Programowanie obiektowe

Ćwiczenie nr. 3

Podstawy obsługi plików. Wprowadzenie do wektorów. Obliczenia.

dr inż. Adam Wolniakowski

1. Cel ćwiczenia

Celem ćwiczenia jest zapoznanie się z podstawami obsługi plików (odczyt i zapis) oraz z wykorzystaniem wbudowanych struktur danych (wektory) do przechowywania danych oraz wykonywania obliczeń.

2. Informacje

2.1. Jak odczytać argument wiersza poleceń?

Sygnatura funkcji *main* powinna mieć postać:

```
int main(int argc, char* argv[])
```

Zmienna *argc* przechowuje ilość argumentów wiersza poleceń, natomiast tablica *argv* zawiera kolejne argumenty. W przypadku, kiedy program ma wymagać jednego argumentu, można posłużyć się następującą konstrukcją (wewnątrz funkcji *main*):

```
if (argc != 2) {
     throw runtime_error(”Należy podać nazwę pliku!”);
}
string nazwa_pliku = argv[1];
cout << ”Odczytuje plik: ” << nazwa_pliku << endl;
```

2.2. Jak otworzyć plik do odczytu i wypisać / wykorzystać każdą jego linię?

Należy wykorzystać bibliotekę <fstream>. Plik otwieramy w następujący sposób:

```
ifstream plik(”nazwa_pliku.txt”);
```

Aby odczytać linie pliku po kolei można wykorzystać następujący fragment kodu:

```
string linia;
while (getline(plik, linia)) {
     // zrob cos z pojedyncza linia, na przyklad:
     cout << linia << endl;
}
```

Po zakończeniu odczytu pliku należy go zamknąć:

```
plik.close();
```

2.3. Jak z pojedynczej linii odczytać dane liczbowe?

Plik .csv ma następującą postać:

```
0.0, 2.0
0.1, 2.1
0.2, 0.3
…
```

W każdej linii znajdują się liczby rozdzielone przecinkami. Pojedynczą linię można zamienić na obiekt typu *stringstream* (#include <sstream>), z którego można odczytać dane przy pomocy operatora >>. W tym celu można wykorzystać następujący fragment kodu:

```
string linia; // to mamy dane

stringstream ss(linia); // inicjalizujemy stringstream zawartoscią linii
double x1, x2; // w tych zmiennych zapiszemy odczytane wartości
ss >> x1; // odczytujemy z linii pierwszą liczbę
ss.ignore(); // ignorujemy przecinek
ss >> x2; // odczytujemy z linii drugą liczbę
```

2.4. W jaki sposób przechowywać dane w programie?

W pojedynczej linii pliku .csv zapisana jest wartość czasu (t) oraz wartość jaką w tej chwili ma sygnał (x). Wartości te charakteryzują próbkę i sensowne jest zapisać je w postaci pojedynczej struktury, np.:

```
struct Probka {
    double t;
    double x;
};

Probka probka1 = { 0.1, 0.2 }; // tak tworzymy nową próbkę w postaci struktury
```

Można również wykorzystać funkcjonalność języka C++ i zapisać te dwie liczby w postaci pary:

```
typedef std::pair<double, double> Probka; // definiujemy typ o nazwie probka przechowujacy parę liczb typu double
Probka probka1(0.1, 0.2); // tworzymy zmienna typu Probka o nazwie probka1
cout << probka1.first << endl; // wypisze pierwsza liczbe
```

```
cout << probka1.second << endl; // wypisze druga liczbe
```

Ponieważ z góry nie wiemy, ile próbek jest zapisanych w pliku, nie możemy ich wszystkich zapisać w postaci tablicy statycznej (o znanym rozmiarze). Należy wykorzystać wbudowane w języku C++ *wektory* (można je rozumieć jako tablice o zmiennym rozmiarze):

```
std::vector<Probka> dane; // dynamiczna tablica przechowująca próbki

Probka probka1(0.1, 0.2); // tworzymy nową próbkę
dane.push_back(probka1); // i dodajemy ją do tablicy

dane.push_back(Probka(0.2, 0.3)); // można też dodać próbkę w ten sposób

cout << dane[0].first << ", " << dane[0].second; // aby odczytać element tablicy używamy nawiasów kwadratowych
```

