Listy, część druga

Na poprzednich zajęciach omówione zostały podstawowe informacje, dotyczące list. Wiadomości te zostaną obecnie rozszerzone o dodawanie elementów do listy na dowolnej pozycji oraz wyszukiwanie informacji na liście. Zagadnienia te zostaną omówione na przykładzie programu, realizującego następujące zagadnienie:

Zadanie

Przygotować program, pozwalający na pobranie od użytkownika serii danych o dowolnej długości, zawierających imię, nazwisko i numer telefonu poszczególnych osób, wyświetlenie listy posortowanej w porządku alfabetycznym względem nazwisk oraz na wyszukanie danych osoby o podanym nazwisku.

Powyższe zadanie najprościej zrealizować za pomocą programu, przechowującego dane wejściowe w liście jednokierunkowej, w której nowe elementy będą wstawiane na takich pozycjach, aby w każdej chwili został utrzymany porządek alfabetyczny nazwisk w elementach listy. Wymagać to będzie od nas zaprogramowania operacji dodawania nowych elementów do listy listy w taki sposób, aby nowe elementy były dodawane na odpowiedniej pozycji, a nie (jak było w zagadnieniu omówionym na poprzednich zajęciach) zawsze na początek listy. Jednym ze sposobów zrealizowania dodawania nowego elementu na właściwą pozycję listy jednokierunkowej jest wydzielenie przypadków dodawania elementu na początek listy oraz na pozostałe pozycje na liście w następujący sposób:

- jeżeli lista jest pusta lub lista ma na pierwszej pozycji nazwisko osoby, które kwalifikuje je do zajęcia pozycji za nowo dodawanym, nowy element dodajemy na początek listy, poprzez przypisanie wskaźnikowi *„nastepny”* w nowo dodawanej strukturze wartości wskaźnika do początku listy, a następnie przypisaniu wskaźnikowi do początku listy adresu nowo dodawanej struktury
- w przeciwnym wypadku musimy znaleźć przed dodaniem nowego elementu listy element, który będzie poprzedzał go po zrealizowaniu procedury dodawania, gdyż w tym elemencie będzie modyfikowany wskaźnik na następny element listy. W tym celu:

1. przeglądamy listę od początku, aż nie natrafimy na ostatni element listy (nie mający następnika – wskaźnik *„nastepny”* ma wartość *NULL*), lub na element, którego następnik ma nazwisko, które kwalifikuje je do znalezienia się na liście po nazwisku nowo dodawanym. W ten sposób znaleziony zostaje element listy, za którym ma znaleźć się nowo dodawany element listy
2. po znalezieniu elementu, który ma poprzedzać nowo dodawany element listy, dołączamy za nim nowy element (przepisujemy wartość wskaźnika *„nastepny”* elementu poprzedzającego do nowo dołączanej struktury, a następnie adres nowo dołączanej struktury przypisujemy wskaźnikowi *„nastepny”* elementu poprzedzającego go na liście)

Warto zauważyć, że dla przeprowadzenia powyższej procedury musimy dysponować możliwością realizacji operacji porównania ze sobą struktur ze względu na wartość składowej *nazwisko*, przyjmując porządek leksykograficzny jako kryterium porównania nazwisk – struktury z nazwiskami, znajdującymi się wcześniej na liście ułożonej alfabetycznie będą uznawane za „mniejsze”. Potrzebna jest również możliwość sprawdzenia równości struktur, w sensie istnienia w nich tych samych nazwisk, aby móc znaleźć na liście dane na temat szukanej osoby, co stanowi drugą część zadania.

