Lista 4

Zestaw I

Zadanie 1.
Zaprojektować i zaimplementować funkcję `wpisz_do_pliku_a`, kótra wpisuje do pliku `a.txt` 10 razy łańcuch "Hello World". łańcuchy są rozdzielone znakami nowych wierszy.
Zadanie 2.
Zaprojektować i zaimplementować funkcję `wpisz_do_pliku`, która przyjmuje trzy argumeny: ciąg znaków `nazwa_pliku`, reprezentujący nazwę pliku; ciąg znaków `zawartosc` i liczbę całkowitą nieujemną `ile`, domyślnie przyjmujący wartość 10.

Funkcja ma dopisywaą do pliku `nazwa_pliku` podany jako drugi argument ciąg `zawartosc`, powtarzając operację `ile` razy i oddzielając każde dopisanie znakiem nowej linii.

Wywołanie funkcji

```
wpisz_do_pliku("a.txt","Hello world",3);
```

powinno spowodować dopisanie do pliku `a.txt` następującej zawartości:

```
Hello world
Hello world
Hello world
```

Zadanie 3.
Zaprojektować i zaimplementować przeciążoną funkcję `wpisz_do_pliku`, która zamiast argumentu `nazwa_pliku` przyjmuje obiekt reprezentujący wyjściowy strumień plikowy.
Zadanie 4.
Zaprojektować i zaimplementować funkcję `wpisz_do_pliku_v`, która powinna działać identycznie jak funkcja `wpisz_do_pliku` z drobnymi wyjątkami:

- po ostatnim dopisaniu na standardowe wyjście (`cout`) powinien zostać wyprowadzony komunikat `"Skonczylem"`;
- co piąte dopisanie ciągu znaków powinno spowodować wysłanie na stardardowe wyjście błędów (`cerr`) komunikatu `"Kolejne 5 napisow"`.

Przetestować powyższą funkcję, wykorzystując przekierowanie strumieni w konsoli.

Zestaw II

Zadanie 1.

Zaprojektować i zaimplementować funkcje:

- `utworz` – tworzącą w pamięci dynamicznej tablicę dwuwymiarową liczb rzeczywistych, a następnie zerującą jej wszystkie elementy,
- `zapisz` – zapisującą w pliku wyjściowym wyniki operacji na macierzach (z zachowaniem formatu wejściowego),
- `usun` – zwalniającą zasoby przydzielone na stercie,

- `czytaj` – pobierającą z pliku wejściowego rozmiary macierzy oraz wartości ich elementów zapisanych w określonym poniżej formacie,

```
m n
x_11 x_12 ... x_1n
x_21 x_22 ... x_2n
...
x_m1 x_m2 ... x_mn

p q
y_11 y_12 ... y_1q
y_21 y_22 ... y_2q
...
y_p1 y_p2 ... y_pq
```

- `suma` – wykonującą operację dodawania macierzy pobranych jako argumenty tej funkcji oraz umieszczającą wynik działania w macierzy utworzonej wewnątrz tej funkcji,

- `roznica` – wykonującą operację odejmowania macierzy pobranych jako argumenty tej funkcji oraz umieszczającą wynik działania w macierzy utworzonej wewnątrz tej funkcji,

- `iloczyn` – wykonującą operację mnożenia macierzy pobranych jako argumenty tej funkcji oraz umieszczającą wynik działania w macierzy utworzonej wewnątrz tej funkcji,

- `transponowanie` – wykonującą operację transponowania macierzy pobranej jako argument wywołania tej funkcji oraz umieszczającą wynik działania w macierzy utworzonej wewnątrz tej funkcji.

Nazwę pliku wejściowego i wyjściowego przekazać jako parametr wywołania programu.

W zadaniu należy pamiętać o sprawdzaniu stanu strumienia podczas pracy z danym strumieniem.