2.5. Jak przeiterować zawartość wektora?

Długość wektora można sprawdzić wykorzystując jego metodę `.size()`. Można więc użyć tradycyjnej pętli for:

```
for (int i = 0; i < wektor.size(); ++i) {
    // zrób coś z i-tym elementem wektora, np.
    cout << wektor[i] << endl;
}
```

Można również wykorzystać składnię wersji 2011 języka C++:

```
for (auto& x : wektor) {
  // x jest teraz kolejnym elementem wektora
}
```

2.6. Jak otworzyć plik do zapisu i zapisać w nim dane?

Plik do zapisu otwieramy w następujący sposób:

```
ofstream plik(„nazwa_pliku.txt");
```

Z pliku można korzystać tak jak z każdego innego strumienia wyjściowego, np.:

```
plik << „Witaj świecie!" << jakas_liczba << endl;
```

Na koniec plik należy koniecznie zamknąć:

```
plik.close();
```

3. Zadanie

Należy napisać program, który wczyta sygnał zapisany w pliku .csv o nazwie zadanej w postaci argumentu wiersza poleceń, wypisze jego zawartość na ekran, a następnie obliczy podstawowe charakterystyki sygnału: długość, wartość minimalną, średnią i maksymalną. Dodatkowo, sygnał zostanie zmodyfikowany, a następnie zapisany do pliku *out.csv.*

Należy utworzyć nowe repozytorium (np. cw_3). Projekt można wykonać w następujących krokach (należy wykonać commit po zakończeniu każdego etapu):

3.1. Stworzyć podstawową strukturę projektu / programu (funkcja main).

3.2. W funkcji *main* dodać odczytywanie i wypisanie na ekran pierwszego argumentu wiersza poleceń.

3.3. Zdefiniować typ opisujący pojedynczą próbkę (typedef jako struktura lub para) oraz typ opisujący tablicę danych (wektor próbek).

3.4. Utworzyć funkcję o nazwie np. *wczytaj* przyjmującą jako argument string zawierający nazwę pliku oraz zwracającą wczytany z pliku wektor próbek.

3.5. Utworzyć funkcję o nazwie np. *wypisz*, przyjmującą jako argument wektor danych i wypisującą go na ekran. Należy wypróbować wypisywanie wczytanych danych na ekran.

3.6. Utworzyć funkcję o nazwie np. *zapisz*, przyjmującą dwa argumenty: wektor danych oraz nazwę pliku. Funkcja powinna zapisywać dane do wskazanego pliku. Należy wypróbować działania programu zapisując wczytane dane do pliku o nazwie *out.csv.*

3.7. Należy utworzyć funkcję o nazwie np. *obliczDlugosc* przyjmującą jako argument wektor danych. Funkcja powinna zwracać wartość typu double stanowiącą różnicę pomiędzy największą a najmniejszą wartością *t* dla wszystkich próbek. Sprawdzić działanie programu.

3.8. Należy utworzyć funkcje o nazwach np. *obliczMinimum* i *obliczMaksimum*, przyjmujące jako argument wektor danych i zwracające odpowiednio: minimalną i maksymalną wartość sygnału w postaci liczb typu double. Sprawdzić działanie programu.

3.9. Utworzyć funkcję o nazwie np. *obliczSrednia*, która przyjmuje jako argument wektor danych oraz zwraca w postaci liczby double obliczoną średnią wartość jego elementów. Sprawdzić działanie programu.

3.10. Utworzyć funkcję o nazwie np. *calkuj*, która przyjmuje jako argument wektor danych. Funkcja powinna obliczać wartość całki otrzymanego sygnału. Sugeruje się użyć metody trapezów (pseudokod):

```
calka ← 0
N ← ilość próbek
for i od 0 do N-1:
     dt ← probka[i+1].t – probka[i].t
     dpole ← (probka[i].x + probka[i+1].x) * dt / 2
     calka ← calka + dpole
return calka
```

3.11. Należy przenieść opracowane funkcje do oddzielnego modułu (utworzyć plik nagłówkowy i plik .cpp opatrzone odpowiednią dokumentacją). Na tym etapie program powinien wykonywać następujące czynności:

(1) Wczytaj dane.
(2) Wypisz długość sygnału.
(3) Wypisz minimum i maksimum sygnału.
(4) Wypisz średnią wartość sygnału.
(5) Wypisz wartość całki sygnału.
(6) Zapisz sygnał w pliku *out.csv.*

3.12. Wprowadzić modyfikacje zasugerowane przez prowadzącego.