
01020             cin » x;
01021             if (cin.fail())
01022                 cout « "Błąd konwersji. Najprawdopodobniej wejście nie jest liczbą. Spróbuj ponownie."
      « endl;
01023         } while (cin.fail());
01024
01025         if (x != 1 && x != 2 && x != 3 && x != 4 && x != 5)
01026         {
01027             x = 0;
01028             cout « "Nie ma takiej opcji. Spróbuj ponownie." « endl;
01029         }
01030     } while (x < 1 || x > 5);
01031     return x;
01032 }
```

4.1.3.21 WypiszVectorK()

```
void WypiszVectorK (
            vector< kontakt > & vec )
```

Funkcja wypisująca vector kontaktów.

Parametry

vec	Referencja do vectora kontaktów który ma zostać wypisany.

Definicja w linii 310 pliku organizer.cpp.

```
00311 {
00312     for (int i = 0; i < vec.size(); i++)
00313     {
00314         cout « endl;
00315         cout « i + 1 « "." « endl;
00316         cout « vec[i].imie « endl;
00317         cout « vec[i].nazwisko « endl;
00318         cout « vec[i].email « endl;
00319         cout « endl;
00320         cout « "-------------------------------------------------" « endl;
00321     }
00322 }
```

4.1.3.22 WypiszVectorZ()

```
void WypiszVectorZ (
            vector< zadanie > & vec )
```

Funkcja wypisująca vector zadań.

Parametry

vec	Referencja do vectora zadań który ma zostać wypisany.

Definicja w linii 324 pliku organizer.cpp.

```
00325 {
00326     for (int i = 0; i < vec.size(); i++)
00327     {
00328         cout « endl;
00329         cout « i + 1 « "." « endl;
00330         cout « vec[i].data « endl;
00331         cout « vec[i].godzina « endl;
00332         cout « vec[i].typ « endl;
00333         cout « vec[i].status « endl;
00334         cout « vec[i].opis « endl;
00335         cout « endl;
00336         cout « "-------------------------------------------------" « endl;
00337     }
00338 }
```

4.1.3.23 ZapiszWPlikach()

```
void ZapiszWPlikach (
            str nazwa_pliku_kontakty,
            str nazwa_pliku_zadania,
            vector< kontakt > & vecK,
            vector< zadanie > & vecZ )
```

Funkcja zapisująca kontakty i zadania w plikach tekstowych. Funkcja ta zapisuje kontakty i zadania z odpowiadających sobie vectorów do pliku tekstowego. Funkcja używa tych samych plików tekstowych co funkcje czytające dane z plików przy starcie programu.

Parametry

nazwa_pliku_kontakty	Nazwa pliku do którego mają zostać zapisane kontakty.
nazwa_pliku_zadania	Nazwa pliku do którego mają zostać zapisane zadania.
vecK	Vector kontaktów zawierający kontakty które mają zostać zapisane w pliku tekstowym.
vecZ	Vector zadań zawierający zadania które mają zostać zapisane w pliku tekstowym.

Definicja w linii 951 pliku organizer.cpp.

```
00952 {
00953     ofstream wyjscieKontakty(nazwa_pliku_kontakty, ios::out | ios::trunc);
00954     ofstream wyjscieZadania(nazwa_pliku_zadania, ios::out | ios::trunc);
00955
00956     for (auto elem : vecK)
00957     {
00958         wyjscieKontakty « elem.imie « " " « elem.nazwisko « " " « elem.email « endl;
00959     }
00960
00961     for (auto elem : vecZ)
00962     {
00963         wyjscieZadania « elem.data « " " « elem.godzina « " " « elem.typ « " " « elem.status « " " «
      elem.opis « endl;
00964     }
00965
00966     wyjscieKontakty.close();
00967     wyjscieZadania.close();
00968 }
```

4.1.3.24 Zapytanie()

```
void Zapytanie ( )
```

Funkcja pytająca użytkownika o chęć kontynuacji działania z programem. Funkcja ta pyta użytkownika czy chce on powrócić do menu głównego (1) czy też zakończyć działanie z programem (2).

Definicja w linii 970 pliku organizer.cpp.

```
00971 {
00972     int x = 0;
00973     do
00974     {
00975         cout « "1.Wróć do menu \n 2.Zakończ" « endl;
00976         cin » x;
00977         if(cin.fail()){
00978             cin.clear();
00979             cin.ignore(std::numeric_limits<int>::max(), '\n');
00980             x = 0;
00981             cout«"Błąd konwersji. Wejście najprawdopodobniej nie jest liczbą. Spróbuj ponownie."«endl;
00982         }
00983         else if (x != 1 && x != 2) cout « "Nie ma takiej opcji. Spróbuj ponownie." « endl;
00984     } while (x != 1 && x != 2);
00985     if (x == 1)
00986         system("cls");
00987     if (x == 2)
00988         DzialanieProgramu = false;
00989 }
```

4.1.3.25 ZmienStatus()

```
void ZmienStatus (
            vector< zadanie > & vec )
```

Funkcja zmieniająca status zadania. Funkcja ta zmienia status zadania z wykonanego na niewykonane lub na odwrót (V -> X lub X -> V).

Parametry

vec	Referencja do vectora zadań który przechowuje zadanie którego status zostanie zmieniony.

Definicja w linii 850 pliku organizer.cpp.

