

XVII. Funkcje w C++

17.1. Ogólna budowa funkcji

Do tej pory miałeś okazję niejednokrotnie wykorzystywać istniejące funkcje we własnych programach. Jak zapewne zauważyłeś, wykorzystywanie funkcji jest bardzo wygodne i proste w użyciu. Dobrze by było, gdyby równie proste było pisanie własnych funkcji... i tak w rzeczywistości właśnie jest.

Każda funkcja posiada trzy własności:

- zwraca dane (lub nie(**void**) jeśli tego nie chcemy);
- posiada swoją nazwę;
- może posiadać dowolną ilość parametrów wejściowych (lub może nie mieć żadnego(**void**), jeśli tego nie chcemy).

Jedynym wyjątkiem co do wartości zwracanego typu jest brak możliwości zwrócenia danych w postaci tablicy!

17.2. Definicja funkcji

Zgodnie z wymienionymi podpunktami ogólna budowa funkcji prezentuje się następująco:

```
zwracany_typ_danych nazwa_funkcji(parametry_funkcji)
{
    //blok funkcji → co funkcja będzie robiła
    return(wartosc_zwracana);
}
//Przykładowo:
int MojaPierwszaFunkcjaDodawania(int a, long b=10)
{
    //tu można umieścić jakieś jeszcze instrukcje, pętle itp.
    //można by powiedzieć, że funkcja to Twój podprogram, któremu dajesz
    //jakieś parametry a on na zakończenie daje Ci wynik
    //który później wykorzystujesz gdzieś dalej
    return(a + b);
    /* return(tablica[2]) → błąd jedyny wyjątek co do typów danych jakie mogą być zwrócone !!!*/
}
```

17.2.1. Co ważnego powinniśmy wiedzieć o funkcjach

Wszystkie zadeklarowane zmienne wewnątrz funkcji lub w parametrach funkcji są widoczne tylko i wyłącznie w obrębie funkcji(**zmienne funkcji są dla niej prywatne**). Ich modyfikacja nie wpływa na wartości zmiennych poza obrębem funkcji. Parametry funkcji możesz traktować jak zwykłe zmienne wewnątrz funkcji, które różnią się tylko tym, że mają przypisaną wartość podaną podczas wywołania funkcji. Pozostałe informacje przedstawiam w punktach:

- Każda funkcja musi mieć unikatową nazwę w obrębie całego programu (są odstępstwa ale o tym później);
- Parametry funkcji oddzielone są przecinkami;
- Słowo kluczowe **void** informuje kompilator, że funkcja nie zwraca żadnych danych.
- Słowo kluczowe **void** użyte w miejscu definiowania parametrów funkcji informuje kompilator, że funkcja nie przyjmuje żadnych parametrów;
- Parametrom funkcji można przypisywać domyślne wartości, które zostaną użyte wtedy gdy podczas jej wywołania parametr nie zostanie uzupełniony;
- Funkcja może zwracać wszystkie dopuszczalne typy – liczby całkowite, zmiennoprzecinkowe, wskaźniki, struktury i obiekty;
- Za pomocą parametrów możemy przekazywać również tablice.

17.3. Deklaracja i wywoływanie funkcji

Jeśli chcemy poinformować kompilator, że gdzieś w kodzie (lub poza nim np. w zewnętrznej bibliotece) znajduje się definicja funkcji musimy użyć do tego **deklaracji funkcji**(możesz się również spotkać z nazwą **prototyp funkcji**). Aby zadeklarować funkcję wystarczy napisać:

```
zwracany_typ_danych nazwa_funkcji(parametry_funkcji);
```

//Przykładowo

```
void nowaFunkcja(long,short,bool);
```

Pisząc **deklarację funkcji** możemy dodatkowo pisać nazwy dla parametrów. Są one jednak ignorowane przez kompilator i pełnią tylko i wyłącznie rolę informacyjną dla użytkownika. Jak pewnie zauważyłeś zamiast bloku głównego na końcu **deklaracji funkcji** jest **średnik**.

