|  |  |
| --- | --- |
| **Projekt podstawy programowania 2**  Wydział Elektrotechniki Automatyki i Informatyki  Politechnika Świętokrzyska | |
| Studia: **Stacjonarne I stopnia** | Kierunek:  **Informatyka** |
| Data wykonania: **3.06.2016** | Grupa: **1ID14A** |
| Ocena: | 1. Lisowski Wojciech 2. Łękawska Kinga |
| Temat projektu: | |
| **Baza kontaktów** | |

1. **Temat projektu**

Tematem projektu jest program „Baza kontaktów” stworzona z wykorzystaniem list. Baza posiada takie funkcje jak: dodawanie, usuwanie i edycja kontaktów złożonej z pól: ID, nazwisko, imię, adres (miasto, ulica, nr domu/mieszkania, kod pocztowy, poczta), telefon i e-mail. Kontakty można sortować po dowolnym polu rosnąco lub malejąco.

1. **Realizacja projektu**
   1. Język programowania i środowisko:

Projekt został napisany w języku C w środowisku Code::Blocks na systemie Windows.

* 1. Załączone pliki nagłówkowe:
* stdio.h - obsługa standardowego wejścia/wyjścia.
* stdlib.h - standardowe narzędzia języka C
* string.h - obsługa ciągów znaków
* ctype.h - obsługa operacji na znakach
  1. Struktura programu:

Projekt został podzielony na pliki źródłowe i nagłówkowe:

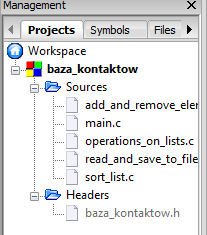
* main.c – plik źródłowy zawiera główne menu i wywoływane są w nim wszystkie funkcję z innych plików.
* add\_and\_remove\_elements.c – plik źródłowy zawiera funkcję dodające i usuwające elementy z listy.
* operations\_on\_lists.c – plik źródłowy zawiera funkcję operacji na listach np. wyświetlanie.
* read\_and\_save\_to\_file.c – plik źródłowy zawiera funkcję operacji na plikach.
* sort\_list.c – plik źródłowy zawiera funkcje sortujące listę.
* baza\_kontaktow.h – plik nagłówkowy zawiera nagłówki funkcji ze wszystkich plików programu.

1. **Kompilacja programu**

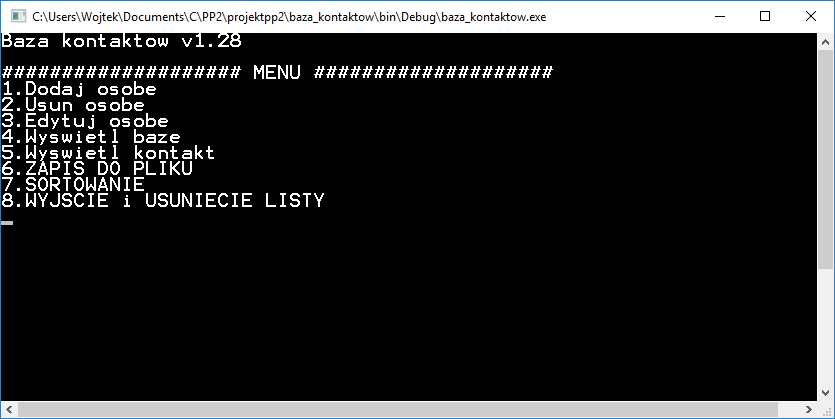
Kompilację należy przeprowadzić w programie Code::Blocks.

* 1. Proces kompilacji:
* Uruchomienie pliku baza\_kontaktow.cbp za pomocą programu Code::Blocks.

Po poprawnym otwarciu struktura programu powinna wyglądać tak:



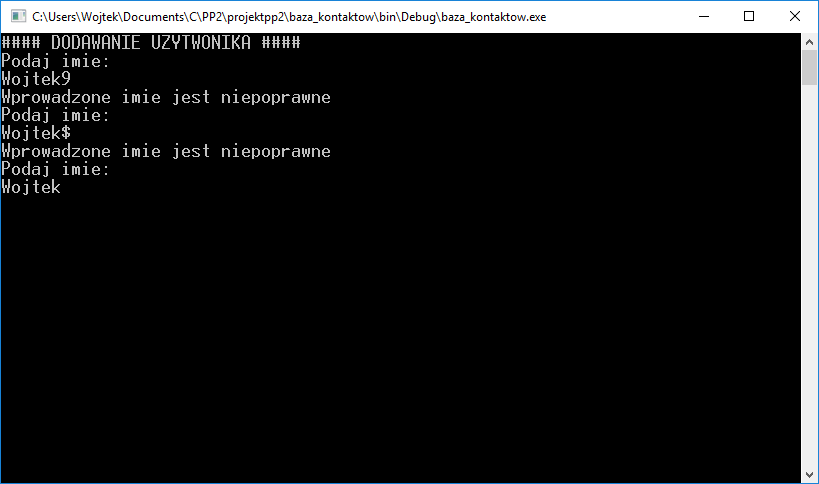
* Następnie w programie Code::Blocks przechodzimy do zakładki „Build” i klikamy opcję „Build and run” lub klikamy klawisz F9.
* Po prawidłowej kompilacji powinno wyświetlić się główne menu programu:



1. **Przykładowe działanie programu**
   1. Dodawanie kontaktu

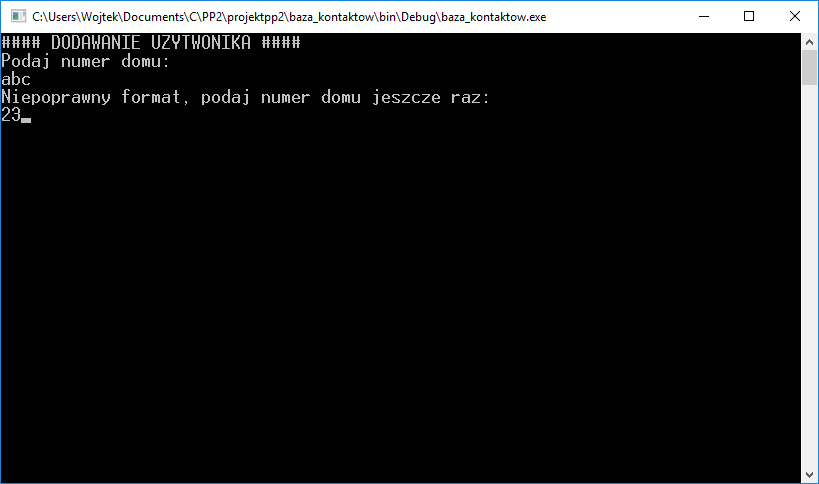
Aby dodać kontakt wybieramy opcję nr 1 w menu głównym.

Wprowadzane dane są sprawdzane pod kątem poprawności np. w imieniu nie można podać liczby ani znaku specjalnego, przykład:

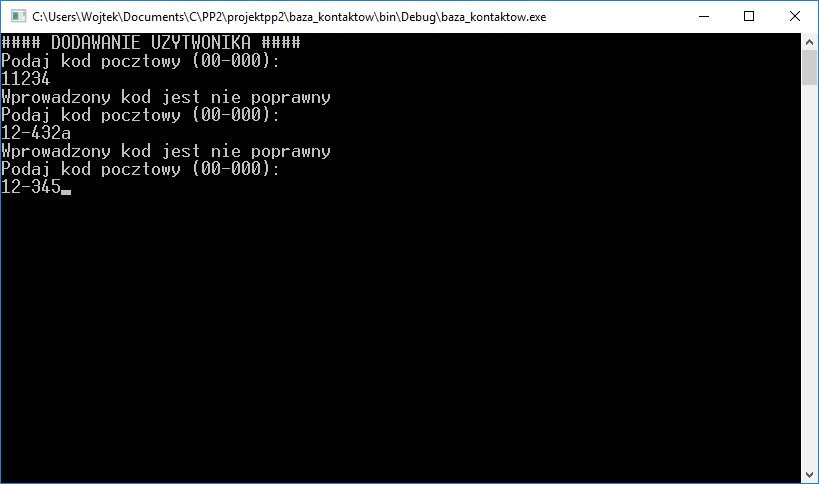


Taka walidacja działa tak samo dla pól: nazwisko, miasto, ulica i poczta.

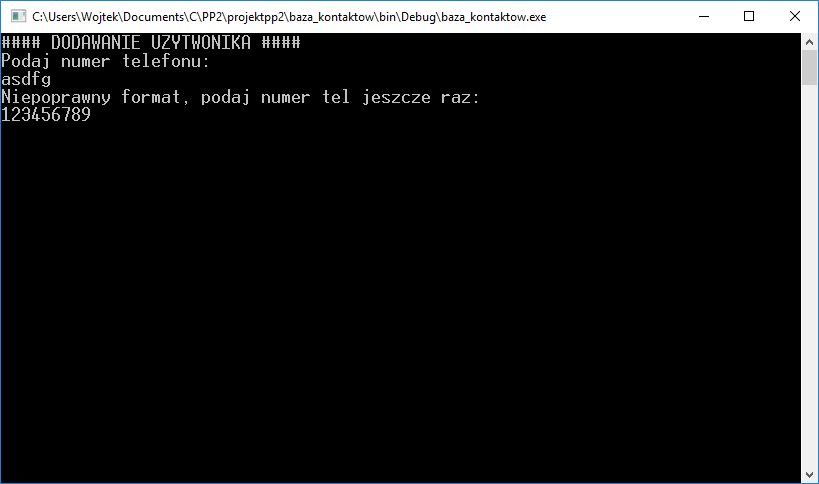
Dla pola „numer domu” nie można wprowadzać znaków specjalnych ani liter:



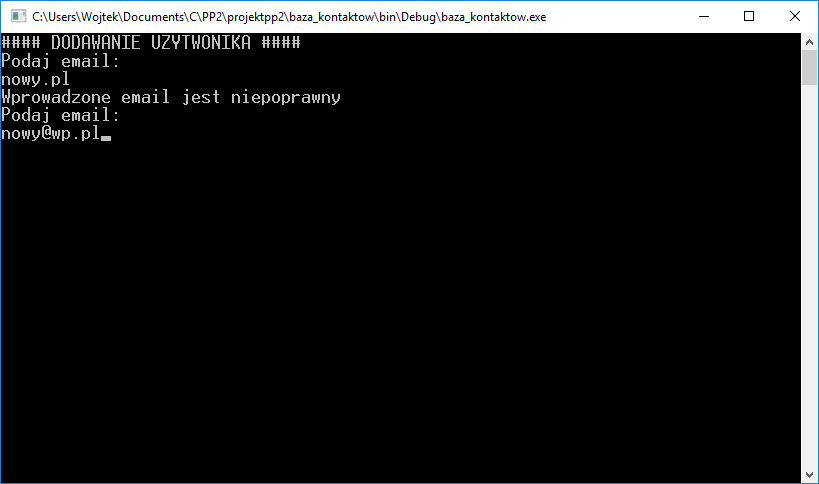
Pole „kod pocztowy” także posiada walidację danych, nie można wprowadzić litery ani znaku specjalnego i wprowadzany kod musi być w formacie „00-000”:



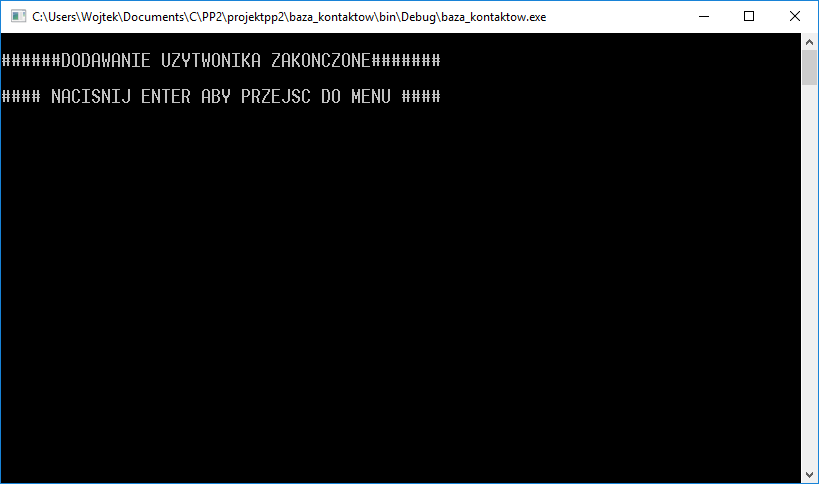
W polu „numer telefonu” można podać kilka numerów telefonów, numery są sprawdzane pod kątem poprawności np.:



Ostatnim polem jest „adres e-mail’. Adresów e-mail można podać kilka. Każdy email jest sprawdzany czy występuje w nim @ i . np.:

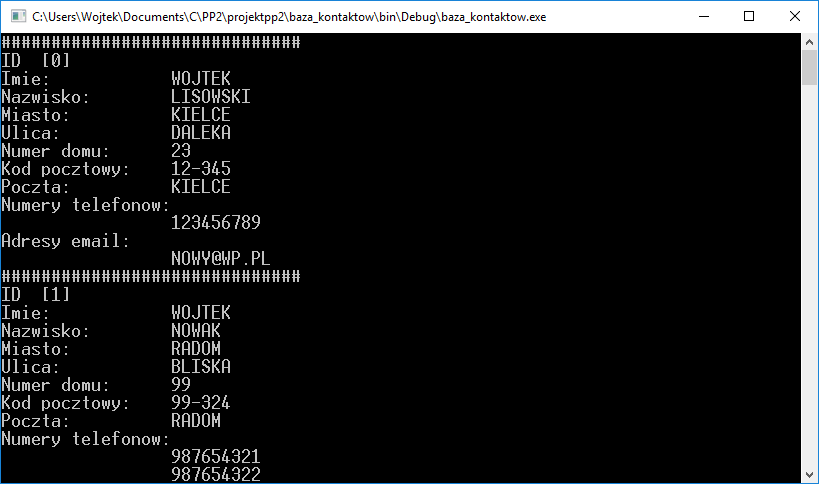


Na koniec dostajemy komunikat:



* 1. Wyświetlanie bazy kontaktów.

Aby wyświetlić dodanych lub wczytanych z pliku użytkowników wybieramy opcję nr „4” z menu głównego, np.:

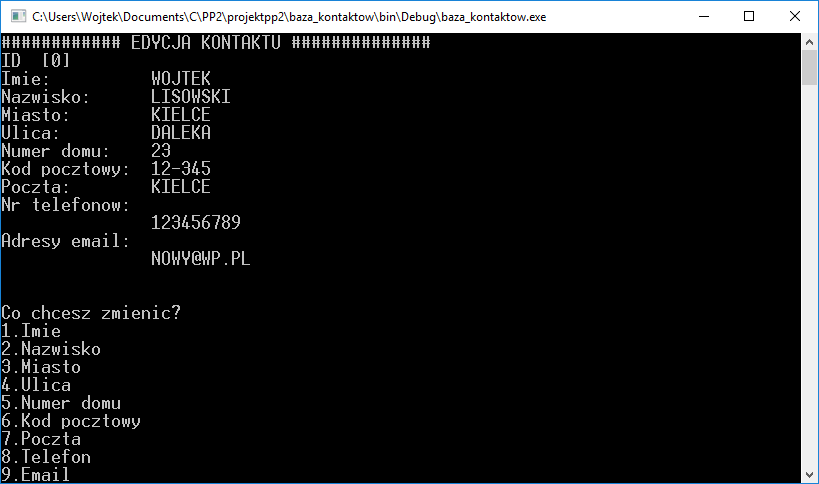


Aby przejść do menu głównego klikamy ENTER

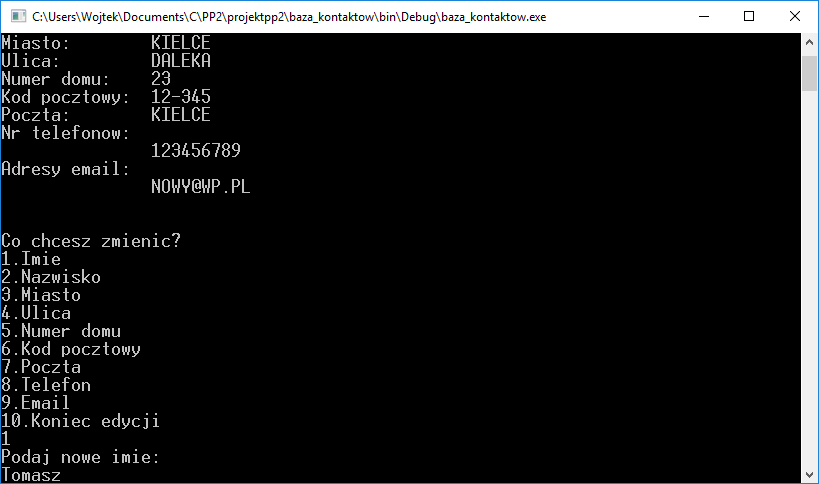
* 1. Edycja kontaktu.

Jeśli chcemy edytować jakiegoś użytkownika wymieramy opcję nr „3” z menu głównego, następne podajemy ID użytkownika, którego chcemy edytować. To pole też posiada walidację danych.

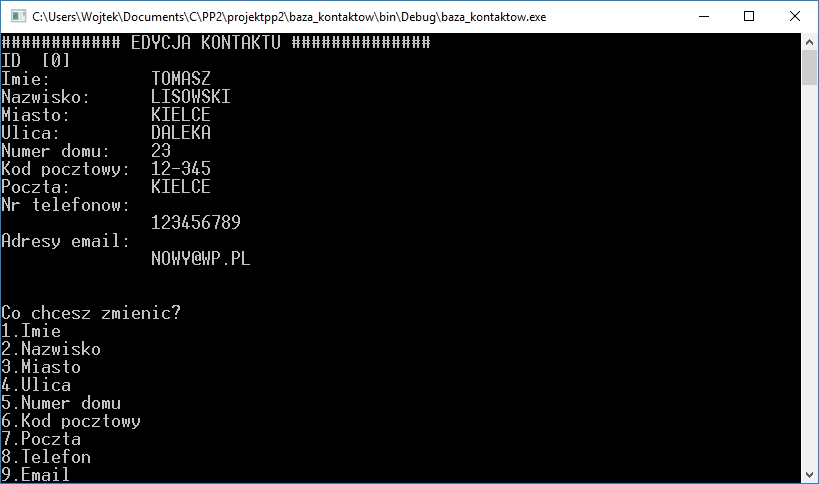
Przykład dla ID 0:



W tym menu możemy edytować każde pole z wyjątkiem ID. Przykładowe działanie dla pola „imię”:



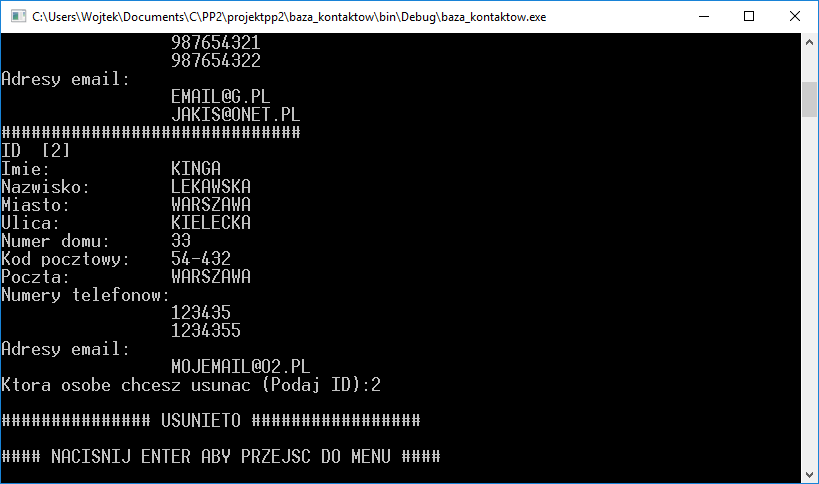
Podaliśmy nowe imię „Tomasz”.



Po potwierdzeniu klawiszem ENTER imię zostało zmienione z „WOJTEK” na „TOMASZ”. Resztę pól edytujemy analogicznie. Wszystkie pola posiadają taką samą walidację wprowadzanych danych jak przy dodawaniu nowego kontaktu.

* 1. Usuwanie kontaktu.