```
00851 {
00852     str x;
00853     int numer = -1;
00854     WypiszVectorZ(vec);
00855     do
00856     {
00857         cout « "Status którego zadania chcesz zmienić? A-anuluj" « endl;
00858         cin » x;
00859         if (x == "A"){
00860             show = false;
00861             return;
00862         }
00863         for (char znak : x)
00864         {
00865             if (!isdigit(znak))
00866                 continue;
00867         }
00868         try
00869         {
00870             numer = stoi(x);
00871         }
00872         catch (exception &e)
00873         {
00874             cout « "Błąd konwersji. Najprawdopodobniej wejście nie jest liczbą. Spróbuj ponownie." «
      endl;
00875             numer = -1;
00876             continue;
00877         }
00878         if (numer < 1 || numer > vec.size())
00879             cout « "Nie ma takiego zadania. Spróbuj ponownie." « endl;
00880     } while (numer < 1 || numer > vec.size());
00881
00882     numer = numer - 1;
```

```
00883
00884     string zamiennik;
00885
00886     do
00887     {
00888         cout « "Wprowadź status V(wykonane) lub X(niewykonane): " « endl;
00889         cin » zamiennik;
00890     } while (zamiennik != "V" && zamiennik != "X" && zamiennik != "v" && zamiennik != "x");
00891
00892     if (zamiennik == "v")
00893         zamiennik = "V";
00894     if (zamiennik == "x")
00895         zamiennik = "X";
00896
00897     vec[numer].status = zamiennik;
00898 }
```

4.1.4 Dokumentacja zmiennych

4.1.4.1 DzialanieProgramu

```
bool DzialanieProgramu = true
```

Na podstawie wartości tej zmiennej program kończy swoje działanie. Gdy zmienna ma wartość true program kontynuuje swoje działanie, a gdy false zostaje zakończony.

Definicja w linii 37 pliku organizer.cpp.

4.1.4.2 show

```
bool show = true
```

Na podstawie tej zmiennej wyświetlane jest podsumowanie. Gdy zmienna ma wartość true zostanie pokazane podsumowanie, a gdy false nie.

Definicja w linii 43 pliku organizer.cpp.

4.2 organizer.cpp

Idź do dokumentacji tego pliku.

```
00001
00012 #include <iostream>
00013 #include <vector>
00014 #include <string>
00015 #include <algorithm>
00016 #include <cmath>
00017 #include <fstream>
00018 #include <cctype>
00019 #include <regex>
00020
00025 using namespace std;
00026
00031 using str = string;
00032
00037 bool DzialanieProgramu = true;
00038
00043 bool show = true;
```

```
00044
00050 struct kontakt
00051 {
00052     str imie;
00053     str nazwisko;
00054     str email;
00055 };
00056
00062 struct zadanie
00063 {
00064     str data;
00065     str godzina;
00066     str typ;
00067     str status;
00068     str opis;
00069 };
00070
00076 void WypiszVectorK(vector<kontakt> &vec);
00077
00083 void WypiszVectorZ(vector<zadanie> &vec);
00084
00092 vector<kontakt> sortujVecK(vector<kontakt> vec, int x);
00093
00101 vector<zadanie> sortujVecZ(vector<zadanie> vec, int x);
00102
00110 vector<kontakt> filtrujVecK(vector<kontakt> kontakty, int x);
00111
00119 vector<zadanie> filtrujVecZ(vector<zadanie> zadania, int x);
00120
00126 void DodajKontakt(vector<kontakt> &vec);
00127
00133 void DodajZadanie(vector<zadanie> &vec);
00134
00140 void UsunKontakt(vector<kontakt> &vec);
00141
00147 void UsunZadanie(vector<zadanie> &vec);
00148
00154 void ZmienStatus(vector<zadanie> &vec);
00155
00162 vector<kontakt> CzytajKontakty(str nazwa_pliku_kontakty);
00163
00170 vector<zadanie> CzytajZadania(str nazwa_pliku_zadania);
00171
00180 void ZapiszWPlikach(str nazwa_pliku_kontakty, str nazwa_pliku_zadania, vector<kontakt> &vecK,
      vector<zadanie> &vecZ);
00181
00186 void Zapytanie();
00187
00194 int WedlugCzegoK(string flag);
00195
00202 int WedlugCzegoZ(string flag);
00203
00210 bool CzySortowac();
00211
00218 bool CzyFiltrowac();
00219
00227 bool TylkoLitery(string nazwa);
00228
00236 bool CzyEmail(string email);
00237
00245 bool CzyData(string data);
00246
00254 bool CzyGodzina(string godzina);
00255
00262 void Start(vector<kontakt> &kontakty, vector<zadanie> &zadania);
00263
00271 int main(int argc, char *args[])
00272 {
00273     setlocale(LC_ALL, "pl_PL");
00274
00275     system("chcp 65001");
00276     system("cls");
00277
00278     string nazwa_pliku_wejsciowego_kontakty;
00279     string nazwa_pliku_wejsciowego_zadania;
00280     try
00281     {
00282         if (string(args[1]) == "-k" && string(args[3]) == "-z")
00283         {
00284             nazwa_pliku_wejsciowego_kontakty = args[2];
00285             nazwa_pliku_wejsciowego_zadania = args[4];
00286         }
00287         else if (string(args[3]) == "-k" && string(args[1]) == "-z")
00288         {
00289             nazwa_pliku_wejsciowego_kontakty = args[4];
00290             nazwa_pliku_wejsciowego_zadania = args[2];
00291         }
```

