**SPRAWOZDANIE**

|  |  |  |  |
| --- | --- | --- | --- |
| **Przedmiot** | Algorytmy i struktury danych | **Zadanie** | 2\_02 |
| **Autor** | Kamil Borkowski | **Grupa** | WCY22IY1S1 |
| **Temat** | Wizytownik | | |

1. Treść

Zaimplementuj program realizujący w formie listy dwukierunkowej następujące funkcje wizytownika:

d – dodaj wizytówkę,

s – szukaj wizytówki,

w – wypisz wizytówki (A… Z),

v – wypisz wizytówki (Z… A),

u – usuń wizytówkę,

z – zapisz rekordy w pliku

inne – powtórz sterowanie

e – kończy działanie

* 1. Metoda realizacji

Deklarujemy listę dwukierunkową i tworzymy funkcje dla każdego polecenia wizytownika. Pobieranie komend dajemy w pętlę while(1), żeby móc podawać komenda po komendzie.

* 1. Założenia / ograniczenia dotyczące danych:
     1. Dane wejściowe *(opis)*

Nazwiska, imię, nr telefonu – podawane z klawiatury

* + 1. Dane wyjściowe *(opis)*

Rekordy listy – wyświetlone na ekranie i zapisane do pliku (\*.txt)

1. Realizacja
   1. Algorytm *(lista kroków lub schemat blokowy)*

Deklaracja struktury listy

Utworzenie funkcji void dodającą nowy element wizytownika

Utworzenie funkcji void wypisującą wszystkie elementy listy

Utworzenie funkcji float wypisującą wszystkie elementy listy od tyłu

Utworzenie funkcji void usuwającą element o podanym nazwisku

Utworzenie funkcji void wypisującą element o podanym nazwisku

Umieszczenie wywoływania funkcji w pętli while(1)

* 1. Kod źródłowy

#include<stdio.h>

#include<stdlib.h>

typedef struct list{

char imie[20];

char nazwisko[20];

int numer\_telefonu;

struct list\* previous;

struct list\* next;

} list;

void dodaj\_element(list\*\* head, char\* imie, char\* nazwisko, int numer\_telefonu)

{

list\* nowy\_element = (list\*)malloc(sizeof(list));

strncpy(nowy\_element->imie, imie, 20);

strncpy(nowy\_element->nazwisko, nazwisko, 20);

nowy\_element->numer\_telefonu = numer\_telefonu;

nowy\_element->next = NULL;

if (\*head == NULL) {

// Jeśli lista jest pusta, nowy element staje się głową listy.

nowy\_element->previous = NULL;

\*head = nowy\_element;

} else {

list\* obecny\_element = \*head;

list\* poprzedni\_element = NULL;

// Szukamy odpowiedniego miejsca na liście alfabetycznie po pierwszej literze nazwiska.

while (obecny\_element != NULL && strcmp(nazwisko, obecny\_element->nazwisko) > 0) {

poprzedni\_element = obecny\_element;

obecny\_element = obecny\_element->next;

}

// Jeśli poprzedni\_element nie jest NULL, to wstawiamy nowy\_element pomiędzy poprzedni\_element i obecny\_element.

if (poprzedni\_element != NULL) {

nowy\_element->previous = poprzedni\_element;

nowy\_element->next = obecny\_element;

poprzedni\_element->next = nowy\_element;

if (obecny\_element != NULL) {

obecny\_element->previous = nowy\_element;

}

} else {

// Jeśli poprzedni\_element jest NULL, to nowy\_element staje się nową głową listy.

nowy\_element->previous = NULL;

nowy\_element->next = \*head;

(\*head)->previous = nowy\_element;

\*head = nowy\_element;

}

}

}

void wyswietl\_liste(list\* head)

{

list\* current = head;

while (current != NULL) {

printf("Imie: %s\n", current->imie);

printf("Nazwisko: %s\n", current->nazwisko);

printf("Numer telefonu: %d\n", current->numer\_telefonu);

printf("\n");

current = current->next;

}

}

void wyswietl\_liste\_tyl(list \*head)

{

printf("\n");

if(head==NULL) printf("List is empty");

else

{

list \*current=head;

while (current->next != NULL){

current = current->next; //idziemy na koniec listy

}

while(current!=NULL){

printf("Imie: %s\n", current->imie);

printf("Nazwisko: %s\n", current->nazwisko);

printf("Numer telefonu: %d\n", current->numer\_telefonu);

printf("\n");

current = current->previous;

}

}

}

void usun(list\*\* head, char\* nazwisko\_)

{

list\* current = \*head;

// Sprawdzamy czy lista jest pusta.

if (current == NULL) {

return; // Lista jest pusta, nie ma nic do usunięcia.

