Politechnika Śląska w Gliwicach

Wydział Automatyki, Elektroniki i Informatyki

Laboratorium Programowania Komputerów

System obsługi hotelu

autor	Bartosz Skrobacki
prowadzący	Piotr Pecka
rok akademicki	2018/2019
kierunek	Informatyka
Rodzaj studiów	SSI
semestr	3
kod programu	dostępny na github.com
grupa	1
sekcja	1
data oddania sprawozdania	2019-02-23

1 Temat

System obsługi hotelu, możliwa rezerwacja pokoju, jej odwołaniem aktualny stan pokoi.

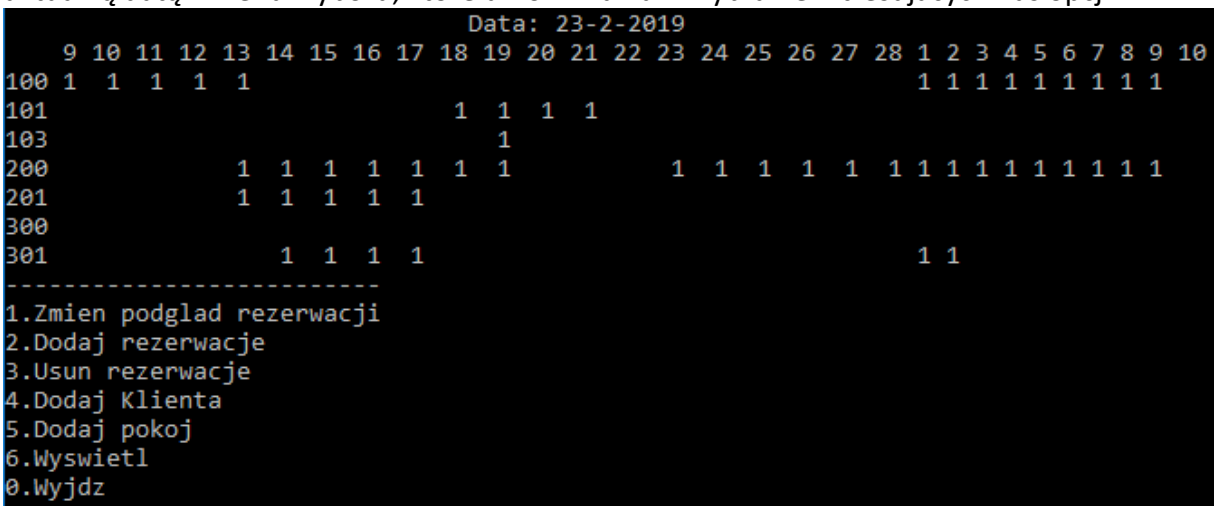
2 Analiza tematu

Projekt miał na celu zbudowanie systemu obsługi hotelu. Zostaną zdefiniowane trzy klasy główne pokój, klient i rezerewacja. Obiekt Rezerwacja będzie przechowywał informacje na temat daty zameldowania i wymeldowania oraz informacje o danym kliencie i pokoju, w którym będzie mieszkał. Wszystkie te informacje będą kolejno przechowywane w dynamicznych listach jednokierunkowych, co ułatwi nam szybkie wyszukiwanie interesujących nas danych. Na początku programu danę będą wczytywane z plików i wpisywane do list z osobnych plików tekstowych i na końcu będą one zapisywane do tych samych plików. Na podstawie listy rezerwacji, zostanie wygenerowania tablica dynamiczna służaca do reprezentacji danych na ekranie. Będzie ona przechowywała listę wszystkich pokoi,
Aktualną datę(+-14dni) oraz informację czy interesujący nas pokój jest zajęty w tym terminie.

3.Specyfikacja zewnętrzna

3.1 Obsługa programu

Program jest uruchamiany z linii polecen. Zostaje wyswietlone kalendarz rezerewacji, wraz z aktualną datą i menu wyboru, które umozliwia nam wybranie intresujacych nas opcji.



Kalendarz rezerwacji w pierwszej kolumnie od lewej wyświetla liste aktualnych pokoi, w pierwszym wierszu wyświetla dni miesiąca od aktualnej daty. W środku kalendarza rezerwacji 1- oznaczają pokój zajęty, „ „ – puste kratki pokój wolny.
Ad.1) Umożliwia zmianę podglądu kalendarza rezerwacji – użytkownik może wybrać aktualną datę lub wprowadzić własną interersującą go datę. Kalendarz rezerwacji zostanie wygenerowany na nowo dla żądanej daty.
Ad.2)Pozwala na dodanie nowej rezerwacji uprzednio po wprowadzeniu danych przez użytkownika.
Ad.3)Pozwala na usunięcie rezerwacji, po podaniu daty i numeru pokoju przez użytkownika.
Ad.4)Umożliwia dodanie nowego klienta.