```
00292     }
00293     catch (exception &e)
00294     {
00295         printf("%s", "Nieprawidłowe wejście!");
00296         return 1;
00297     }
00298
00299     vector<kontakt> kontakty = CzytajKontakty(nazwa_pliku_wejsciowego_kontakty);
00300     vector<zadanie> zadania = CzytajZadania(nazwa_pliku_wejsciowego_zadania);
00301
00302     while (DzialanieProgramu)
00303         Start(kontakty, zadania);
00304
00305     ZapiszWPlikach(nazwa_pliku_wejsciowego_kontakty, nazwa_pliku_wejsciowego_zadania, kontakty,
      zadania);
00306
00307     return 0;
00308 }
00309
00310 void WypiszVectorK(vector<kontakt> &vec)
00311 {
00312     for (int i = 0; i < vec.size(); i++)
00313     {
00314         cout « endl;
00315         cout « i + 1 « "." « endl;
00316         cout « vec[i].imie « endl;
00317         cout « vec[i].nazwisko « endl;
00318         cout « vec[i].email « endl;
00319         cout « endl;
00320         cout « "-------------------------------------------------" « endl;
00321     }
00322 }
00323
00324 void WypiszVectorZ(vector<zadanie> &vec)
00325 {
00326     for (int i = 0; i < vec.size(); i++)
00327     {
00328         cout « endl;
00329         cout « i + 1 « "." « endl;
00330         cout « vec[i].data « endl;
00331         cout « vec[i].godzina « endl;
00332         cout « vec[i].typ « endl;
00333         cout « vec[i].status « endl;
00334         cout « vec[i].opis « endl;
00335         cout « endl;
00336         cout « "-------------------------------------------------" « endl;
00337     }
00338 }
00339
00340 vector<kontakt> sortujVecK(vector<kontakt> vec, int x)
00341 {
00342     vector<string> PosortowaneImiona;
00343     vector<string> PosortowaneNazwiska;
00344     vector<string> PosortowaneEmaile;
00345     vector<kontakt> PosortowaneKontakty;
00346
00347     for (auto elem : vec)
00348     {
00349         string klucz1 = elem.imie;
00350         string klucz2 = elem.nazwisko;
00351         string klucz3 = elem.email;
00352         if (!count(PosortowaneImiona.begin(), PosortowaneImiona.end(), klucz1))
00353             PosortowaneImiona.push_back(elem.imie);
00354         if (!count(PosortowaneNazwiska.begin(), PosortowaneNazwiska.end(), klucz2))
00355             PosortowaneNazwiska.push_back(elem.nazwisko);
00356         if (!count(PosortowaneEmaile.begin(), PosortowaneEmaile.end(), klucz3))
00357             PosortowaneEmaile.push_back(elem.email);
00358     }
00359
00360     switch (x)
00361     {
00362     case 1:
00363         sort(PosortowaneImiona.begin(), PosortowaneImiona.end());
00364         for (auto elem : PosortowaneImiona)
00365         {
00366             for (auto elem1 : vec)
00367             {
00368                 if (elem1.imie == elem)
00369                     PosortowaneKontakty.push_back(elem1);
00370             }
00371         }
00372         break;
00373     case 2:
00374         sort(PosortowaneNazwiska.begin(), PosortowaneNazwiska.end());
00375         for (auto elem : PosortowaneNazwiska)
00376         {
00377             for (auto elem1 : vec)
```

```
00378             {
00379                 if (elem1.nazwisko == elem)
00380                     PosortowaneKontakty.push_back(elem1);
00381             }
00382         }
00383         break;
00384     case 3:
00385         sort(PosortowaneEmaile.begin(), PosortowaneEmaile.end());
00386         for (auto elem : PosortowaneEmaile)
00387         {
00388             for (auto elem1 : vec)
00389             {
00390                 if (elem1.email == elem)
00391                     PosortowaneKontakty.push_back(elem1);
00392             }
00393         }
00394         break;
00395 
00396     default:
00397         break;
00398     }
00399     return PosortowaneKontakty;
00400 }
00401 
00402 vector<zadanie> sortujVecZ(vector<zadanie> vec, int x)
00403 {
00404     vector<string> PosortowaneDaty;
00405     vector<string> PosortowaneGodziny;
00406     vector<string> PosortowaneTypy;
00407     vector<string> PosortowaneStatusy;
00408     vector<string> PosortowaneOpisy;
00409     vector<zadanie> PosortowaneZadania;
00410 
00411     for (auto elem : vec)
00412     {
00413         string klucz1 = elem.data;
00414         string klucz2 = elem.godzina;
00415         string klucz3 = elem.typ;
00416         string klucz4 = elem.status;
00417         string klucz5 = elem.opis;
00418         if (!count(PosortowaneDaty.begin(), PosortowaneDaty.end(), klucz1))
00419             PosortowaneDaty.push_back(elem.data);
00420         if (!count(PosortowaneGodziny.begin(), PosortowaneGodziny.end(), klucz2))
00421             PosortowaneGodziny.push_back(elem.godzina);
00422         if (!count(PosortowaneTypy.begin(), PosortowaneTypy.end(), klucz3))
00423             PosortowaneTypy.push_back(elem.typ);
00424         if (!count(PosortowaneStatusy.begin(), PosortowaneStatusy.end(), klucz4))
00425             PosortowaneStatusy.push_back(elem.status);
00426         if (!count(PosortowaneOpisy.begin(), PosortowaneOpisy.end(), klucz5))
00427             PosortowaneOpisy.push_back(elem.opis);
00428     }
00429 
00430     switch (x)
00431     {
00432     case 1:
00433         sort(PosortowaneDaty.begin(), PosortowaneDaty.end());
00434         for (auto elem : PosortowaneDaty)
00435         {
00436             for (auto elem1 : vec)
00437             {
00438                 if (elem1.data == elem)
00439                     PosortowaneZadania.push_back(elem1);
00440             }
00441         }
00442         break;
00443     case 2:
00444         sort(PosortowaneGodziny.begin(), PosortowaneGodziny.end());
00445         for (auto elem : PosortowaneGodziny)
00446         {
00447             for (auto elem1 : vec)
00448             {
00449                 if (elem1.godzina == elem)
00450                     PosortowaneZadania.push_back(elem1);
00451             }
00452         }
00453         break;
00454     case 3:
00455         sort(PosortowaneTypy.begin(), PosortowaneTypy.end());
00456         for (auto elem : PosortowaneTypy)
00457         {
00458             for (auto elem1 : vec)
00459             {
00460                 if (elem1.typ == elem)
00461                     PosortowaneZadania.push_back(elem1);
00462             }
00463         }
00464         break;
```