Rozwiązanie zadania należy przetestować podanym poniżej ciągiem instrukcji:

```
//...
ifstream fin;
//...
ofstream fout;
//...

double** A = 0, ** B = 0, ** C = 0;
unsigned int n, m, p, q;

if (czytaj(fin, A, m, n, B, p, q)) {
  if(!suma(A, m, n, B, p, q, C))
    cerr << "Macierze maja nieprawidlowe wymiary" << endl;
  else{
    fout << "suma macierzy" << endl;
```

```
    zapisz(fout, C, m, n);
    usun(C, m);
  }
  if(!roznica(A, m, n, B, p, q, C))
    cerr << "Macierze maja nieprawidlowe wymiary" << endl;
  else{
    fout << "roznica macierzy" << endl;
    zapisz(fout, C, m, n);
    usun(C, m);
  }
  if(!iloczyn(A, m, n, B, p, q, C))
    cerr << "Macierze maja nieprawidlowe wymiary" << endl;
  else{
    fout << "iloczyn macierzy" << endl;
    zapisz(fout, C, m, q);
    usun(C, m);
  }
  fout << "macierz przed transponowaniem" << endl;
  zapisz(fout, B, p, q);
  fout << "macierz transponowana" << endl;
  transponowanie(B, p, q, C);
  zapisz(fout, C, q, p);
  usun(C, q);
}
  usun(B, p);
  usun(A, m);
//...
```

Nie wolno zmieniać sposobu wywoływania poszczególnych funkcji.
Komentarze należy zamienić na brakujące fragmenty kodu źródłowego.

Zadanie 2.

Zaprojektować i zaimplementować funkcję realizującą następujące zadania:

- odczytanie danych ze strumienia plikowego przekazanego, jako parametr wywołania funkcji (wczytanie słowa „END” kończy wczytywanie danych),
- wyselekcjonowanie adresów email (sprawdzenie czy występuje znak @),
- zapisanie tych adresów (rozdzielonych średnikiem) do strumienia plikowego przekazanego jako drugi parametr funkcji.

Fragment kodu testującego:

```
ofstream plik("out.txt");
fun1(cin, plik);
```

Zadanie 3.

Zaprojektować i zaimplementować funkcję wyszukującą w pliku tekstowym osób o podanym numerze GaduGadu oraz wypisującą na standardowe wyjście jej imię i wiek (liczba

lat w bieżącym roku). Dane pobrać z pliku z danymi o następujcym formacie:
`numer_gg adres@email rok_urodzenia imie;`
np.

```
12345 ala@ma.kotka.pl 1999 Alicja;
2345 ola@ma.pieska.eu 2000 Aleksandra;
```

Fragment kodu testującego:

```
ifstream adresy1("adresy.txt");
if (!fun2(adresy1, 2345)) cerr << "Nie znaleziono";
```

Zadanie 4.

Zaprojektować i zaimplementować funkcję pobierającą dane z pliku tekstowego w formacie opisanym w poprzednim zadaniu i zapisującą do pliku te dane w formacie XML według podanego wzoru:

```
<osoby>
      <osoba rok_urodzenia="1999">
            <imie>Alicja</imie>
            <email>ala@ma.kotka.pl</email>
            <gg>12345</gg>
      </osoba>
      <osoba rok_urodzenia="2000">
            <imie>Aleksandra</imie>
            <email>ola@ma.pieska.eu</email>
            <gg>2345</gg>
      </osoba>
</osoby>
```

Dane zapisać na dwa różne strumienie (przekazane do funkcji jako parametry) z podziałem ze względu na płeć (na podstawie ostatniej litery imienia).

Fragment kodu testującego:

```
ifstream adresy2("adresy.txt");
ofstream kobiety("kobiety.txt");
ofstream mezczyzni("mezczyzni.txt");
fun3(adresy2, kobiety, mezczyzni);
```

Zadanie 5.

Zaprojektować i zaimplementować funkcje:

- zwracającą liczbę wszystkich wystąpień danego symbolu (pobranego z linii wywołania programu) w pliku tekstowy,
- filtrującą zawartość pliku wejściowego w celu znalezienia wszystkich wyrazów, w których dany symbol występuje dwukrotnie; wyrazy te zapisać w pliku wyjściowym (każdy wyraz w nowym wierszu); przyjąć, że wyrazem jest każdy ciąg znaków oddzielony spacją, tabulacją, końcem linii oraz końcem pliku.

Nazwy plików przekazać, jako parametry wywołania programu. **Do rozwiązania zadania należy użyć typu** `std::string`.