Ad.5)Umożliwia dodaniego nowego pokoju.

Ad.6)Wywołuje podmenu wyświetl, w którym użytkownik może wybrać liste pokoi, klientów lub rezerwacji do wyświetlenia.

Ad.7)Konczy program.

3.2 Komunikaty

Użytkownik po wpisaniu błędnych danych będzie odpowiednio informowany o tym co należy zrobić, aby skorzystać z *systemu obsługi hotelu*.

Np. Przy wpisywaniu peselu, jeżeli użytkownik poda wartość o długości różnej od 11, zostanie wyświetlony komuniktat:

```
Podaj prawidłowy pesel
```

W tym momencie użytkownik powinien jeszcze raz wpisać pesel, ale tym razem prawidłowy.

4.Specyfikacja wewnętrzna

Program został podzielony na pliki .cpp oraz .hpp:

- klient.hpp
- lista.hpp
- menu.hpp
- pokoj.hpp
- rezerwacja.hpp
- tabela_rezerwacji.hpp
- klient.cpp
- lista.cpp
- menu.cpp
- pokoj.cpp
- rezerwacja.cpp
- tabela_rezerwacji.cpp
- main.cpp

Pliki cpp zawierają ciała funkcji, natomiast pliki hpp definicje funkcji użytych w plikach cpp o tej samej nazwie.

4.1 Klasy

Klient:

```
class Klient {
	string nazwisko;
	string imie;
	string pesel;
public:
```

```
	Klient(string nazwisko = "Kowalski", string imie = "Jan", string pesel =
"9709110672");

	void wyswietl();
	int operator>(const Klient &kli);
	friend ostream & operator<<(ostream &, const Klient &);
	friend istream & operator >>(istream &, Klient &);


	friend class Lista_klient;
	};
```

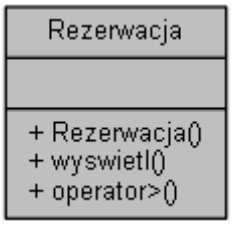
Pokoj:

```
class Pokoj
{
	int numer;
	int ilosc_osob;
	bool zajety;
public:
	Pokoj(int num = 0, int ilo = 1, bool zaj = false);
	int operator>(const Pokoj &pok);
	void wyswietl();


	friend ostream & operator<<(ostream &, const Pokoj &);
	friend istream & operator>>(istream &, Pokoj &);


	friend class Lista_pokoj;
	friend class Lista_rezerwacja;
	friend class Tabela_rezerwacji;
};
```

Rezerwacja:

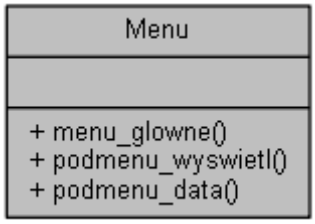
```
class Rezerwacja {

	struct tm poczatek_rezerwacji;
	struct tm koniec_rezerwacji;
	int numer_rezerwacji;

	Klient *wskklient;
	Pokoj  *wskpokoj;
```

```
public:
	Rezerwacja(int numer_rez = 0, int poczatek_dzien=1,int poczatek_miesiac=1,int
poczatek_rok=2019,int koniec_dzien=31,int koniec_miesiac=1,int koniec_rok=2019, Klient
*wskkli=nullptr, Pokoj *wskpok = nullptr);
	void wyswietl();
	bool operator>(const Rezerwacja &);

	friend ostream & operator<<(ostream &, const Rezerwacja &);
	friend istream & operator>>(istream &, Rezerwacja &);
	friend class Lista_rezerwacja;
	friend class Tabela_rezerwacji;

};
```



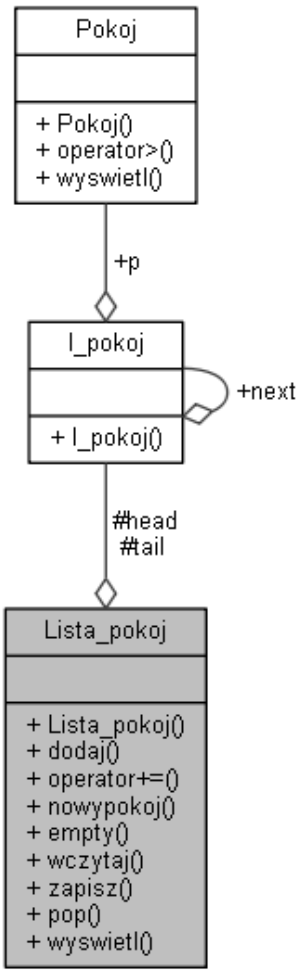
Menu:

```
class Menu {

public:

	void menu_glowne();
	void podmenu_wyswietl(const Lista_rezerwacja &list_r, const Lista_pokoj &list_p,
const Lista_klient &list_k);
	void podmenu_data( Lista_rezerwacja &list_r,  Lista_pokoj &list_p,
Tabela_rezerwacji &tab);
};
```