```
00465     case 4:
00466         sort(PosortowaneStatusy.begin(), PosortowaneStatusy.end());
00467         for (auto elem : PosortowaneStatusy)
00468         {
00469             for (auto elem1 : vec)
00470             {
00471                 if (elem1.status == elem)
00472                     PosortowaneZadania.push_back(elem1);
00473             }
00474         }
00475         break;
00476     case 5:
00477         sort(PosortowaneOpisy.begin(), PosortowaneOpisy.end());
00478         for (auto elem : PosortowaneOpisy)
00479         {
00480             for (auto elem1 : vec)
00481             {
00482                 if (elem1.opis == elem)
00483                     PosortowaneZadania.push_back(elem1);
00484             }
00485         }
00486         break;
00487 
00488     default:
00489         break;
00490     }
00491     return PosortowaneZadania;
00492 }
00493 
00494 vector<kontakt> filtrujVecK(vector<kontakt> kontakty, int x)
00495 {
00496     vector<kontakt> PofiltrowaneKontakty;
00497     switch (x)
00498     {
00499     case 1:
00500     {
00501         str imie;
00502         do
00503         {
00504             cout « "Podaj imie: " « endl;
00505             cin » imie;
00506             if (!TylkoLitery(imie))
00507                 cout « "Błędne imie. Spróbuj ponownie." « endl;
00508         } while (!TylkoLitery(imie));
00509 
00510         for (int i = 0; i < kontakty.size(); i++)
00511         {
00512             if (kontakty[i].imie == imie)
00513                 PofiltrowaneKontakty.push_back(kontakty[i]);
00514         }
00515         break;
00516     }
00517     case 2:
00518     {
00519         str nazwisko;
00520         do
00521         {
00522             cout « "Podaj nazwisko: " « endl;
00523             cin » nazwisko;
00524             if (!TylkoLitery(nazwisko))
00525                 cout « "Błędne nazwisko. Spróbuj ponownie." « endl;
00526         } while (!TylkoLitery(nazwisko));
00527 
00528         for (int i = 0; i < kontakty.size(); i++)
00529         {
00530             if (kontakty[i].nazwisko == nazwisko)
00531                 PofiltrowaneKontakty.push_back(kontakty[i]);
00532         }
00533         break;
00534     }
00535     case 3:
00536     {
00537         str email;
00538         do
00539         {
00540             cout « "Podaj E-Mail: " « endl;
00541             cin » email;
00542             if (!CzyEmail(email))
00543                 cout « "Błędny E-Mail. Spróbuj ponownie." « endl;
00544         } while (!CzyEmail(email));
00545 
00546         for (int i = 0; i < kontakty.size(); i++)
00547         {
00548             if (kontakty[i].email == email)
00549                 PofiltrowaneKontakty.push_back(kontakty[i]);
00550         }
00551         break;
```

```
00552     }
00553     default:
00554         break;
00555     }
00556     return PofiltrowaneKontakty;
00557 }
00558
00559 vector<zadanie> filtrujVecZ(vector<zadanie> zadania, int x)
00560 {
00561     vector<zadanie> PofiltrowaneZadania;
00562
00563     switch (x)
00564     {
00565     case 1:
00566     {
00567         str data;
00568         do
00569         {
00570             cout « "Podaj datę: " « endl;
00571             cin » data;
00572             if (!CzyData(data))
00573                 cout « "Błędna data. Spróbuj ponownie." « endl;
00574         } while (!CzyData(data));
00575
00576         for (int i = 0; i < zadania.size(); i++)
00577         {
00578             if (zadania[i].data == data)
00579                 PofiltrowaneZadania.push_back(zadania[i]);
00580         }
00581
00582         break;
00583     }
00584     case 2:
00585     {
00586         str godzina;
00587         do
00588         {
00589             cout « "Podaj godzinę: " « endl;
00590             cin » godzina;
00591             if (!CzyGodzina(godzina))
00592                 cout « "Błędna godzina. Spróbuj ponownie." « endl;
00593         } while (!CzyGodzina(godzina));
00594
00595         for (int i = 0; i < zadania.size(); i++)
00596         {
00597             if (zadania[i].godzina == godzina)
00598                 PofiltrowaneZadania.push_back(zadania[i]);
00599         }
00600
00601         break;
00602     }
00603     case 3:
00604     {
00605         str typ;
00606         do
00607         {
00608             cout « "Podaj typ: " « endl;
00609             cin » typ;
00610             if (!TylkoLitery(typ))
00611                 cout « "Błędny typ. Spróbuj ponownie." « endl;
00612         } while (!TylkoLitery(typ));
00613
00614         for (int i = 0; i < zadania.size(); i++)
00615         {
00616             if (zadania[i].typ == typ)
00617                 PofiltrowaneZadania.push_back(zadania[i]);
00618         }
00619
00620         break;
00621     }
00622     case 4:
00623     {
00624         str status;
00625         do
00626         {
00627             cout « "Podaj status: (X lub V)" « endl;
00628             cin » status;
00629             if (status.size() > 1 || status != "X" && status != "x" && status != "V" && status != "v")
00630                 cout « "Błędny status. Spróbuj ponownie." « endl;
00631         } while (status.size() > 1 || status != "X" && status != "x" && status != "V" && status !=
      "v");
00632
00633         for (int i = 0; i < zadania.size(); i++)
00634         {
00635             if (zadania[i].status == status)
00636                 PofiltrowaneZadania.push_back(zadania[i]);
00637         }
```

