BAZA KONTAKTÓW

Podstawy Programowania 2

Informatyka 1, semestr 2, grupa 13B
Politechnika Świętokrzyska

Kinga Nowakowska, przy udziale Julia Pająk i Małgorzata Operacz

I. Temat projektu.

Tematem projektu jest baza kontaktów dla wielu użytkowników. Należało zrealizować logowanie z użyciem loginu i hasła. Program powinien posiadać funkcjonalności takie jak: dodawanie/usuwanie/edycja kontaktów, wyświetlanie, wyszukiwanie, sortowanie listy kontaktów, tworzenie/usuwanie/edycja grup kontaktów, a także zapisywanie oraz odczytywanie kontaktów do/z plików. Co ważne projekt należało zrealizować z wykorzystaniem list.

II. Link do prezentacji.

Część 1 prezentacji: https://youtu.be/ZBMUclVwUDY

Część 2 prezentacji: https://youtu.be/T1OM9M0kOBg

III. Wykorzystane narzędzia.

Język programowania: C

Środowisko programistyczne IDE: CodeBlocks

System operacyjny: Windows 10 Home

IV. Opis realizacji programu.

1. Pierwszy Kamień Milowy.

   Zgodnie z harmonogramem pierwszy kamień milowy obejmował:

   - Zakładanie i logowanie użytkowników,
   - Stworzenie interfejsu użytkownika,
   - Zapisywanie/odczytywanie kontaktów do/z pliku,
   - Wyświetlanie,
   - Dodawanie/usuwanie/edycja kontaktów.

   Aby zarządzać użytkownikami stworzono strukturę *Users*, która posłużyła do zbudowania listy jednokierunkowej przechowującej użytkowników. Posiada ona pola *nickname* oraz *password* przechowujące login i hasło użytkownika. Jest tam również wskaźnik *next* na następny element listy. Początkowo struktura miała również przechowywać wskaźnik do pliku z kontaktami danego użytkownika, jednak zrezygnowano z tego rozwiązania z powodu jego niepoprawności.

   Do obsługi listy użytkowników stworzono następujące funkcje:

   a) *Create_users_list* – funkcja tworząca pierwszy element (co za tym idzie inicjująca listę) listy użytkowników. Jako parametry przyjmuje login oraz hasło pobrane od użytkownika, a zwraca wskaźnik na pierwszy element listy. Jej zadaniem było

również utworzenie pliku, w którym przechowywane będą kontakty danego użytkownika.

b) *insert_user_front, find_user_spot, insert_user_after, insert_user_back* – funkcje pomocnicze wykorzystywane do dodawania nowych użytkowników w funkcji *add_new_user()*. Są to typowe funkcje wykorzystywane przy obsłudze list jednokierunkowych; ułatwiają dodawanie elementów do listy uporządkowanej.

c) *add_new_user* – funkcja dodająca nowy kontakt do niepustej listy. Jako parametry przyjmuje ona login i hasło nowego użytkownika oraz wskaźnik na pierwszy element listy. Jej zadaniem było utworzenie nowego kontaktu, wypełnienie go odpowiednimi danymi i dodanie do uporządkowanej listy. Funkcja zwraca wskaźnik na początek listy.

d) *loadingUsers* – funkcja, której zadaniem jej sprawdzenie czy istnieją (zapisani w odpowiednim pliku) użytkownicy i dodanie ich do listy. Przyjmuje ona jako parametr wskaźnik na początek listy i również zwraca wskaźnik na początek listy. W celu przechowywania istniejących użytkowników stworzono plik „*listOfUsers.txt*".

e) *updatingUsersFile* – funkcja, która odpowiada za zapisanie użytkowników do pliku pod koniec działania programu. Przyjmuje wskaźnik na pierwszy element listy, a zwraca wartość typu bool: *true* – jeśli zapis się powiedzie, *false* – jeśli zapis się nie powiedzie.

f) *delete_user_front, find_user_prev_node, delete_user_after, delete_user_node* – funkcje obsługujące usuwanie użytkownika. Funkcja *delete_node()* jako parametr przyjmuje wskaźnik na początek listy oraz login użytkownika, po którym będzie on wyszukiwany. Funkcja ta ma prawo zostań wywołana tylko jeśli zostało wcześniej podane prawidłowe hasło dla tego użytkownika. Implementacja funkcji usuwających jest typowa dla obsługi list jednokierunkowych uporządkowanych.

g) *print_all_users* – funkcja wyświetlająca nazwy wszystkich istniejących użytkowników. Przyjmuje wskaźnik na pierwszy element jako parametr i nie zwraca żadnej wartości.