17.3.1. Przykłady zastosowania funkcji

//Funkcja potęgująca—przykład 1-----

```
#include<iostream>
```

```
#include<conio.h>
```

```
// deklaracja (prototyp) funkcji
```

```
int PodniesDoPotegiDrugiej(int liczba);
```

```
//funkcja główna -----
```

```
int main()
```

```
{
```

```
    using namespace std;
```

```
    int liczba;
```

```
    cout << "Podaj liczbe: ";
```

```
    cin >> liczba;
```

```
    cout << "Potega liczby = ";
```

```
    // wywołanie funkcji potegujacej
```

```
    cout << PodniesDoPotegiDrugiej(liczba);
```

```
    getch();
```

```
    return(0);
```

```
}
```

```
//definicja funkcji, czyli co funkcja ma robić
```

```
int PodniesDoPotegiDrugiej(int liczba)
```

```
{
```

```
    return (liczba *= liczba);
```

```
}
```

//Funkcja potęgująca—przykład 2-----

```
#include<iostream>
```

```
#include<conio.h>
```

```
// deklaracja(prototyp) funkcji
```

```
void WyswietlKomunikaty(int liczba, char semafor);
```

```
int PodniesDoPotegiDrugiej(int liczba);
```

```
//funkcja główna -----
```

```
int main()
```

```
{
```

```
    using namespace std;
```

```
    int liczba;
```

```
    // wywołanie funkcji
```

```
    WyswietlKomunikaty(0, 'N');
```

```
    cin >> liczba;
```

```
    // wywołanie funkcji
```

```

WyswietlKomunikaty(liczba, 'T');

getch();
return(0);
}
//definicja funkcji, czyli co funkcja ma robić
int PodniesDoPotegiDrugiej(int liczba)
{
return (liczba *= liczba);
}
//definicja funkcji, która wyświetla komunikaty i wynik
void WyswietlKomunikaty(int liczba, char semafor)
{
using std::cin;
using std::cout;
if (cin.fail())
cout << "Bład wprowadzonych danych !!!";
else if(semafor == 'T')
{
cout << "Potega liczby = ";
// wywołanie funkcji potegujacej
cout << PodniesDoPotegiDrugiej(liczba);
} else if (cin.good() && semafor == 'N')
cout << "Podaj liczbę: ";
}

```

17.3.2. Analiza przykładów

Jak zobaczysz oba przykłady wykonują dokładnie to samo. Pierwszą różnicą jest liczba użytych funkcji. W obu przypadkach przed funkcją główną użyliśmy **definicji(prototypu) funkcji**.

W pierwszym przykładzie:

```

// deklaracja (prototyp) funkcji
int PodniesDoPotegiDrugiej(int liczba);

```

informuje kompilator o tym, iż funkcja ma jeden parametr typu **int** i umożliwi wychwycenie błędu, w przypadku gdyby ten parametr nie został podany. Dodatkowo kompilator wie również o tym, iż funkcja zwróci wartość typu **int**. Analogicznie dla przykładu drugiego:

```

// deklaracja(prototyp) funkcji
void WyswietlKomunikaty(int liczba, char semafor);
int PodniesDoPotegiDrugiej(int liczba);

```

w tym wypadku, pierwsza funkcja pobiera dwa parametry typu **int i char** i nic nie będzie zwracać(**void**). Natomiast druga funkcja jest analogiczna jak w przykładzie pierwszym. Co warto wiedzieć o **definicji funkcji**:

- pozwala kompilatorowi sprawdzić czy funkcja zwraca prawidłową wartość,
- informuje kompilator czy została przekazana odpowiednia liczba parametrów,
- oraz czy typy parametrów są zgodne(przy brak zgodności, będzie próbować je prze-konwertować).

Polecam stosować definicje funkcji, pozwoli Tobie uniknąć wielu błędów, oraz ich zapobiegać.

Na koniec kilka informacji o strumieniu **cin** i użytych w zadaniu rozwiązaniach. Pierwsze użyte rozwiązanie **cin.fail()**, powoduje sprawdzenie czy w strumieniu są wprowadzone poprawne dane. Jeżeli strumień oczekuje liczby **int** to gdy wprowadzisz literę(znak) **char** to strumień ustawia błąd. Drugie rozwiązanie **cin.good()** sprawdza czy strumień jest gotowy do użytku(czyli czy nie zawiera błędu). Użycie drugiego rozwiązania jest trochę na wyrost i właściwie jest ono nie potrzebne w przykładzie(powinieneś sam wiedzieć dlaczego) jednak chciał pokazać jakie mogą być **stany strumienia**(stanów jest znacznie więcej) i jak można je wykorzystać(np. do sprawdzenia czy użytkownik podał to co chcieliśmy).