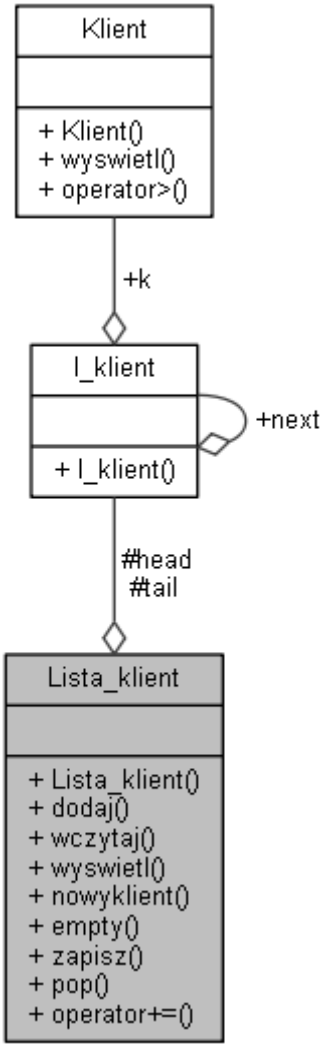
Lista_pokoj:

```
class Lista_pokoj {
protected:
	l_pokoj * head;
	l_pokoj * tail;
public:
	Lista_pokoj() :head(nullptr) {};

	Lista_pokoj dodaj(const Pokoj &pok);
	Lista_pokoj operator+=(const Pokoj &pok);
	void nowypokoj(Lista_pokoj &list);
	void empty();

	void wczytaj( Lista_pokoj &list);
	void zapisz(const Lista_pokoj &pok);


	Pokoj* pop(int numer, Lista_pokoj &pok);

	void wyswietl();
	friend ostream & operator<<(ostream &, const Lista_pokoj &);
	friend istream & operator >>(istream &, Lista_pokoj &);

	friend class Rezerwacja;
	friend class Lista_rezerwacja;
	friend class Tabela_rezerwacji;

                                                };
```



Lista_klient:

```
class Lista_klient {
protected:
	l_klient * head;
	l_klient * tail;
public:
	Lista_klient() :head(nullptr) {};
	Lista_klient dodaj(const Klient &kli);
	void wczytaj(Lista_klient &lista);
	void wyswietl();
	void nowyklient(Lista_klient &list);
	void empty();
	void zapisz(const Lista_klient &kli);
	Klient* pop(string naz, Lista_klient &kli);
	Lista_klient operator+=(const Klient &kli);
	friend ostream & operator<<(ostream &, const Lista_klient &);
	friend istream & operator >>(istream &, Lista_klient &);
	friend class Rezerwacja;


};
```



Lista_rezerwacja:

```
class Lista_rezerwacja {
protected:
	l_rezerwacja * head;
	l_rezerwacja * tail;
public:
	Lista_rezerwacja() :head(nullptr) {};
	Lista_rezerwacja dodaj(const Rezerwacja &rez);
	Lista_rezerwacja operator+=(const Rezerwacja &rez);
	void empty();
	void nowyrezerwacja(Lista_rezerwacja &list_r, Lista_pokoj &list_p, Lista_klient
&list_k);
	void wczytaj(Lista_rezerwacja &lista_r, Lista_klient &lista_k, Lista_pokoj
&lista_p);
	void zapisz(const Lista_rezerwacja &rez);
	bool zajety(Lista_rezerwacja &list_r, Rezerwacja &rez);
	friend ostream & operator<<(ostream &, const Lista_rezerwacja &);
	friend class Tabela_rezerwacji;

};
```