```
00638
00639         break;
00640     }
00641     case 5:
00642     {
00643         str opis;
00644         cout « "Podaj opis: " « endl;
00645         getline(cin, opis);
00646
00647         for (int i = 0; i < zadania.size(); i++)
00648         {
00649             if (zadania[i].opis == opis)
00650                 PofiltrowaneZadania.push_back(zadania[i]);
00651         }
00652         break;
00653     }
00654     default:
00655         break;
00656     }
00657
00658     return PofiltrowaneZadania;
00659 }
00660
00661 void DodajKontakt(vector<kontakt> &vec)
00662 {
00663     str imie, nazwisko, email;
00664     do
00665     {
00666         cout « "Podaj imię: A-anuluj" « endl;
00667         cin » imie;
00668         if (imie == "A"){
00669             show = false;
00670             return;
00671         }
00672         if (!TylkoLitery(imie))
00673             cout « "Błędne imie. Spróbuj ponownie." « endl;
00674     } while (!TylkoLitery(imie));
00675     do
00676     {
00677         cout « "Podaj nazwisko: A-anuluj" « endl;
00678         cin » nazwisko;
00679         if (nazwisko == "A"){
00680             show = false;
00681             return;
00682         }
00683         if (!TylkoLitery(nazwisko))
00684             cout « "Błędne nazwisko. Spróbuj ponownie." « endl;
00685     } while (!TylkoLitery(nazwisko));
00686     do
00687     {
00688         cout « "Podaj email: A-anuluj" « endl;
00689         cin » email;
00690         if (email == "A"){
00691             show = false;
00692             return;
00693         }
00694         if (!CzyEmail(email))
00695             cout « "Błędny E-Mail. Spróbuj ponownie." « endl;
00696     } while (!CzyEmail(email));
00697
00698     kontakt NowaOsoba;
00699     NowaOsoba.imie = imie;
00700     NowaOsoba.nazwisko = nazwisko;
00701     NowaOsoba.email = email;
00702
00703     vec.push_back(NowaOsoba);
00704 }
00705
00706 void UsunKontakt(vector<kontakt> &vec)
00707 {
00708     str x;
00709     int numer = -1;
00710     WypiszVectorK(vec);
00711     do
00712     {
00713         cout « "Który kontakt usunąć? A - anuluj" « endl;
00714         cin » x;
00715         if (x == "A"){
00716             show = false;
00717             return;
00718         }
00719         for (char znak : x)
00720         {
00721             if (!isdigit(znak))
00722                 continue;
00723         }
00724         try
```

```
00725         {
00726             numer = stoi(x);
00727         }
00728         catch (exception &e)
00729         {
00730             cout « "Błąd konwersji. Najprawdopodobniej wejście nie jest liczbą. Spróbuj ponownie." « endl;
00731             numer = -1;
00732             continue;
00733         }
00734         numer = numer - 1;
00735         if (numer < 0 || numer > vec.size() - 1)
00736         {
00737             cout « "Nie ma takiego kontaktu. Spróbuj ponownie." « endl;
00738             continue;
00739         }
00740     } while (numer < 0 || numer > vec.size() - 1);
00741     vec.erase(vec.begin() + numer);
00742 }
00743
00744 void DodajZadanie(vector<zadanie> &vec)
00745 {
00746     str data, godzina, typ, status, opis;
00747     do
00748     {
00749         cout « "Podaj datę (RRRR.MM.DD): A-anuluj" « endl;
00750         cin » data;
00751         if (data == "A"){
00752             show = false;
00753             return;
00754         }
00755         if (CzyData(data) == false)
00756             cout « "Błędna data. Spróbuj ponownie." « endl;
00757     } while (!CzyData(data));
00758     do
00759     {
00760         cout « "Podaj godzinę (GG:MM): A-anuluj" « endl;
00761         cin » godzina;
00762         if (godzina == "A"){
00763             show = false;
00764             return;
00765         }
00766         if (CzyGodzina(godzina) == false)
00767             cout « "Błędna godzina. Spróbuj ponownie." « endl;
00768     } while (!CzyGodzina(godzina));
00769     do
00770     {
00771         cout « "Podaj typ: A-anuluj" « endl;
00772         cin » typ;
00773         if (typ == "A"){
00774             show = false;
00775             return;
00776         }
00777         if (TylkoLitery(typ) == false)
00778             cout « "Błędny typ. Spróbuj ponownie." « endl;
00779     } while (!TylkoLitery(typ));
00780     do
00781     {
00782         cout « "Podaj status(X - niewykonane, V - wykonane): A-anuluj" « endl;
00783         cin » status;
00784         if (status == "A"){
00785             show = false;
00786             return;
00787         }
00788         if (status != "X" && status != "V" && status != "x" && status != "v")
00789             cout « "Błędny status. Spróbuj ponownie." « endl;
00790     } while (status != "X" && status != "V" && status != "x" && status != "v");
00791     if (status == "x")
00792         status = "X";
00793     else if (status == "v")
00794         status = "V";
00795     cout « "Podaj opis: A-anuluj" « endl;
00796     cin.clear();
00797     cin.sync();
00798     getline(cin, opis);
00799
00800     if (opis == "A"){
00801             show = false;
00802             return;
00803         }
00804
00805     zadanie NoweZadanie;
00806     NoweZadanie.data = data;
00807     NoweZadanie.godzina = godzina;
00808     NoweZadanie.typ = typ;
00809     NoweZadanie.status = status;
00810     NoweZadanie.opis = opis;
```