Zestaw III

W zestawie tym będzie wykorzystywany wektor (w ujęciu matematycznym), którego format zapisu w pliku jest następujący:

```
rozmiar
pierwszy_element
drugi_element
...
rozmiar-1_element
```

Przykładowo, plik o zawartości:

```
4
0
1.5
3.5
7
```

reprezentuje wektor liczb rzeczywistych (takie będą używane w zadaniu) `[0; 1.5; 3.5; 7]`.

Zadanie 1.

Zaprojektować i zaimplementować funkcję `zapisz_wektor`, która przyjmuje trzy argumenty: `plik`, reprezentujący wyjściowy strumień plikowy; `wek`, reprezentujący wektor (jest to dynamiczna tablica liczb rzeczywistych, która należy przekazać przez wskaźnik) i liczbę całkowią nieujemną `rozmiar`, reprezentującą rozmiar wektora.

Funkcja powinna zapisywać podany wektor do pliku, zgodnie z formatem podanym na początku tego zestawu zadań.

Zadanie 2.

Zaprojektować i zaimplementować funkcję `wczytaj_wektor`, która przyjmuje dwa argumenty: `plik` (wejściowy strumień plików) i `rozmiar` (rozmiar wektora; argument powinien być możliwy do odebrania po wywołaniu funkcji). Funkcja zwraca wskaźnik na utworzoną w środku dynamiczną tablicę, reprezentującą wczytany z pliku wektor.

Przykładowo, poniższy kod (`plik` jest wejściowym strumieniem plików, zawierający uchwyt do otwartego pliku `a.txt`) :

```
unsigned rozmiar = 0;
double* wektor = wczytaj_wektor(plik,rozmiar);
```

dla następującej zawartości pliku `a.txt`:

```
3
0.5
5
8.0
```

powinien spowodować utworzenie tablicy `wektor` o zawartości `[0.5; 5; 8.0]`, dodatkowo ustawiając wartość zmiennej `rozmiar` na 3.

Zadanie 3.

Zaprojektować i zaimplementować funkcję `dodaj_wektory`, która przyjmuje pięć argumentów:

- `wek1` (tablica dynamiczna, zawierająca wartości pierwszego wektora);
- `wek2` (tablica dynamiczna, zawierająca wartości drugiego wektora);

- `roz1` (rozmiar pierwszego wektora);
- `roz2` (rozmiar drugiego wektora);
- `roz3` (przekazany przez referencję rozmiar wektora wynikowego).

i zwracająca wskaźnik na utworzoną wewnątrz funkcji dynamiczną tablicę liczb rzeczywistych.

Funkcja powinna dodawać wektory, podane jako argumenty i zwracać rezultat dodawania jako tablicę dynamiczną. Jeżeli `roz1!=roz2`, to na standardowe wyjście błędów należy wypisać komunikat `"Zle wymiary"`, zwrócić pusty wskaźnik i ustawić wartość `roz3` na `0`.

Zadanie 4.
Zaprojektować i zaimplementować funkcję `zapisz_inzyniersko`, działającą prawie tak samo jak funkcja `zapisz_wektor`. Różnica polega na zapisie elementów wektora - powinny być zapisane w notacji inżynierskiej (`ios::scientific`).

Zestaw IV

Zadanie 1.
Zaimplementować funkcję `dolacz_naglowki`, przyjmująca trzy argumenty: wyjściowy strumień plikowy, tablicę napisów i rozmiar tablicy. Funkcja tworzy (zastępuje) plik i wpisuje do niego następujące dane:

```
#include<[zerowy napis]>
#include<[pierwszy napis]>
#include<[drugi napis]>
...
#include<[n-1 napis]>
```

Zadanie 2.
Zaimplementować funkcję `dolacz_funkcje`, przyjmującą szeć argumentów: wyjściowy strumień plikowy, napis (typ wartości zwracanej z funkcji), drugi napis (nazwa funkcji), trzeci napis (typ argumentów funkcji), czwarty napis (ciało funkcji) oraz liczbę całkowitą nieujemną (liczba argumentów). Wywołanie powyższej funkcji powinno dopisać do strumienia plikowego kod źródłowy funkcji według poniższego schematu:

```
[wartość zwracana] [nazwa_funkcji]([typ argumentu] arg1,
      [typ argumentu] arg2, ...,
      [typ argumentu] argn-1) {
[cialo funkcji]
}
```