h) *passwordCheck* – funkcja odpowiadająca za sprawdzenie czy użytkownik o podanym loginie istnieje oraz czy podane hasło jest prawidłowe. Funkcja jako parametry przyjmuje wskaźnik na pierwszy element listy, login oraz hasło, a zwraca wartość logiczną: *true* – jeżeli użytkownik istnieje i podane hasło jest prawidłowe, *false*

– jeżeli użytkownik nie istnieje lub hasło jest nieprawidłowe. Działanie funkcji oparte jest na przejściu przez listę.

i) *changePassword* – funkcja umożliwiająca zmianę hasła. Jest bardzo podobna do funkcji *passwordCheck()* jednak przyjmuje dodatkowy parametr – *new_password*, a gdy znajdzie odpowiedniego użytkownika i hasło się zgadza, wtedy nadpisuje stare hasło nowym i zwraca wartość *true*. W przeciwnym wypadku zwraca wartość *false*.
j) *signIn* – funkcja odpowiadająca za logowanie się użytkowników. Jako parametr przyjmuje wskaźnik na pierwszy element listy użytkowników, login oraz hasło. Przeszukuje ona listę użytkowników w celu znalezienia użytkownika o podanym loginie i haśle. Jeżeli nie ma takiego użytkownika lub hasło jest nieprawidłowe, funkcja zwraca wartość *false*. W przeciwnym wypadku zwraca wartość *true* i umożliwia operowanie na kontaktach danego użytkownika.
k) *dataInput* – funkcja pomocnicza prosząca użytkownika o podanie loginu i hasła. Nie zwraca żadnej wartości.
l) *userInterface* – funkcja realizująca interfejs obsługi użytkownika. Wyświetla ona powitanie oraz menu prezentujące funkcjonalności obsługi kontaktów. Interfejs ten będzie wyświetlany po zalogowaniu użytkownika. Jako parametr przyjmuje on nazwę użytkownika (tylko w celu umieszczenia jej w powitaniu). Nie zwraca żadnej wartości.

Ponadto w funkcji *main()* zaimplementowany jest interfejs zarzadzania użytkownikami. Gdy lista użytkowników jest pusta wyświetlane jest zapytanie o chęć założenia pierwszego użytkownika. Jeżeli użytkownik nie wyrazi takiej chęci program jest kończony (bez żadnego użytkownika nie da się obsługiwać kontaktów). W przeciwnym wypadku użytkownik jest tworzony i wyświetlane jest menu umożliwiające zarządzanie użytkownikami. Możliwe opcje: logowanie, założenie nowego konta, zmiana hasła, usunięcie konta, wyświetlenie użytkowników oraz wyjście. Wyjście jednoznaczne jest z zapisaniem listy użytkowników do pliku.

Ponadto podjęto próbę stworzenia struktury obsługującej listę kontaktów oraz odpowiednich funkcji umożlwiających zarządzanie nią, a także stworzenie przykładowych programów.

Na etapie pierwszego kamienia milowego pojawiły się następujące problemy:

- logowanie z wykorzystaniem niepoprawnego hasła,
- brak spójności programu (program w 3 plikach),
- nieprawidłowe działanie niektórych części programu.

2. Drugi Kamień Milowy.

   W ramach II Kamienia Milowego poprawiono błędy: logowanie z użyciem niepoprawnego hasła oraz stworzono listę jednokierunkową *Contact* przechowującą dane kontaktowe osób i id każdego kontaktu. Dokonano również zmiany w strukturze *Users*, gdzie pojawiło się pole przechowujące nazwę pliku, w którym zapisane będą kontakty danego użytkownika. Również lista użytkowników stała się listą nieuporządkowaną. Zdecydowano się na taki zabieg, ponieważ uporządkowanie listy nie dawało programowi żadnych realnych korzyści.

   Ponadto stworzono funkcje pozwalające przeprowadzać podstawowe operacje na liście kontaktów:

   - *create_list* – tworzenie listy kontaktów;
   - *insert_back, insert_node* – dodawanie kontaktu do końca listy (kontakty na liście są posortowane według id, przydzielanego im podczas dodawania, z tego względu wystarczy dodawać je zawsze na końcu listy);
   - *delete_front*, *find_prev_node*, *delete_after*, *delete_node* – usuwanie kontaktu z listy;
   - *id_updating* – funkcja aktualizująca wartości id, po usunięciu jakiegoś elementu z listy;
   - *editContact* – edycja wybranego pola kontaktu wskazanego poprzez id;
   - *print_list, print_contact* – wyświetlanie kontaktu wskazanego przez id lub całej listy kontaktów;
   - *remove_list* – usunięcie listy kontaktów i zwolnienie pamięci.