```
00811
00812     vec.push_back(NoweZadanie);
00813 }
00814
00815 void UsunZadanie(vector<zadanie> &vec)
00816 {
00817     str x;
00818     int numer = -1;
00819     WypiszVectorZ(vec);
00820     do
00821     {
00822         cout « "Które zadanie usunąć? A-anuluj" « endl;
00823         cin » x;
00824         if (x == "A"){
00825             show = false;
00826             return;
00827         }
00828         for (char znak : x)
00829         {
00830             if (!isdigit(znak))
00831                 continue;
00832         }
00833         try
00834         {
00835             numer = stoi(x);
00836         }
00837         catch (exception &e)
00838         {
00839             cout « "Błąd konwersji. Najprawdopodobniej wejście nie jest liczbą. Spróbuj ponownie." «
      endl;
00840             numer = -1;
00841             continue;
00842         }
00843         numer = numer - 1;
00844         if (numer < 0 || numer > vec.size() - 1)
00845             cout « "Nie ma takiego zadania. Spróbuj ponownie." « endl;
00846     } while (numer < 0 || numer > vec.size() - 1);
00847     vec.erase(vec.begin() + numer);
00848 }
00849
00850 void ZmienStatus(vector<zadanie> &vec)
00851 {
00852     str x;
00853     int numer = -1;
00854     WypiszVectorZ(vec);
00855     do
00856     {
00857         cout « "Status którego zadania chcesz zmienić? A-anuluj" « endl;
00858         cin » x;
00859         if (x == "A"){
00860             show = false;
00861             return;
00862         }
00863         for (char znak : x)
00864         {
00865             if (!isdigit(znak))
00866                 continue;
00867         }
00868         try
00869         {
00870             numer = stoi(x);
00871         }
00872         catch (exception &e)
00873         {
00874             cout « "Błąd konwersji. Najprawdopodobniej wejście nie jest liczbą. Spróbuj ponownie." «
      endl;
00875             numer = -1;
00876             continue;
00877         }
00878         if (numer < 1 || numer > vec.size())
00879             cout « "Nie ma takiego zadania. Spróbuj ponownie." « endl;
00880     } while (numer < 1 || numer > vec.size());
00881
00882     numer = numer - 1;
00883
00884     string zamiennik;
00885
00886     do
00887     {
00888         cout « "Wprowadź status V(wykonane) lub X(niewykonane): " « endl;
00889         cin » zamiennik;
00890     } while (zamiennik != "V" && zamiennik != "X" && zamiennik != "v" && zamiennik != "x");
00891
00892     if (zamiennik == "v")
00893         zamiennik = "V";
00894     if (zamiennik == "x")
00895         zamiennik = "X";
```

```
00896
00897     vec[numer].status = zamiennik;
00898 }
00899
00900 vector<kontakt> CzytajKontakty(str nazwa_pliku_kontakty)
00901 {
00902     fstream wejscie(nazwa_pliku_kontakty);
00903
00904     vector<kontakt> kontakty;
00905
00906     if (wejscie.good())
00907     {
00908         str imie, nazwisko, email;
00909
00910         while (wejscie » imie » nazwisko » email)
00911         {
00912             kontakt osoba;
00913             osoba.imie = imie;
00914             osoba.nazwisko = nazwisko;
00915             osoba.email = email;
00916             kontakty.push_back(osoba);
00917         }
00918     }
00919     wejscie.close();
00920     return kontakty;
00921 }
00922
00923 vector<zadanie> CzytajZadania(str nazwa_pliku_zadania)
00924 {
00925     fstream wejscie(nazwa_pliku_zadania);
00926
00927     vector<zadanie> zadania;
00928
00929     str data, godzina, typ, status, opis;
00930
00931     if (wejscie.good())
00932     {
00933         while (wejscie » data » godzina » typ » status)
00934         {
00935             getline(wejscie, opis);
00936             opis.erase(0, 1);
00937             zadanie zad;
00938             zad.data = data;
00939             zad.godzina = godzina;
00940             zad.typ = typ;
00941             zad.status = status;
00942             zad.opis = opis;
00943             zadania.push_back(zad);
00944         }
00945     }
00946
00947     wejscie.close();
00948     return zadania;
00949 }
00950
00951 void ZapiszWPlikach(str nazwa_pliku_kontakty, str nazwa_pliku_zadania, vector<kontakt> &vecK,
    vector<zadanie> &vecZ)
00952 {
00953     ofstream wyjscieKontakty(nazwa_pliku_kontakty, ios::out | ios::trunc);
00954     ofstream wyjscieZadania(nazwa_pliku_zadania, ios::out | ios::trunc);
00955
00956     for (auto elem : vecK)
00957     {
00958         wyjscieKontakty « elem.imie « " " « elem.nazwisko « " " « elem.email « endl;
00959     }
00960
00961     for (auto elem : vecZ)
00962     {
00963         wyjscieZadania « elem.data « " " « elem.godzina « " " « elem.typ « " " « elem.status « " " «
    elem.opis « endl;
00964     }
00965
00966     wyjscieKontakty.close();
00967     wyjscieZadania.close();
00968 }
00969
00970 void Zapytanie()
00971 {
00972     int x = 0;
00973     do
00974     {
00975         cout « "1.Wróć do menu \n 2.Zakończ" « endl;
00976         cin » x;
00977         if(cin.fail()){
00978             cin.clear();
00979             cin.ignore(std::numeric_limits<int>::max(), '\n');
00980             x = 0;
```