Przykładowo, wywołanie funkcji:

```
dolacz_funkcje(plik,"int","daj_piec","int","return 5;", 2)
```

powinno dopisać do pliku następujący ciąg znaków:

```
int daj_piec(int arg0, int arg1) {
return 5;
}
```

Zadanie 3.
Zaimplementować funkcję `dolacz_main`, przyjmującą dwa argumenty: wyjściowy strumień plików i napis (ciało funkcji main). Wywołanie funkcji `dodaj_main` powinno dodać do wyjściowego strumienia plików podany jako argument ciąg znaków.
Przykładowy kod testujący.
Jeżeli w funkcji `main` pisanego programu (reprezentacji zestawu IV) zostanie wykorzystany poniższy kod testujący

```
//...
ofstream plik;
plik.open(argv[1]);
string tab[] = {"iostream","cstring"};
dolacz_naglowki(plik,tab,2);
plik << endl;
dolacz_funkcje(plik,"int","daj_piec","int","return 5;", 2);
dodaj_main(plik,"cout << \"Hello world\" << endl;");
plik.close();
//...
```

to uruchomienie programu z argumentem `pp.cpp` (warto sprawdzić czy lista argumentów wywołania programu jest zgodna z oczekiwaniami) spowoduje wygenerowanie poniższego pliku `pp.cpp`:

```
#include<iostream>
#include<cstring>

int daj_piec(int arg0, int arg1) {
return 5;
}
int main() {
cout << "Hello world" << endl;
}
```

Zestaw V

Zadanie 1.
W ramach tego zestawu należy napisać program (konstrukcja, użyte funkcje itp. dowolne), będący bardzo prostą, plikową bazę danych, zawierająca skarbnicę "wyjątkowych odzywek, pochodzących z internetu". Plik danych jest zapisany w poniższym formacie (zakładamy, że każdy z tekstów nie zawiera znaków nowej linii i nie zawiera polskich znaków):

```
n
tekst0
tekst1
tekst2
...
tekstn-1
```

Przykładowo, poniższy plik:

```
5
```

```
O skurczebyk
To je amelinium, tego nie pomalujesz
Ale urwal
Jestes zwycienscom
Daj kamjenja
```

reprezentuje 5 napisów.

Program powinien działać następująco:

- odpowiednio wczytuje (pobranie rozmiaru, utworzenie tablicy napisów, uzupełnianie jej) plik, podany jako argument wywołania programu. Należy sprawdzić czy lista argumentów jest zgodna z oczekiwaniami. Jeżeli dany plik nie może zostać wczytany (bo np. nie został utworzony), to na standardowe wyjście błędów należy wysłać odpowiedni komunikat i zakończyć działanie programu z wartością `-1`;
- umożliwia dodanie nowej odzywki. Dodanie nowej odzywki wymusza skopiowanie tablicy napisów o rozmiarze `n` do tablicy o rozmiarze `n+1` i dopisanie odzywki na koniec nowo utworzonej tablicy;
- umożliwia usunięcie ostatniej odzywki. Usunięcie odzywki wymusza skopiowanie tablicy napisów o rozmiarze `n` do tablicy o rozmiarze `n-1` (z pominięciem ostatniej odzywki);
- umożliwia zapis odzywek, w formacie przestawionym powyżej, do pliku podanego jako argument wywołania programu (zarówno odczyt danych, jak i zapis dotyczy tego samego pliku);
- umożliwia proste wyszukiwanie odzywek. Tj. program oczekuje na podanie przez użytkownika ciągu znaków, który jest następnie wyszukiwany we wszystkich odzywkach. Jeżeli dany ciąg zostanie znaleziony - należy wypisać odzywkę, która zawiera podany ciąg znaków;
- pozwala użytkownikowi monitorować stan programu, tj. wypisać listę wszystkich odzywek.

Powyższą funkcjonalność powinna być dostępna dla użytkownika za pomocą bardzo prostego menu - na podobnej zasadzie, jak implementowany przez Państwa kalkulator na jednym z poprzednich zajęć.