17.3. Funkcje a tablice

Mimo, że funkcje nie mogą zwracać tablic, to jednak mogą je pobrać, zobaczmy jak to się odbywa(wykorzystano przykład 8.3.4. Tablice dwuwymiarowe).

```
//Funkcja pobierająca tablice-----
#include<iostream>
#include<string>
#include<conio.h>
// deklaracja(prototyp) funkcji
void WyświetlKomunikaty(std::string tablica[], int index1, int tablica1[][7], int index2);
//funkcja główna -----
int main()
{
    using namespace std;
    const int max_znak = 30;
    const int LATA = 4;
    const int PANSTWA = 7;

    string panstwa[PANSTWA] =
    {
        "Wyspy Cooka",
        "Norfolk",
        "Watykan",
        "Pitcairn",
        "San Marino",
        "Kajmany",
        "Bermudy"
    };

    int ludnosc[LATA][PANSTWA] =
    {
        {21388, 1828, 932, 45, 29251, 45436, 65773},
        {21750, 2114, 821, 48, 29615, 46600, 66163},
        {21923, 2128, 824, 48, 29973, 47862, 66536},
        {11870, 2141, 826, 48, 30324, 49035, 67837},
    };

    WyświetlKomunikaty(panstwa, PANSTWA, ludnosc, LATA);

    getch();
    return 0;
}
//funkcja pobierająca dwie tablice panstwa i ludnosc
void WyświetlKomunikaty(std::string tablica[], int index1, int tablica1[][7], int index2){

    using std::cout;
    using std::endl;
```

```

cout << "\nLiczba ludności w latach 2006 - 2009.\n";
for (int licznik = 0; licznik < index1; ++licznik)
{
    cout << tablica[licznik] << ":\t";
    for (int rok = 0; rok < index2; ++rok)
        cout << tablica1[rok][licznik] << "\t";
    cout << endl;
}
cout << "\nZrodlo Wikipedia.pl";
return;
}

```

Funkcja pobiera dwie tablice(jedno i dwuwymiarową). W kursie wskaźniki podano fragmenty, które stanowią klucz do zrozumienia pobierania tablic przez funkcję - *Zauważmy, że wskaźnik ze zmiennej tablica i ze zmiennej tablica[0] jest taki sam.* , oraz

//wskaźniki

double *wsk_waga = waga; //nazwa tabeli = adres

//czyli

tablica == &tablica[0] //nazwa tabeli równa jest adresem pierwszego elementu

Funkcja **WyswietlKomunikaty(panstwa, PANSTWA, ludnosc, LATA);**, jako pierwszy parametr pobiera **nazwę tablicy(adres pierwszego elementu) i liczbę jej elementów(drugi parametr)**. Czyli de fa kto, funkcja nie pobiera **tablicy** lecz jej **adres pierwszego elementu**(wynika to z zapisy **waga[0] = *waga - Ponieważ tak działa kompilator C++, zamienia on zapis tab[10] dla bardziej czytelny sobie zapis *(tab + 10)**. Więc tak naprawdę operujemy na **wskaźniku i jego adresie**, natomiast liczba elementów służy do przechodzenia na kolejne adresy(elementów tabeli). Takie podejście pozwala nam oszukać kompilator

WyswietlKomunikaty(panstwa + 1, PANSTWA - 2, ludnosc + 2, LATA - 1); w takim przypadku początkiem tabeli **panstwa** będzie drugi element, a tabela będzie się składać tylko z 5 elementów. Polecam byś Sam spróbował modyfikować parametry podczas wywołania.

Przyjrzymy się teraz parametrowi tablicy dwuwymiarowej **int tablica1[][7]**. Dokładnie tablice jaką chcemy przetworzyć to tablica **int ludnosc[4][7]**. Nazwa tabeli(int ludnosc[0] == *ludnosc) jest zarazem jej pierwszym elementem, który składa się z **siedmiu** wartości typu **int**. To jest powód, dla którego nie musimy pobierać indeksu drugiego wymiaru tabeli, ponieważ jest on stały i nie ma znaczenia czy tabela ma 2 czy 100 elementów.