Przykładowy program, realizujący rozważane zadanie ma postać (plik *ksiazka.cpp*):

```
#include <iostream>
#include <string>
#include <cctype>
#include <new>
```

```
using namespace std;

struct numer
{
	string imie;
	string nazwisko;
	string telefon;

	numer *nastepny;
};

// Prototypy
void dodajTelefon(numer **lista, numer *dane);
numer *znajdzOsobe(numer *lista, string nazwisko);
void usunListe(numer **lista);
void wypiszListe(numer *lista);
numer *wczytajOsobe();
void wypiszOsobe(numer *dane);

int main()
{
numer *lista = NULL;

	cout << "Program utworzy baze numerow telefonicznych" << endl << endl;

	cout << "Podaj czynnosc do wykonania:" << endl;
	cout << "d - dodaj wpis do bazy" << endl;
	cout << "s - szukaj osoby o podanym nazwisku" << endl;
	cout << "w - wypisz liste rekordow w bazie" << endl;
	cout << "z - zakoncz dzialanie programu" << endl << endl;

	char o;
	do
	{
		cout << "Podaj czynnosc do wykonania [d/s/w/z]: ";
		cin >> o;
		cout << endl;
		cin.ignore();

		// Zamień na dużą literę
		o = toupper(o);
		switch(o)
		{
			case 'D':
				{
					cout << "Wpisz dane nowej osoby w bazie: " << endl <<endl;
					numer *nowy = wczytajOsobe();
					if(nowy != NULL)
						dodajTelefon(&lista, nowy);
					break;
				}

			case 'S':
				{
					string szukany;

					cout << "Podaj szukane nazwisko: ";
					getline(cin, szukany);

					numer *znaleziony = znajdzOsobe(lista, szukany);
					if(znaleziony == NULL)
						cout << "Osoby nie ma w bazie" << endl << endl;
					else
					{
						cout << "Znaleziono nastepujaca osobe:" << endl;
						wypiszOsobe(znaleziony);
						cout << endl;
					}
					break;
				}
```

```
                                case 'W':
                                        wypiszListe(lista);
                                        cout << endl;
                                        break;

                                case 'Z':
                                        break;

                                default:
                                        cout << "Dopuszczalne opcje to d [dodaj], s [szukaj], w [wypisz] lub z [zakoncz]" <<
endl;
                        }
                }
                while(o != 'Z');
                usunListe(&lista);
}

void dodajTelefon(numer **lista, numer *dane)
{
                if(lista == NULL || dane == NULL) return;

                // Jesli lista jest pusta lub nowe nazwisko ma byc pierwsze na liscie...
                if(*lista == NULL || (*lista)->nazwisko > dane->nazwisko)
                {
                        // ... dodajemy "dane" na poczatek listy
                        dane->nastepny = *lista;
                        // "dane" jest obecnie pierwszym elementem
                        *lista = dane;
                        return;
                }
                else // *lista != NULL && *lista->nazwisko <= dane->nazwisko
                {
                        // Znajdujemy poprzednik nowego elementu w tworzonej liscie
                        numer *poprzednik = *lista;
                        while(  poprzednik->nastepny != NULL &&
                                        poprzednik->nastepny->nazwisko < dane->nazwisko)
                                poprzednik = poprzednik->nastepny;

                        // Dodawanie elementu za elementem "poprzedni"
                        dane->nastepny = poprzednik->nastepny;
                        poprzednik->nastepny = dane;
                }
}

numer *znajdzOsobe(numer *lista, string nazwisko)
{
                numer *aktualny = lista;
                while(aktualny != NULL)
                {
                        if(aktualny->nazwisko == nazwisko) return aktualny;
                        aktualny = aktualny->nastepny;
                }
                // Nie udalo sie znalezienie osoby w ksiazce
                return NULL;
}

void usunListe(numer **lista)
{
                numer *kasowany = *lista;
                while(kasowany != NULL)
                {
                        numer *nastepny = kasowany->nastepny;
                        delete kasowany;
                        kasowany = nastepny;
                }
                *lista = NULL;
}

void wypiszListe(numer *lista)
{
                if(lista == NULL)
```

```
                cout << "Lista jest pusta" << endl;
        else
        {
                cout << "Zawartosc bazy telefonow:" << endl << endl;
                unsigned i = 1;
                numer *aktualny = lista;
                while(aktualny != NULL)
                {
                        cout << "Osoba nr " << i << endl;;
                        wypiszOsobe(aktualny);
                        cout << endl;

                        aktualny = aktualny->nastepny;
                        i++;
                }
        }
}

numer *wczytajOsobe()
{
        numer *osoba;
        try {
                osoba = new numer;
        }
        catch(bad_alloc) {
                cout << "Brak pamieci" << endl;
                return NULL;
        }

        cout << "Podaj imie: ";
        getline(cin, osoba->imie);
        cout << "Podaj nazwisko: ";
        getline(cin, osoba->nazwisko);
        cout << "Podaj numer: ";
        getline(cin, osoba->telefon);
        cout << endl;
        return osoba;
}

void wypiszOsobe(numer *dane)
{
        if(dane != NULL)
        {
                cout << "Imie: " << dane->imie << endl;
                cout << "Nazwisko: " << dane->nazwisko << endl;
                cout << "Numer: " << dane->telefon << endl;
        }
}
```