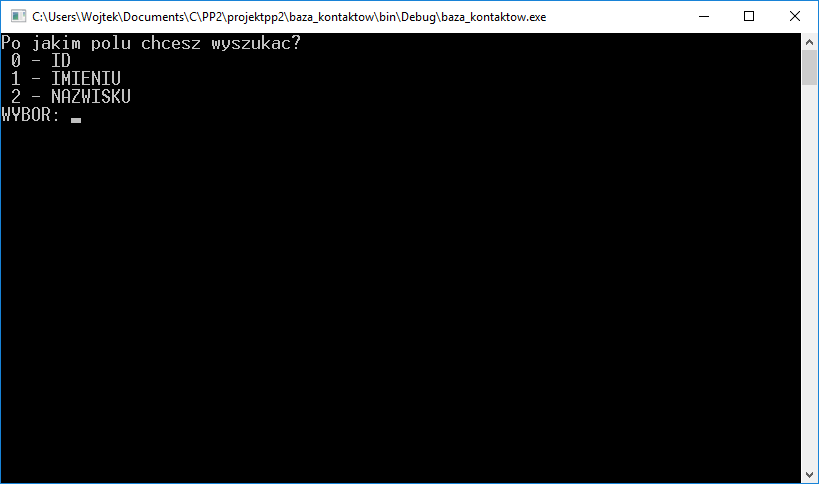
Aby usunąć kontakt trzeba wybrać opcję nr „2” w menu głównym. Następnie wybrać ID kontaktu, którego chcemy usunąć np.:



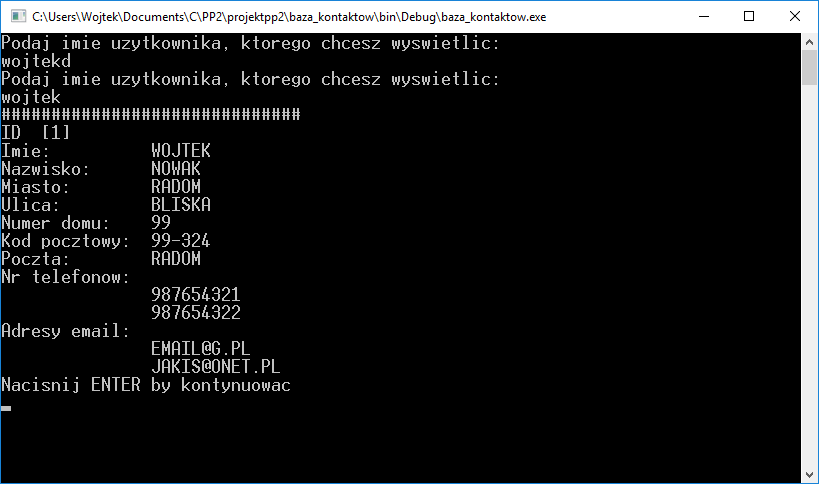
* 1. Wyszukiwanie kontaktu

Aby wyszukać kontakt wybieramy opcję nr „5” w menu głównym.

Szukać można po 3 polach(ID, imię, nazwisko). W tym przykładzie będziemy szukać po imieniu, więc wybieramy opcję nr 1.



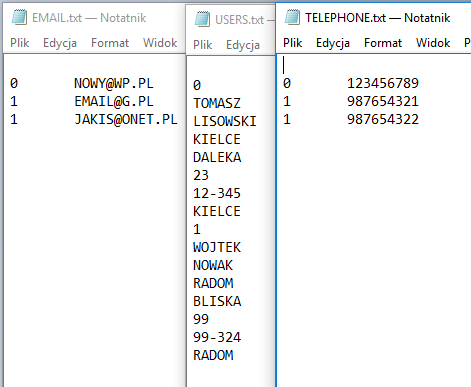
Następnie podajemy imię szukanego kontaktu. Jeśli nie znajdzie takiego kontaktu to zostaniemy poproszeni, aby podać imię jeszcze raz:



Jeśli znalazłoby kilka osób o takim samym imieniu to te kontakty zostałby wyświetlone.

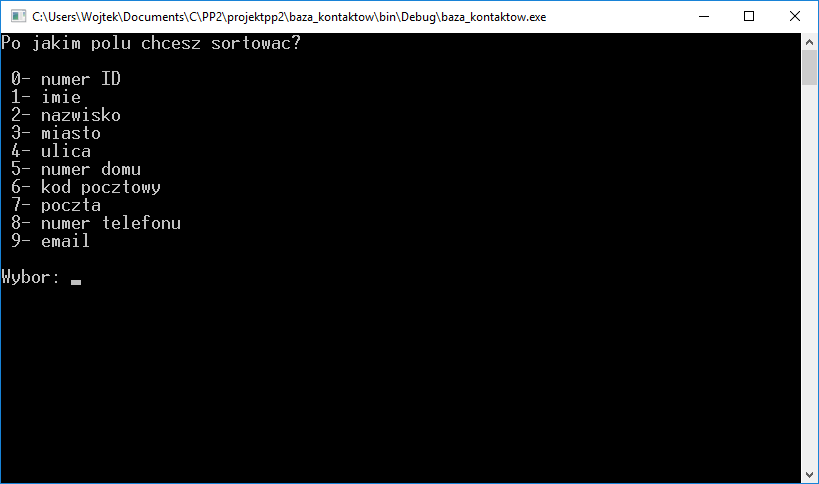
* 1. Zapis do pliku

Aby zapisać kontakty do pliku należy wybrać opcję nr, „6”. Jeśli lista jest pusta to dostaniemy komunikat, że operacją się nie powiodła i żadne pliki nie zostaną utworzone. Natomiast jak lista nie jest pusta to dane zostaną zapisane w plikach. Przykładowe dane zapisane w plikach:

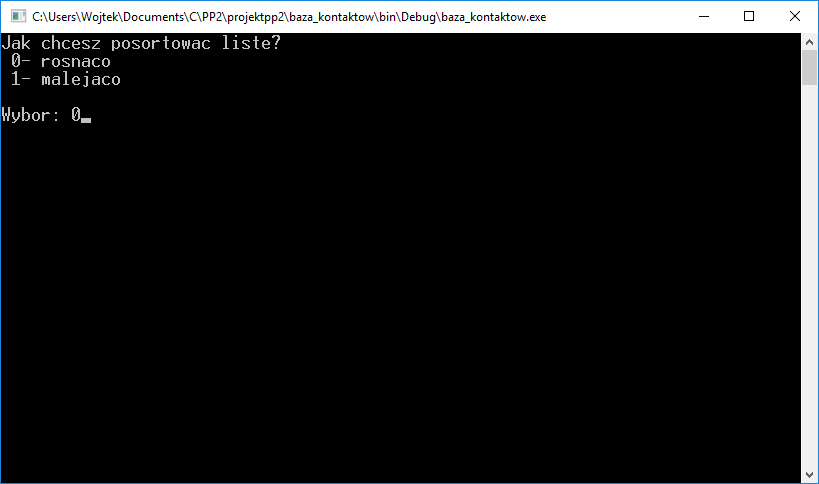


* 1. Sortowanie

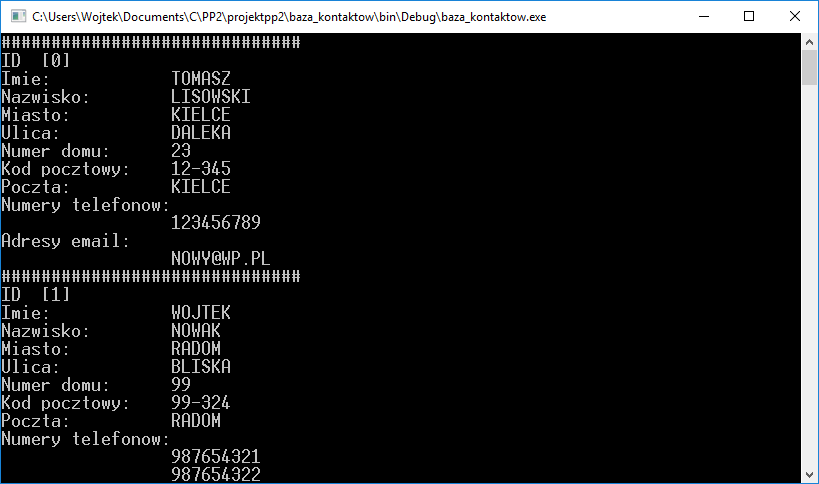
Aby posortować listę należy wybrać opcję nr „7”. W następnym menu wybieramy, po jakim polu chcemy sortować.



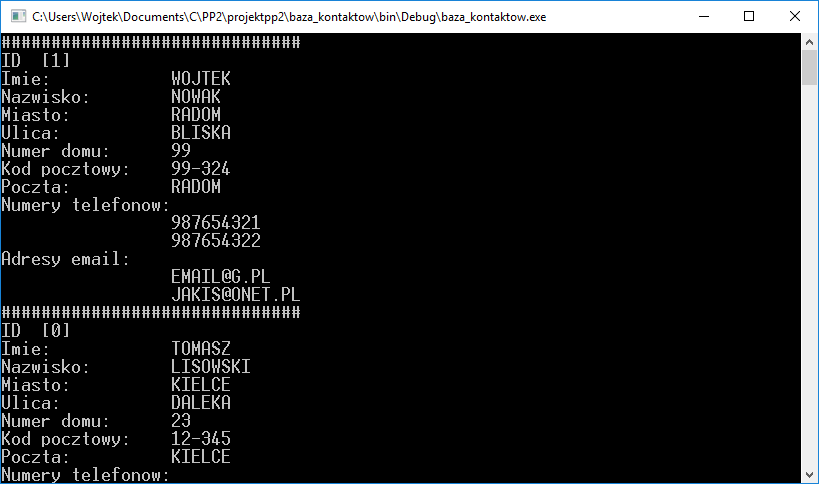
Dla przykładu posortujemy po imieniu, czyli wybieramy opcję nr „1”. Następnie wybieramy kierunek sortowania, w przykładzie posortujemy rosnąco, czyli opcja „0”.



Wynik sortowania wyświetlamy opcją nr „4” w menu głównym.



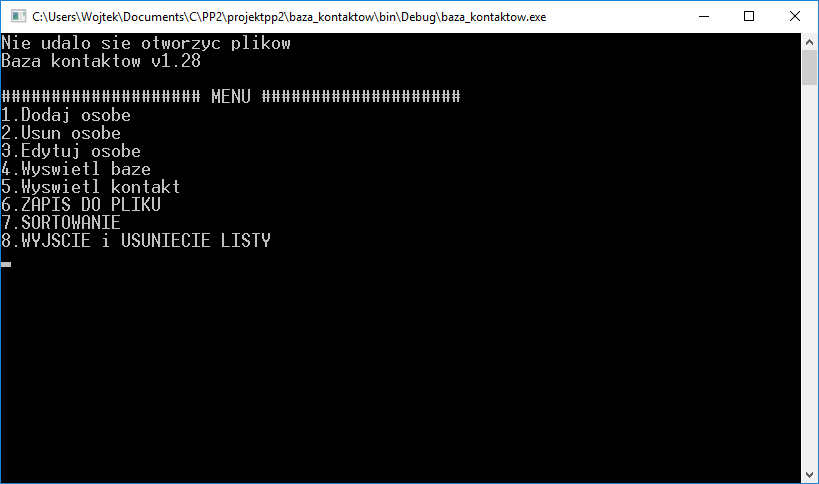
„T” jest przed „W” w alfabecie, więc sortowanie działa poprawnie. Aby się upewnić, że sortowanie działa poprawie posortujemy jeszcze raz po polu „imię”, ale malejąco.