Reasumując:

- funkcja pobierająca tablice jako parametr tak naprawdę pobiera adres jej pierwszego elementu, który jest określany przez jej nazwę,
- drugi parametr jest to liczba elementów tabeli,
- funkcja bez problemu wczytuje tablice dwuwymiarowe, w tym przypadku liczbę kolumn jest stała dlatego musimy ją jawnie podać (np. int tablica[][10]),
- ponieważ funkcja pracuje na adresie, czyli operacje na wskaźnikach są jak najbardziej dozwolone (np. zapis int tablica1[][7] == int *tablica1[7]),
- funkcja dzięki wskaźnikowi wie gdzie jest tabela, oraz jakiego jest typu.

17.4 Budowa aplikacji opartej na funkcjach

Załóżmy, że masz za zadanie wykonać aplikację z interfejsem, która będzie służyła do obsługi okienka kasowego na lotnisku. Ma obsługiwać 6 połączeń do wybranych miast (np. Barcelona, Londyn, Paryż, New York, Moskwa, Tokio). By trochę uprościć sprawę dla każdej linii jest przeznaczonych 10 biletów. Zobacz jak mógłby wyglądać przykładowe rozwiązanie:

```
// aplikacja linie lotnicze---
#include <conio.h>
#include <string>
#include <iostream>
#include "ddtconsole.h"
const int BILETY = 10;
//definicje funkcji-----
void Kursor(int, int);
void Menu();
void Wyswietl_Dane(const std::string [][][BILETY], int, const int[], int);
int BazaLotow(std::string tabela1[][BILETY], int indeks1);
int OperacjeKasjera();
//funkcja główna -----
int main()
{
    using std::string;
    const int LINIE = 6;
    string LotyKlient[LINIE][BILETY];
    int max_wierszy = 0;
    // tabela która będzie przechowywać informacje dla których miast zostały zakupione bilety
    int tab_wiersze[6] = {-1, -1, -1, -1, -1, -1};
    Menu();
    //pętla zakończy się gdy Funkcja OperacjeKasjera zwróci wartość 27 (jest wartością klawisza ESC)
    while ((max_wierszy = OperacjeKasjera()) != 27){
        //sprawdza jaka opcja z menu została wybrana
        if (max_wierszy != 27 && max_wierszy != -1 && max_wierszy != 6){
            //jeżeli od 0 - 5 to wprowadzamy dane dla wybranego lotu(miasta)
            tab_wiersze[max_wierszy] = BazaLotow(LotyKlient, max_wierszy);
        }else if (max_wierszy == 6) {
            //jeżeli 6 to wyświetlamy dane dla wszystkich lotów
            Wyswietl_Dane(LotyKlient, LINIE, tab_wiersze, LINIE);
        }
        //powrót do menu
        Menu();
    }
    return 0;
}
//FUNKCJIE APLIKACJI -----
//funkcja wyświetlająca wprowadzone dane
void Wyswietl_Dane(std::string const tabela1[][BILETY], int indeks1, const int tabela2[], int indeks2)
{
    using namespace ddt::console;
    using std::cout;
    std::string Miasta[6] = {"Barcelona", "Londyn", "Paryz",
        "New York", "Moskwa", "Tokyo"};
    clrscr();
    gotoxy(5, 5);
    textcolor(10);
    cout << "Oto Dane dla linii lotniczych\n";
```

```

textcolor(11);
for (int i = 0; i < indeks2; i++) {
    //sprawdza czy w tabeli są dane jeżeli większe od -1 to dane są wprowadzone
    if (tabela2[i] != -1) {
        cout << "-----\n";
        textcolor(10);
        cout << Miasta[i] << "\n";
        textcolor(11);
        for (int j = 0; j <= tabela2[i]; j++) {
            cout << tabela1[i][j] << "\n";
        }
        cout << "-----\n";
    }
}
// pod Linuxem system(
//wyświetla komunikat systemowy i robi pauzę
system("pause");
}
//funkcja odpowiadająca za kursor (strzałkę) dokładnie jej pozycja na ekranie
void Kursor(int pion, int poziom)
{
    using namespace ddt::console;
    using std::cout;
    using std::endl;
    gotoxy(pion, poziom);
    textcolor(7);
    cout << "->" << endl;
}
// funkcja wprowadzająca dane do tabeli
int BazaLotow(std::string tabela1[][BILETY], int indeks1)
{
    using namespace ddt::console;
    using std::cin;
    using std::cout;
    std::string bufor;
    int bilety;
    int straznik = BILETY;
    int licz = 0;
    //główna pętla wprowadzania
    for (licz; licz < straznik; licz++){
        clrscr();
        gotoxy(20,8);
        textcolor(15);
        cout << "Literki 'q' i 'Q' - koncza wprowadzenie!\n\n";
        cout << "Imie i nazwisko klienta: ";
        //pobiera dana dla danego miasta-----
        getline(cin, tabela1[indeks1][licz]);
        if (tabela1[indeks1][licz] == "q" || tabela1[indeks1][licz] == "Q") {
            tabela1[indeks1][licz] = "\0";
            return licz - 1;
        }
        gotoxy(20,12);
        cout << "Liczba ujemna przerywa wprowadzanie\n";
        cout << "Ile biletow: ";
        (cin >> bilety).get();
        // sprawdza czy została podana liczba