```
00981             cout«"Błąd konwersji. Wejście najprawdopodobniej nie jest liczbą. Spróbuj ponownie."«endl;
00982         }
00983         else if (x != 1 && x != 2) cout « "Nie ma takiej opcji. Spróbuj ponownie." « endl;
00984     } while (x != 1 && x != 2);
00985     if (x == 1)
00986         system("cls");
00987     if (x == 2)
00988         DzialanieProgramu = false;
00989 }
00990
00991 int WedlugCzegoK(string flag)
00992 {
00993     int x = 0;
00994     do
00995     {
00996         do
00997         {
00998             cout « flag « " kontakty według: \n 1.Imienia \n 2.Nazwiska \n 3.E-Maila" « endl;
00999             cin » x;
01000             if (cin.fail())
01001                 cout « "Błąd konwersji. Najprawdopodobniej wejście nie jest liczbą. Spróbuj ponownie."
      « endl;
01002         } while (cin.fail());
01003         if (x != 1 && x != 2 && x != 3)
01004         {
01005             x = 0;
01006             cout « "Nie ma takiej opcji. Spróbuj ponownie." « endl;
01007         }
01008     } while (x < 1 || x > 3);
01009     return x;
01010 }
01011
01012 int WedlugCzegoZ(string flag)
01013 {
01014     int x = 0;
01015     do
01016     {
01017         do
01018         {
01019             cout « flag « " zadania według: \n 1.Daty \n 2.Godziny \n 3.Typu \n 4.Statusu \n 5.Opisu"
      « endl;
01020             cin » x;
01021             if (cin.fail())
01022                 cout « "Błąd konwersji. Najprawdopodobniej wejście nie jest liczbą. Spróbuj ponownie."
      « endl;
01023         } while (cin.fail());
01024
01025         if (x != 1 && x != 2 && x != 3 && x != 4 && x != 5)
01026         {
01027             x = 0;
01028             cout « "Nie ma takiej opcji. Spróbuj ponownie." « endl;
01029         }
01030     } while (x < 1 || x > 5);
01031     return x;
01032 }
01033
01034 bool CzySortowac()
01035 {
01036     string wejscie;
01037     char x;
01038     do
01039     {
01040         do
01041         {
01042             cout « "Czy sortować? T/N" « endl;
01043             cin » wejscie;
01044             if (cin.fail() || wejscie.size() > 1)
01045                 cout « "Błąd konwersji. Wejście najprawdopodobniej nie jest znakiem. Spróbuj
      ponownie." « endl;
01046         } while (cin.fail() || wejscie.size() > 1);
01047         x = wejscie[0];
01048         if (x != 'T' && x != 'N' && x != 't' && x != 'n')
01049             cout « "Nie ma takiej opcji. Spróbuj ponownie." « endl;
01050
01051     } while (x != 'T' && x != 'N' && x != 't' && x != 'n');
01052
01053     if (x == 'T' || x == 't')
01054         return true;
01055     else
01056         return false;
01057 }
01058
01059 bool CzyFiltrowac()
01060 {
01061     string wejscie;
01062     char x;
01063     do
```

```
01064     {
01065         do
01066         {
01067             cout « "Czy filtrować? T/N" « endl;
01068             cin » wejscie;
01069             if (cin.fail() || wejscie.size() > 1)
01070                 cout « "Błąd konwersji. Wejście najprawdopodobniej nie jest znakiem. Spróbuj
      ponownie." « endl;
01071         } while (cin.fail() || wejscie.size() > 1);
01072         x = wejscie[0];
01073         if (x != 'T' && x != 'N' && x != 't' && x != 'n')
01074             cout « "Nie ma takiej opcji. Spróbuj ponownie." « endl;
01075     } while (x != 'T' && x != 'N' && x != 't' && x != 'n');
01076
01077     if (x == 'T' || x == 't')
01078         return true;
01079     else
01080         return false;
01081 }
01082
01083 void Start(vector<kontakt> &kontakty, vector<zadanie> &zadania)
01084 {
01085     int x = 0;
01086     do
01087     {
01088         cout « "Witaj w Organizerze! Co chciał(a)byś zrobić?" « endl;
01089         cout « "1.Dodaj kontakt \n 2.Usuń kontakt \n 3.Dodaj zadanie \n 4.Usuń zadanie \n 5.Wyświetl
      kontakty \n 6.Wyświetl zadania \n 7.Zmień status zadania \n 8.Zakończ działanie programu" « endl;
01090         cin » x;
01091         if (cin.fail())
01092         {
01093             x = 0;
01094             cout « "Błąd konwersji. Najprawdopodobniej wejście nie jest liczbą. Spróbuj ponownie." «
      endl;
01095             cin.clear();
01096             str ignore;
01097             cin » ignore;
01098             continue;
01099         }
01100         else if (x != 1 && x != 2 && x != 3 && x != 4 && x != 5 && x != 6 && x != 7 && x != 8)
01101         {
01102             x = 0;
01103             cout « "Nie ma takiej opcji. Spróbuj ponownie." « endl;
01104         }
01105     } while (x < 1 || x > 8);
01106
01107     switch (x)
01108     {
01109     case 1:
01110         DodajKontakt(kontakty);
01111         if(show){
01112             cout « endl
01113              « "Podsumowanie:" « endl;
01114             WypiszVectorK(kontakty);
01115         }
01116         show = true;
01117         Zapytanie();
01118         break;
01119     case 2:
01120         UsunKontakt(kontakty);
01121         if(show){
01122             cout « endl
01123              « "Podsumowanie:" « endl;
01124             WypiszVectorK(kontakty);
01125         }
01126         show = true;
01127         Zapytanie();
01128         break;
01129     case 3:
01130         DodajZadanie(zadania);
01131         if(show){
01132             cout « endl
01133              « "Podsumowanie:" « endl;
01134             WypiszVectorZ(zadania);
01135         }
01136         show = true;
01137         Zapytanie();
01138         break;
01139     case 4:
01140         UsunZadanie(zadania);
01141         if(show){
01142             cout « endl
01143              « "Podsumowanie:" « endl;
01144             WypiszVectorZ(zadania);
01145         }
01146         show = true;
01147         Zapytanie();
```