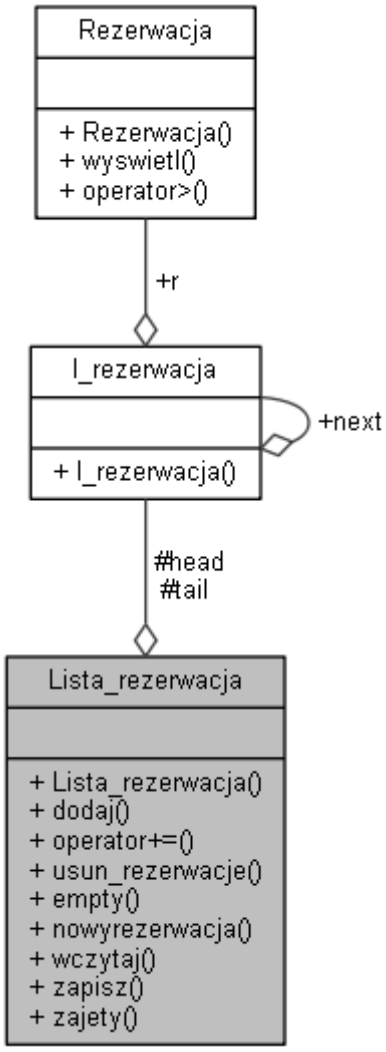


Tabela rezerwacji:

```
class Tabela_rezerwacji
{
	size_t W;
	size_t K;
	int **tablica;
```

```
        tm *data;
public:

        Tabela_rezerwacji(const Lista_rezerwacja &list_r, const Lista_pokoj &list_p);
        Tabela_rezerwacji refresh(const Lista_rezerwacja &list_r, const Lista_pokoj
&list_p);
        void zmien_date();
        void aktualna_data();
        void empty();
        void wyswietl();
};
```

4.2 Funkcje

W projekcie stworzyłem wiele funckji, które mają nazwę adekwatna do tego, co robią. Zrobiłem tak by kod był czytelny i można było łatwo znaleźć szukaną rzecz. Opiszę najważniejsze z nich.

```
void nowyrezerwacja(Lista_rezerwacja &list_r, Lista_pokoj &list_p, Lista_klient
&list_k);
```

Funkcja jako argumenty przyjmuje kolejno listę rezerwacji, pokojów i klientów. Umożliwia ona użytkownikowi na wprowadzenie nowej rezerwacji – podając rozpoczęcie i koniec rezerwacji, pokój w gość hotelowy chce zamieszkać oraz informacje o kliencie. Jeżeli klient znajduję się już w bazie danych wystarczy podać jego nazwisko, jeżeli nie to umożliwia dodanie nowego klienta, uprzednio podając jego dane. Jeżeli pokoju w danej dacie jest zajęty, rezerwacja nie zostaje dodana.

```
Lista_rezerwacja Lista_rezerwacja::usun_rezerwacje(Lista_rezerwacja &list)
```

Funkcja pozwalająca użytkownikowi na usuwanie wybranej przez niego rezerwacji - poprzez podanie daty oraz numer pokoju danej rezerwacji.

```
Klient* pop(string naz, Lista_klient &kli); / Pokoj* pop(int numer, Lista_pokoj &pok);
```

Funckcja zdefiniowana dla każdej klas Klient i Pokój, która po podaniu nazwiska/numer pokoju, zwraca wskaźnik na dany element listy.

```
Tabela_rezerwacji refresh(const Lista_rezerwacja &list_r, const Lista_pokoj &list_p);
```

Dane w kalendarzu rezerwacji często zostają zmieniane – zostają dodawane/usuwane rezerwacje lub zostaje zmieniona data. Funkcja refresh, ma za zadanie wygenerowania kalendarza rezerewacji, tak aby pokazywał aktualne dane.

```
int operator>(const Klient &kli);

Lista_klient operator+=(const Klient &kli);

friend istream & operator >>(istream &, Lista_klient &);

friend ostream & operator<<(ostream &, const Lista_klient &);
```

w programie prawie dla każdej klasy głównej zostały zdefiniowane powyższe operatory przeciążenia. Zostały wielokrotnie wykorzystywane w innych bardziej skomplikowanych funkcjach. Kolejno „>” - porównuje nazwiska i zwraca większy z nich, „+=” dodaje nowy element do list. „>>” i „<<” – operatory przypisania, głównie zdefiniowane w celu odczytu plików i wypisywania wiadomości na ekran.

5. Testowanie

Program został przetestowany dla różnych danych. Jeśli użytkownik zastosuje się do podanych w instrukcji punktów to program działa poprawnie, a jeśli nie to odpowiednio reaguje.

6.Wnioski

Program do obsługi hoteli był bardziej wymagający od poprzednich projektów za sprawą klas. Pisanie projektu obiektowego bardzo mi się podobało i było dla mnie dużym wyzwyzwaniem. Nauczyło mnie nowego podejścia do programowania, w programowaniu obiektowym najpierw należy dobrze przemyśleć cel swojego działania i wszystko dobrze zaplanować, a dopiero potem pisać kod, pozwala to na zoszczędzenie dużej ilości czasu.