```

```

if (!cin) //kontrola danych
{ // czyści strumień
  cin.clear();
  gotoxy(20,19);
  cout << "Nie podano liczby!!! Restart!!!";
  //robi pauzę na czas podanyc w milisekundach
  Sleep(4000);
  return licz - 1;
}
// jeżeli została podana liczba ujemna przerywa
else if (bilety <= 0){
  if (bilety == 0) {
    // ustawiono że dla zera bilet automatycznie jest równy 1
    bilety = 1;
  }else{
    // zeruje tabele bo wprowadzono imienie i nazwisko
    tabela1[indeks1][licz] = "\0";
    return licz - 1;
  }
  //sprawdza ile jest dostępnych biletów
if (straznik >= bilety){
  straznik -= bilety;
  bufor = ", ilosc biletow: ";
  tabela1[indeks1][licz] += bufor;
  bufor = 'a'; //wprowadzam znak by usunąć z bufora poprzedni łańcuch
  // zamienia liczbe int na łańcuch string
  std::sprintf((char*)bufor.c_str(), "%d", bilety);
  //opatologiczne rozwiązanie problemu wpisania do tabeli sring liczby 10
  if (bilety == 10) {
    tabela1[indeks1][licz] += bufor;
    tabela1[indeks1][licz] += '0';
  }else
  tabela1[indeks1][licz] += bufor;
} else{
  cout << "\nPrzekroczono liczbe biletow!!!\n"
  << "Zostaló ich tylko - " << straznik;
  //robi pauzę na czas podanyc w milisekundach
  Sleep(5000);
  licz -= 1;
}
}
return licz - 1;
}
//funkcja do nawigacji i obsługi menu ----
int OperacjeKasjera(){
  using namespace ddt::console;
  using std::cout;
  using std::cin;
  using std::endl;
  int znak;
  int pion = 17;
  int poziom = 12;
  // nawigacja kursorem-----
  do
  {
    Cursor(pion, poziom);