   Stworzono również dwie funkcje: *updatingContactsFile()* oraz *loadingContacts(),* których zadaniem jest zapisanie kontaktów z listy do pliku lub pobranie ich z pliku i dodanie do struktury.

   Zarządzanie kontaktami danego użytkownika odbywa się w jednej funkcji tj. *userInterface()*. Znajduje się tam menu, z którego można wybrać różne opcje obsługi kontaktów począwszy od tworzenia, usuwania i edycji kontaktów, skończywszy na sortowaniu i tworzeniu grup.

Funkcjonalności, które zgodnie z harmonogramem obejmuje II Kamień Milowy:

a) Sortowanie.

   Zarządzanie sortowaniem odbywa się w jednej funkcji – *sorting_interface(),* gdzie początkowo użytkownik wybiera sortowanie nierosnące lub niemalejące, a następnie pole według którego chce posortować elementy. Pytanie o preferencje sortowania odbywa się z wykorzystaniem instrukcji wyboru *switch.* Zgodnie z wyborem użytkownika uruchamiana jest jedna z 18 funkcji. Samo sortowanie odbywa się z wykorzystaniem meta-listy, w której umieszczane są wskaźniki do elementów oryginalnej listy w odpowiedniej kolejności. W tym celu stworzono dodatkową strukturę *MetaList*, która przechowuje tylko wskaźniki do pierwotnej struktury i wskaźnik do następnego elementu. Po posortowaniu meta-lista jest wyświetlana i usuwana. Dzięki temu oryginalna lista jest zachowana w niezmienionej wersji. Do usuwania listy stworzono dodatkową funkcję *remove_mList(),* która jest taka sama jak *remove_list(),* ale operuje na innej strukturze.

b) Wyszukiwanie.

   Do wyszukiwania zastosowano prostą funkcję *find_contact()*, która wyszukuje kontakty na liście według wskazanego id.

c) Grupowanie.

   Aby obsłużyć tę funkcjonalność stworzono dodatkową listę jednokierunkową *Group*. Struktura ta ma następujące pola: *title* (nazwa grupy), *number_of_elements* (liczba kontaktów należących do tej grupy, początkowo 0), wskaźnik na następny element listy *next* oraz tablicę dynamiczną przechowującą wskaźniki do kontaktów. Podczas dodawania kolejnych kontaktów do tablicy, pamięć jest realokowana dla nowej liczby elementów, a gdy użytkownik usunie kontakt z grupy, wtedy z wykorzystaniem funkcji: *rewrite_array()* oraz *delete_contact_from_group()* stara tablica jest przepisywana (bez usuniętego elementu) do nowej, usuwana i zastępowana nową. Nie wykorzystano w tym przypadku funkcji *realloc*, ponieważ, w przypadku usunięcia elementu ze środka tablicy, nie było pewności czy w prawidłowy sposób realokuje pamięć na mniejszą tablice. Stworzono również funkcje obsługujące różne działania na grupach: *create_group_list(), add_group(), delete_group(), find_group(), print_group(), print_all_groups()*, a także *edit_group()*, która umożliwia dodawanie i usuwanie kontaktów z grupy. Wyszukiwanie konkretnej grupy odbywa się z wykorzystaniem jej nazwy, czyli *title.*

Na etapie drugiego kamienia milowego pojawiły się następujące błędy:

- Niepotrzebna funkcja *id_updating()* – należy zachować id bez zmian,
- Wyszukiwanie tylko po ID oraz bez ignorowania wielkości znaków,
- Zapis grup do pliku nie działa,
- Brak wielokrotnych tel. i e-maili.

3. Obrona projektu.

Na tym etapie projektu wprowadzono następujące zmiany:

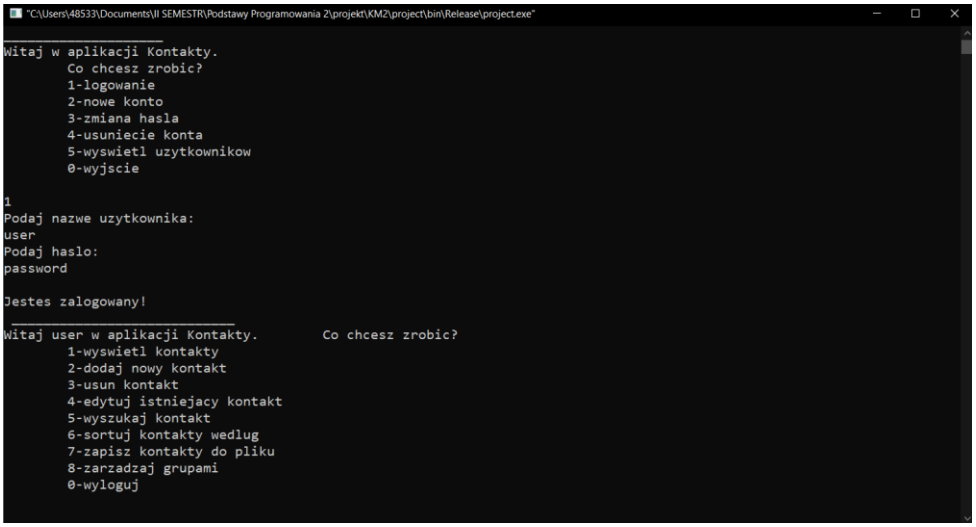
- Usunięto funkcję aktualizującą id.
- Dodano funkcję pomocniczą *user_check()* sprawdzającą czy dany użytkownik istnieje. Funkcja ta zwraca wartość typu logicznego. Wykorzystywana jest przy tworzeniu nowego użytkownika aby zapobiec sytuacji wielu użytkowników o takich samych loginach.
- Funkcja *find_contact()* została zastąpiona funkcją *find_node()*, która podobnie jak jej poprzedniczka wyszukuje kontakt na podstawie podanego id. Ponadto stworzono nową funkcję *find_contacts()*, która obsługuje wyszukiwanie kontaktu po dowolnym polu. W funkcji znajduje się menu, z którego można wybrać pole według, którego chcemy wyszukać użytkownika. Funkcja ta wyszukuje kontakt po fragmencie tekstu lub tekście z ignorowaniem wielkości liter. Po znalezieniu dane kontaktu są wyświetlane z użyciem funkcji *print_contact()*. Co warto zaznaczyć funkcja wyświetla wszystkie kontakty pasujące do wyszukiwania, nie tylko pierwszy z nich.
- Stworzono osobną funkcję *group_interface()* odpowiadającą za zarządzanie grupami. Ma ona swoje własne menu w którym użytkownik ma następujące możliwości: wyświetlenie wszystkich grup, dodanie nowej grupy, usunięcie grupy, wyszukanie grupy, edycja grupy, zapisanie grupy do pliku, wyjście.
- Stworzono dwie funkcje odpowiadające za zapis i odczyt grup z plików: *loadingGroups()* oraz *updatingGroupsFile()*. Ich zadaniem jest trwały zapis składu grup oraz możliwość ich odczytu po ponownym uruchomieniu programu.
- Zmniejszono liczbę dostępnych opcji w menu głównym użytkownika, gdzie był wcześniej dostęp do różnych możliwości zarządzania grupami. Co się z tym wiąże niektóre opcje zostały przeniesione do osobnych funkcji posiadających własne wewnętrzne menu.

- Dodano również pytania do użytkownika o potwierdzenie akcji, w przypadku usuwania lub edycji kontaktu, a także wyszukiwanie kontaktów do edycji lub usunięcia tylko po id.

Podczas poprawiania programu zrezygnowano z:

- Przeniesienie fragmentów funkcji sortujących do osobnej funkcji (w celu zmniejszenia ilości kodu), ponieważ nie jest to zabieg niezbędny, a rzeczą ważniejszą jest zapewnienie poprawności sortowania co już udało się osiągnąć.
- Czyszczenie grupy bez usuwania jej, ponieważ nie jest to opcja potrzebna. Jej obecność nie wniosłaby nic istotnego do projektu.

V. Zrzuty ekranu z przykładami działania programu:



VI. Podział pracy pomiędzy członków grupy.

Julia Pająk – próba realizacji obsługi kontaktów w IKM;

Małgorzata Operacz – próba stworzenia przykładowych kontaktów w IKM;

Kinga Nowakowska – duża część obsługi użytkowników w IKM; od tamtego momentu praca samodzielna.

VII. Podsumowanie.

Ostateczny projekt realizuje prawie wszystkie założenia z harmonogramu i polecenia. Nie udało się zrealizować przypisania do kontaktu większej niż jeden liczby numerów telefonu i adresów e-mail. Program można wzbogacić o więcej możliwości zarządzania kontaktami, takie jak np. dodawanie zdjęcia do kontaktu. Projekt mógłby zostać również zrealizowany z wykorzystaniem biblioteki graficznej, gdyby była taka potrzeba.