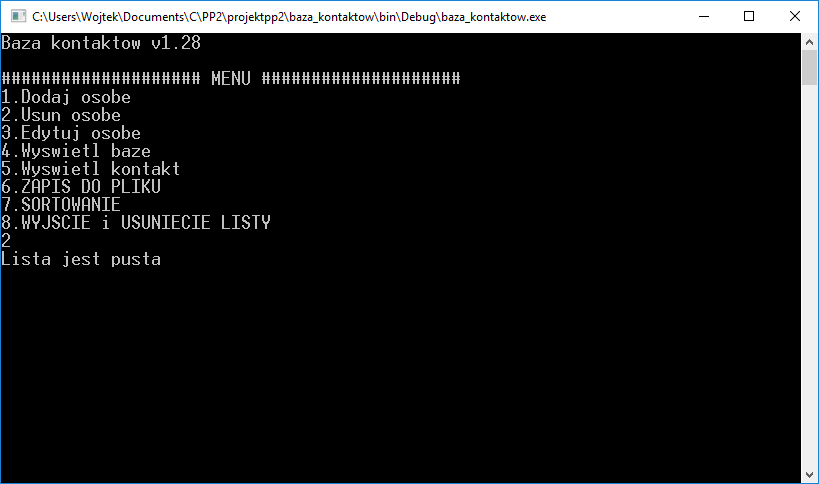
Jak widać sortowanie działa poprawnie, ponieważ kontakty zostały zamienione miejscami.

* 1. Działanie programu, gdy nie udało się odczytać plików

Gdy nie uda się odczytać plików program wyświetli stosowny komunikat:



W takim przypadku dostępne będą tylko dwie opcje (1- Dodawanie osoby i 8-wyjście z programu) reszta nie zadziała, a program wyświetli komunikat „Lista jest pusta”:



1. Opis użytych algorytmów i najważniejszych fragmentów implementacji.

Program napisany jest na liście jednokierunkowej bez wartowników. Implementacja listy jest wzorowana na podstawie instrukcji laboratoryjnej nr 5 przygotowanej przez dr inż. Arkadiusza Chrobota i mgr inż. Pawła Piętę.

Kilka ważnych elementów programu:

**Dodawanie numeru telefonu do listy:**

41 **struct** telephone\_nr \* InsertTelNum(**struct** telephone\_nr \*FrontTelList)

42 {

43 **struct** telephone\_nr \*new\_node = (**struct** telephone\_nr \*)malloc (**sizeof** (**struct** telephone\_nr));

44 **struct** telephone\_nr \*wsk=FrontTelList;

45 **unsigned long int** NrTel;

46 **while** (wsk->next != NULL)

47 {

48 wsk = wsk->next;

49 }

50 **if** (NULL != new\_node)

51 {

52 system("cls");

53 puts("#### DODAWANIE UZYTWONIKA ####");

54 puts("Podaj numer telefonu:");

55 wsk->next = new\_node;

56 **while**(scanf("%lu", &NrTel) != 1 || NrTel<1 || NrTel>999999999)

57 {

58 puts("Niepoprawny format, podaj numer tel jeszcze raz:");

59 fflush(stdin);

60 }

61 new\_node->ID\_tel=NumID;

62 new\_node->tel\_nr=NrTel;

63 new\_node->next=NULL;

64 }

65 **return** FrontTelList;

66 }

Funkcja ta dodaję na koniec listy nowy numer telefonu.

**Szczegółowy opis:**

* W linii 43 przydzielany jest nowy obszar pamięci
* W linii 44 wskaźnikowi roboczemu przypisujemy adres pierwszego elementu listy
* W linii 45 deklarujemy zmienna, która będzie przechowywać numer telefonu podany przez użytkownika
* W liniach 46-49 przesuwamy roboczy wskaźnik na koniec listy
* W linii 50 sprawdzamy czy udało się przydzielić pamięć na nowy element
* W linii 52 czyścimy ekran
* W linii 55 dołączamy nowy element na koniec listy
* W linii 56 podajemy numer telefonu, warunek sprawdza czy podany nr nie jest ciągiem znaków i czy jest z przedziału od 1 do 999999999 jeśli któryś z warunków się spełni to zostaną wykonane linie 58 i 59 po czym użytkownik będzie musiał podać nr jeszcze raz
* W linie 58 wyświetlany jest komunikat o błędzie
* W linii 59 czyszczony jest bufor
* W linii 61 zapisywane jest ID numeru telefonu
* W linii 62 zapisywany jest numer telefonu do listy
* W linii 63 wskaźnikowi na następny element przypisywana jest wartość NULL
* W linii 65 funkcja zwraca wskaźnik na pierwszy element listy

Funkcje dodające nowe adresy e-mail i nowych użytkowników do list działają na tej samej zasadzie tylko różnią się walidacją danych. Kilka funkcji sprawdzających poprawność danych zostanie omówiona poniżej.