```

```
znak = getch();
/* cout << znak
   << " -- "
   << (int)znak;*/
// użycie strzałek to tak naprawdę dwa znaki dlatego należy usunąć pierwszy znak
if (znak == 224 || znak == 0)
znak = getch();
```

```
switch(znak)
{ // strzałka w dół
  case 80:
  {
    gotoxy(pion, poziom);
    cout << " " << endl;
    if (poziom == 14)
      poziom = 11;
    else
      poziom++;
  }break;
```

```
  case 72:
  { //strzałka w górę
    gotoxy(pion, poziom);
    cout << " " << endl;
    if (poziom == 11)
      poziom = 14;
    else
      poziom--;
  }break;
```

```
  case 77:
  { //strzałka w prawo
    gotoxy(pion, poziom);
    cout << " " << endl;
    if (pion == 17){
      pion = 37;
    }else
      pion -= 20;
  }break;
```

```
  case 75:
  { //strzałka w lewo
    gotoxy(pion, poziom);
    cout << " " << endl;
    if (pion == 37)
      pion = 17;
    else
      pion += 20;
  }break;
```

```
//jeśli ENTER to
  case 13:
  { // w zależności od pozycji strzałki taka opcja będzie wybrana
    if (poziom == 11 && pion == 17)
      return 0;
    else if (poziom == 12 && pion == 17)
      return 1;
```

```

else if (poziom == 13 && pion == 17)
return 2;
else if (poziom == 14 && pion == 17)
return 3;
else if (poziom == 11 && pion == 37)
return 4;
else if (poziom == 12 && pion == 37)
return 5;
else if (poziom == 13 && pion == 37)
return 6;
else if (poziom == 14 && pion == 37)
return 27;
}break;
/* default:
{
//clrscr();
cout << "error" << endl;
} break;*/
}
//Gdy ESC to koniec
}while(znak != 27);
return znak;
}
// menu aplikacji-----
void Menu(){
using namespace ddt::console;
using std::cout;
using std::endl;
//czyści ekran
clrscr();
//ustawia pozycję
gotoxy(30,8);
//zmienia kolor czcionki
textcolor(10);
cout << "Linie AirDDT" << endl;
gotoxy(20,11);
textcolor(11);
cout << "1 - Barcelona" << endl;
gotoxy(20,12);
textcolor(12);
cout << "2 - Londyn" << endl;
gotoxy(20,13);
textcolor(13);
cout << "3 - Paryz" << endl;
gotoxy(20,14);
textcolor(14);
cout << "4 - New York" << endl;
gotoxy(40,11);
textcolor(15);
cout << "5 - Moskwa" << endl;
gotoxy(40,12);
textcolor(13);
cout << "6 - Tokyo" << endl;
gotoxy(40,13);
textcolor(11);
cout << "7 - Wszystkie linie" << endl;

```

```

gotoxy(40,14);
textcolor(10);
cout << "8 - Przerwa" << endl;
gotoxy(30,20);
textcolor(11);
cout << "Esc - Zakoncz program" << endl;
gotoxy(30,21);
textcolor(11);
cout << "Enter - Zatwierdzenie" << endl;
}

```

Jak widać kod nie jest mały, przez co jego analiza jest trudna. Skupię się tylko na fragmentach kody by wyjaśnić jak działają niektóre mechanizmy. Zaczniemy od funkcji jest ich 5:

- void Kursor(int, int); → pobiera dwie wartości **int**, nic nie zwraca,
- void Menu(); → nic nie pobiera i nie zwraca,
- void Wyszwietl_Dane(const std::string [][][BILETY], int, const int[], int); → pobiera dwie **tablice**, jedną dwuwymiarową, oraz dwie wartości **int**(indeksy dla tych tablic), nic nie zwraca,
- int BazaLotow(std::string tabela1[][BILETY], int indeks1); → pobiera **tablice** i jej indeks, zwraca **int**(liczbę),
- int OperacjeKasjera(); → nic nie pobiera, zwraca **int**(liczbę).

Nazwy funkcji mówią same za siebie, dodatkowo w kodzie są komentarze, które rozwijają ich znaczenie.

Pierwszym aspektem, na który chciałbym zwrócić uwagę to funkcja:

```
void Wyszwietl_Dane(const std::string [][][BILETY], int, const int[], int);
```

Po pierwsze nie podano nazwy tabel ponieważ w deklaracji nie jest to wymagane, o czym zostało powiedziane na początku kursu. Jednak istotniejszy jest zapis, **const std::string [][][BILETY]** i **const int[]**. Ponieważ funkcja ta służy tylko do wyświetlania danych, dlatego nie powinna zmienić wartości danych w obu tabelach, które pobiera. Modyfikator **const** zapewnia nam właśnie taką sytuację. Powoduje, iż pobrane tabele dla tej funkcji będą stałymi i nie będzie można zmienić żadnych danych w nich występujących.