}

// Sprawdzamy czy pierwszy element pasuje do nazwiska.

if (strcmp(current->nazwisko, nazwisko\_) == 0) {

\*head = current->next;

if (current->next != NULL) {

current->next->previous = NULL;

}

free(current);

return;

}

// Przeszukujemy resztę listy.

while (current->next != NULL) {

if (strcmp(current->next->nazwisko, nazwisko\_) == 0) {

list\* tmp = current->next;

current->next = tmp->next;

if (tmp->next != NULL) {

tmp->next->previous = current;

}

free(tmp);

return;

}

current = current->next;

}

}

// Funkcja wypisująca elementy o podanym nazwisku.

void wypisz\_elementy\_po\_nazwisku(list\* head, const char\* nazwisko\_do\_wyswietlenia) {

list\* current = head;

int znaleziono = 0;

while (current != NULL) {

if (strcmp(current->nazwisko, nazwisko\_do\_wyswietlenia) == 0) {

// Znaleziono element o pasującym nazwisku.

printf("Imie: %s\n", current->imie);

printf("Nazwisko: %s\n", current->nazwisko);

printf("Numer telefonu: %d\n", current->numer\_telefonu);

printf("\n");

znaleziono = 1;

}

// Przechodź do następnego elementu.

current = current->next;

}

if (!znaleziono) {

printf("Nie znaleziono osoby o nazwisku: %s\n", nazwisko\_do\_wyswietlenia);

}

}

int main()

{

list\* head = NULL;

// Dodajemy pierwsze elementy do listy

dodaj\_element(&head, "Jan", "Kowalski", 123456789);

dodaj\_element(&head, "Anna", "Nowak", 987654321);

dodaj\_element(&head, "Piotr", "Zielinski", 456123789);

// Petla pozwala nam wykonywac wiele polecen jedno po drugim

while(1)

{

// Wybieramy komende do przeprowadzenia

char i;

printf("Mozliwe komendy:\n");

printf("d - dodaj wizytowke:\n");

printf("s - szukaj wizytowki\n");

printf("w - wypisz wizytowki(A...Z)\n");

printf("v - wypisz wizytowki(Z...A)\n");

printf("u - usun wizytowke\n");

printf("e - wyjdz\n");

scanf(" %c",&i);

if(i=='d')

{

char imie[20];

char nazwisko[20];

int numer\_telefonu;

// Pobieramy dane nowego elementu

printf("Podaj imie: ");

scanf("%s", imie);

printf("Podaj nazwisko: ");

scanf("%s", nazwisko);

printf("Podaj numer telefonu: ");

scanf("%d", &numer\_telefonu);

//Dodajemy nowy element do listy

dodaj\_element(&head, imie, nazwisko,numer\_telefonu);

}

if(i=='s')

{

printf("Podaj nazwisko osoby, ktora chcesz wyswietlic: \n");

char nazwisko\_do\_szukania[20];

scanf("%s", nazwisko\_do\_szukania);

wypisz\_elementy\_po\_nazwisku(head,nazwisko\_do\_szukania);

}

if(i=='w')

{

//Wyœwietlamy wszystkie elementy listy

wyswietl\_liste(head);

}

if(i=='v')

{

//Wyœwietlamy wszystkie elementy listy od tylu

wyswietl\_liste\_tyl(head);

}

if(i=='u')

{

//Podajemy nazwisko osoby do usuniecia i pozbywamy sie jej

char nazwisko\_do\_usuniecia[20];

printf("Wpisz nazwisko osoby, ktorej chcesz usunac:\n");

scanf("%s", nazwisko\_do\_usuniecia);

usun(&head, nazwisko\_do\_usuniecia);

}

if(i=='e')

{

//wychodzimy z petli - opuszczamy program

printf("Wychodzenie...\n");

break;

}

}

return 0;

}

Dane wejściowe *(ekran uruchomienia programu)*

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

* 1. Dane wyjściowe *(ekran uruchomienia programu)*

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

Obraz zawierający tekst, zrzut ekranu, Czcionka, design

Opis wygenerowany automatycznie

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

1. Wnioski *(złożoność asymptotyczna algorytmu przy użyciu notacji* ***O lub innej i inne wnioski)***

Złożoność obliczeniowa programu wynosi: O(n), ponieważ w najgorszym przypadku program będzie musiał przejrzeć wszystkie elementy listy od początku do końca w funkcjach: usun, dodaj lub szukaj.