```
01148         break;
01149     case 5:
01150     {
01151         vector<kontakt> PosortowaneKontakty;
01152         vector<kontakt> PofiltrowaneKontakty;
01153 
01154         if (CzyFiltrowac())
01155         {
01156             int kryterium = WedlugCzegoK("Pofiltrować");
01157             PofiltrowaneKontakty = filtrujVecK(kontakty, kryterium);
01158             if (CzySortowac())
01159             {
01160                 int kryterium = WedlugCzegoK("Posortować");
01161                 PosortowaneKontakty = sortujVecK(PofiltrowaneKontakty, kryterium);
01162                 WypiszVectorK(PosortowaneKontakty);
01163             }
01164             else
01165             {
01166                 WypiszVectorK(PofiltrowaneKontakty);
01167             }
01168         }
01169         else
01170         {
01171             if (CzySortowac())
01172             {
01173                 int kryterium = WedlugCzegoK("Posortować");
01174                 PosortowaneKontakty = sortujVecK(kontakty, kryterium);
01175                 WypiszVectorK(PosortowaneKontakty);
01176             }
01177             else
01178                 WypiszVectorK(kontakty);
01179         }
01180         Zapytanie();
01181         break;
01182     }
01183     case 6:
01184     {
01185         vector<zadanie> PosortowaneZadania;
01186         vector<zadanie> PofiltrowaneZadania;
01187 
01188         if (CzyFiltrowac())
01189         {
01190             int kryterium = WedlugCzegoZ("Pofiltrować");
01191             PofiltrowaneZadania = filtrujVecZ(zadania, kryterium);
01192             if (CzySortowac())
01193             {
01194                 int kryterium = WedlugCzegoZ("Posortować");
01195                 PosortowaneZadania = sortujVecZ(PofiltrowaneZadania, kryterium);
01196                 WypiszVectorZ(PosortowaneZadania);
01197             }
01198             else
01199             {
01200                 WypiszVectorZ(PofiltrowaneZadania);
01201             }
01202         }
01203         else
01204         {
01205             if (CzySortowac())
01206             {
01207                 int kryterium = WedlugCzegoZ("Posortować");
01208                 PosortowaneZadania = sortujVecZ(zadania, kryterium);
01209                 WypiszVectorZ(PosortowaneZadania);
01210             }
01211             else
01212                 WypiszVectorZ(zadania);
01213         }
01214         Zapytanie();
01215         break;
01216     }
01217     case 7:
01218         ZmienStatus(zadania);
01219         if(show){
01220             cout « endl
01221              « "Podsumowanie:" « endl;
01222             WypiszVectorZ(zadania);
01223         }
01224         show = true;
01225         Zapytanie();
01226         break;
01227     case 8:
01228         DzialanieProgramu = false;
01229         break;
01230 
01231     default:
01232         break;
01233     }
01234 }
```

```
01235
01236 bool TylkoLitery(string nazwa)
01237 {
01238     for (char znak : nazwa)
01239     {
01240         if (!isalpha(znak))
01241         {
01242             return false;
01243         }
01244     }
01245     return true;
01246 }
01247
01248 bool CzyEmail(string email)
01249 {
01250     regex wzorzec("[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\\.[a-zA-Z]{2,}");
01251     if (regex_match(email, wzorzec))
01252         return true;
01253     else
01254         return false;
01255 }
01256
01257 bool CzyData(string data)
01258 {
01259     regex wzorzec("[0-9]{4}.[0-9]{2}.[0-9]{2}");
01260     if (regex_match(data, wzorzec))
01261     {
01262         if (data[5] == '0' && data[6] == '0')
01263             return false;
01264         if (data[8] == '0' && data[9] == '0')
01265             return false;
01266         return true;
01267     }
01268     else
01269         return false;
01270 }
01271
01272 bool CzyGodzina(string godzina)
01273 {
01274     regex wzorzec("[0-2]+[0-9]:[0-5]+[0-9]");
01275     if (regex_match(godzina, wzorzec))
01276     {
01277         regex wzorzec1("[4-9]");
01278         string s(1, godzina[1]);
01279         if (godzina[0] == '2' && regex_match(s, wzorzec1))
01280             return false;
01281         return true;
01282     }
01283     else
01284         return false;
01285 }
```