Drugim ciekawym aspektem jest fragment(zawarty w funkcji - char OperacjeKasjera()):

```
// użycie strzałek to tak naprawdę dwa znaki dlatego należy usunąć pierwszy znak
if (znak == 224 || znak == 0)
znak = getch();
```

W funkcji tej zastosowano rozwiązanie **znak = getch()**, które przypisuje wartość **int**, czyli kodu ASCII użytego klawisza. Dzięki temu wartość uzyskana jest sprawdzana w **switch-u** i możemy wykonać odpowiednią operację. Na koniec dla klawisza **ENTER(case 13)**, zostaje sprawdzona pozycja strzałki i w ten sposób wybrana odpowiednia opcja wybrania. Dzieje się to przez zwrócenie odpowiedniej wartości np. **return 0** odpowiada wybraniu Kursu do Barcelony i wprowadzenie danych klienta, natomiast **return 27**, oznacza zakończenie aplikacji(czyli wciśnięcie klawisza ESC).

Kolejnym fragmentem analizy jest wykorzystanie i użycie funkcji **int BazaLotow(std::string tabela1[][BILETY], int indeks1);char OperacjeKasjera()**; Zobaczmy jak do właściwie działa:

```
//pętla zakończy się gdy Funkcja OperacjeKasjera zwróci wartość 27 (jest wartością klawisza ESC)
while ((max_wierszy = OperacjeKasjera()) != 27){
```

```

//sprawdza jaka opcja z menu została wybrana
if (max_wierszy != 27 && max_wierszy != -1 && max_wierszy != 6){
    //jeżeli od 0 - 5 to wprowadzamy dane dla wybranego lotu(miasta)
    tab_wiersze[max_wierszy] = BazaLotow(LotyKlient, max_wierszy);
}else if (max_wierszy == 6) {
    //jeżeli 6 to wyświetlamy dane dla wszystkich lotów
    Wyswietl_Dane(LotyKlient, LINIE, tab_wiersze, LINIE);
}
//powrót do menu
Menu();
}

```

Na początku jest wywołana funkcja **OperacjeKasjera()**, która ma za zadanie zwrócić wartość liczbową. Gdy zwróci wartość od 0 do 5 to zostanie wywołana kolejna funkcja **tab_wiersze[max_wierszy] = BazaLotow(LotyKlient, max_wierszy)**; Zadaniem tej funkcji jest wprowadzenia danych do tablicy **LotyKlient**, są w niej przechowywane dane klienta. Następnie w zależności od liczby wprowadzonych klientów funkcja ta zwraca wartość od 0 do 10(zależy to od liczby klientów i liczby zakupionych przez nich biletów). Tablica **tab_wiersze[max_wierszy]** przechowuje indeks miasta, dla którego zostały wprowadzone dane oraz liczbę wprowadzonych klientów. Dzięki temu rozwiązaniu nie dopuścimy do sytuacji, w której będziemy wyświetlać informacje o miastach, które nie zostały wprowadzone. Funkcja odpowiedzialna za wyświetlanie danych zostanie wywołana, gdy funkcja **OperacjeKasjera()** zwróci **wartość 6**. Aplikacja zakończy działanie gdy w/w funkcja zwróci **wartość 27**.

To właściwie koniec jeśli chodzi o analizę przykładu. Przykład nie jest skomplikowany pod względem programistycznym(stopnia zaawansowania). Jednak zdaję sobie sprawę, że osobie uczącej się może przysporzyć wiele kłopotów ze zrozumieniem ze względu na objętość. Przedstawiłem tutaj kilka ciekawych rozwiązań, które mogą ci się przydać podczas Swojej przygody z programowaniem.

17.5. Ćwiczenia

Zadania będą się oparte na ostatnim przykładzie w tym kursie.

1. Podaj inną kombinację klawiszy, która umożliwi sterowaniem strzałki(góra, dół, prawo, lewo).
2. Aplikacja ma pewną wadę, gdy wybieram punkt 8 kończy działania, napisz funkcję i wkomponuj ją w kod by robiła Pauzę. Użytkownik podaje czas długości pauzy. Wyświetl komunikat o tym, że jest pauza i ile ona potrwa, najlepiej w formacie zrozumiałym dla każdego z nas(np. minutach).
3. Z użyciem getch(), napisz prosty programik, zawierający funkcję, która będzie pokazywać kod ASCII klawisza, który został naciśnięty. Aplikacja ma się zakończyć, gdy zostanie naciśnięty klawisz ← **Backspace**.