**Funkcja sprawdzająca poprawność kodu pocztowego:**

359 **int** check\_postal\_code(**char** array[])

360 {

361 **if**(strlen(array)>6) **return** 0;

362 **if**(isdigit(array[0])==0) **return** 0;

363 **if**(isdigit(array[1])==0) **return** 0;

364 **if**((array[2])!='-') **return** 0;

365 **if**(isdigit(array[3])==0) **return** 0;

366 **if**(isdigit(array[4])==0) **return** 0;

367 **if**(isdigit(array[5])==0) **return** 0;

368 **else**

369 **return** 1;

370 }

* W linii 359 funkcja jako parametr przyjmuję tablicę znaków
* W linii 361 sprawdzana jest długość ciągu jeśli jest dłuższa od 6 to funkcja zwróci wartość 0
* W liniach 362-363 sprawdzamy czy w indexie 0 i 1 nie ma liczby
* W linii 364 sprawdzamy czy w tablicy o indexie 2 nie ma znaku „-”
* W liniach 365-367 sprawdzany czy nie ma liczb w tablicy o indeksach 3-5
* W linii 369 funkcja zwróci wartość 1 jeśli żaden z poprzednich warunków nie został spełniony

**Funkcja sprawdzająca poprawność podanego ciągu**

**int** check\_string(**char** array[])

379 {

380 **int** i,k=1;

381 **for**(i=0; i<strlen(array); ++i)

382 {

383 **if**(isalnum(array[i])!=0 && isdigit(array[i])==0)

384 {

385 }

386 **else**

387 {

388 k=0;

389 }

390 }

391 **if**(k==0)**return** 0;

392 **return** 1;

393 }

* W linii 383 pierwszy warunek sprawdza czy podany znak jest liczbą lub literą, a drugi sprawdza czy znak nie jest liczbą
* W linii 388 zmiennej k jest powypisywana wartość 0 gdy warunek nie zostanie spełniony
* W liniach 391-392 funkcja zwraca wartość 0 gry k==0 lub 1 gdy k==1

**Funkcja sprawdzająca czy użytkownik o podanym ID istnieje:**

421 **int** user\_exist(**struct** User\_Node \* FrontUserList,**int** data)

422 {

423 **for** (; NULL != FrontUserList; FrontUserList = FrontUserList -> next)

424 **if**(FrontUserList->ID==data)

425 {

426 **return** 1;

427 **break**;

428 }

429 **return** 0;

430 }

* Funkcja, jako argumenty pobiera wskaźnik na pierwszy element w liście i ID podane przed użytkownika
* W linii 423 poruszamy się po liście
* Jeśli warunek z linii 424 zostanie spełniony to funkcja zwróci 1 i przerwie swoje działanie.

**Procedura zapisująca dane do plików:**

**void** save(FILE \*file,FILE \*file2,FILE \*file3, **struct** User\_Node \* FrontUserList, **struct** telephone\_nr \* FrontTelList, **struct** email\_node \*FrontEmailList)

19 {

20 file=fopen("USERS.txt","w");

21 file2=fopen("TELEPHONE.txt","w");

22 file3=fopen("EMAIL.txt","w");

23 **if**(file==NULL || file2==NULL || file3==NULL)

24 {

25 printf("Blad odczytu pliku");

26 exit(0);

27 }

28

29 **for** (; NULL != FrontUserList ; FrontUserList = FrontUserList -> next )

30 {

31 fprintf (file,"\n%u\n%s\n%s\n%s\n%s\n%hu\n%s\n%s",

32 FrontUserList->ID, FrontUserList->name, FrontUserList->surname, FrontUserList -> city,

33 FrontUserList->street, FrontUserList->house\_number, FrontUserList->postal\_code,

34 FrontUserList->post\_office);

35 }

36 **for** (; NULL != FrontTelList ; FrontTelList = FrontTelList -> next )

37 {

38 fprintf(file2,"\n%u\t", FrontTelList->ID\_tel);

39 fprintf(file2,"%lu", FrontTelList->tel\_nr);

40 }

41 **for** (; NULL != FrontEmailList ; FrontEmailList = FrontEmailList -> next )

42 {

43 fprintf(file3,"\n%u\t", FrontEmailList->ID\_email);

44 fprintf(file3,"%s", FrontEmailList->em);

45 }

46

47 **if**(fclose(file) || fclose(file2) || fclose(file3))

48 {

49 printf("Blad zamkniecia pliku");

50 }

51 }

* W liniach 20-22 otwierane są pliki do zapisu
* W linii 23 sprawdzane jest czy udało się utworzyć pliki
* W liniach 29-35 zapisywane są dane kontaktów do pliku USERS.txt
* W liniach 36-40 zapisywane są numery telefonów do pliku TELEPHONE.txt
* W liniach 41-45 zapisywane są adresy e-mail do pliku EMAIL.txt
* W liniach 47 sprawdzamy czy poprawnie zamknięto pliki

1. **Podsumowanie:**

Nie udało się zrealizować dwuosobowego projektu. Projekt napisała jedna osoba.

No i parę rzeczy można by poprawić :D