Wykład 9

- Pliki cz.3 (`getline(string)`)
- Typ wyliczeniowy `enum`

Przykład 4 – funkcja get(char &)

Napisz funkcję, która dla pliku o nazwie podanej przez pierwszy parametr funkcji oblicza prawdopodobieństwo z jakim znak podany przez drugi parametr występuje w pliku
(tj. *liczbę wystąpień znaku / liczbę wszystkich znaków*).
Jeśli operacje plikowe nie powiodą się lub plik jest pusty, wartością funkcji ma być –1.

```
float prawdopZnaku(string n, char szuk){
    ifstream we;
    we.open(n);
    if(!we.is_open())
        return -1;
    char zn;
    int ilew=0, ileszuk=0;
```

Przykład 4 – funkcja get(char &)

```
    while(we.get(zn)){
        ilew++;
        if(zn == szuk)
            ileszuk++;
    }
    we.close();
    if(ilew)    return ileszuk*1.0f/ilew;
    else //plik pusty
       return -1;
}

int main()
{   float p = prawdopZnaku("t.txt",'\n');
    if(p<0)  cout<<"blad operacji plikowych"<<endl;
    else     cout<<"prawdopodobienstwo = "<< p <<endl;
    return 0;
}
```

getline(string)

Wprowadzanie danych do obiektów `string`:

```
string s;
cin >> s; //wczytuje słowo
getline(cin, s); //wczytuje linię, pomija \n
```

Mamy dwie wersje funkcji `getline()`:

```
istream& getline (istream& is, string& str);
istream& getline (istream& is, string& str, char delim);
```

druga pozwala na określenie znaku kończącego wczytywanie. Powyższe wersje funkcji dla klasy `string` automatycznie dopasowują rozmiar obiektu string do liczby zapisywanych znaków

```
string lname;
cin>>lname; //potrafi wczytać bardzo długie słowo
getline(cin, lname); //wczytuje linię, obcina '\n'
```

getline(string)

Istnieją pewne ograniczenia na długość wprowadzanego ciągu do obiektu `string`:

1) maksymalny dopuszczalny rozmiar przechowywany w stałej `string::npos` (zazwyczaj jest to maksymalna wartość typu `unsigned int`)

2) ilość pamięci dostępnej w programie

Funkcja **`getline()`** klasy **`string`** wczytuje wprowadzane znaki i zapisuje je w obiekcie klasy `string`, dopóki nie zajdzie jedna z trzech okoliczności:

a) program natrafi na koniec pliku

b) program natrafi na znak kończący wczytywanie, domyślnie jest to `\n`. Znak ten jest pobierany ze strumienia, ale nie jest zapisywany do obiektu `string`

c) zostanie wczytana maksymalna liczba znaków, czyli mniejsza z wartości: `string::npos` oraz liczby bajtów pamięci dostępnych na zapisanie ciągu.

Funkcja **`operator>>()`** w klasie **`string`** czyta słowo, czyli omija wiodące znaki białe, a potem dochodzi do znaku białego i zostawia go w kolejce wejścia.

Powyższych funkcji klasy `string` możemy używać do wczytywania plików.

Przykład 5 - getline(string)

Napisz funkcję, która policzy liczbę wszystkich linii oraz liczbę pustych linii w pliku tekstowym, którego nazwa jest parametrem funkcji. Wynikiem funkcji jest liczba linii lub -1, jeśli otwarcie pliku nie powiodło się. Liczbę pustych linii przekazujemy dodatkowym parametrem. Jeśli linia zawiera jakieś znaki białe, to nie jest pustą linią.

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;
int ileLinii(string nplik, int &ilpuste)
{
    ifstream pwe(nplik);
    if(!pwe.is_open()) return -1;
    //plik istnieje i jest otwarty
```

Przykład 5 - getline(string)

```
    string linia;
    int ile=0; //liczba wszystkich linii w pliku
    ilpuste=0; //liczba pustych linii - zerujemy parametr
    while(getline(pwe,linia))// dopóki wczytano linie
     {  ile++;
        //sprawdzamy czy pusta
        if (linia.empty()) //lub if(linia.size()==0)
            ilpuste++;
      }
    pwe.close();
    return ile;
}
```

Przykład 5 - getline(string)

```
int main()
{   string nazwaplik;
    cout << "Podaj nazwe pliku: ";
    cin >> nazwaplik;
    int ilp;
    int ill = ileLinii(nazwaplik, ilp);
    if(ill<0) cout<<"Blad otwarcia pliku"<<endl;
    else{
      cout<<"W pliku "<<nazwaplik<<" mamy "
          <<ill<<" wszystkich linii "
          <<endl<<"w tym  pustych: "<<ilp<<endl;
    }
    return 0;
}
```

enum – typ wyliczeniowy

Wyliczenie jest typem zdefiniowanym przez zbiór (zwykle niewielki) nazwanych stałych całkowitych**.**

Na przykład

```
enum Dni {pon, wto, sro, czw, pia, sob, nie};
```

definiuje typ wyliczeniowy o nazwie D**ni**.

Zmienne tego typu będą mogły przybierać dokładnie siedem wartości, wyliczonych w nawiasach klamrowych w definicji tego typu
(stąd nazwa *wyliczenia*, typ *wyliczeniowy*).

Definicja typu wyliczeniowego składa się ze słowa kluczowego `enum`, nazwy wprowadzanego typu oraz ujętej w nawiasy klamrowe listy wyliczeniowej tj. nazw elementów wyliczenia (*stałych wyliczeniowych*).

Elementom listy wyliczeniowej można przypisywać wartości liczbowe (patrz niżej), a nazwę typu można opuścić (typ anonimowy).

enum – typ wyliczeniowy

```
enum Dni {pon, wto, sro, czw, pia, sob, nie};
```

Wewnętrznie zmienne typu **dni** będą reprezentowane za pomocą kolejnych liczb całkowitych poczynając od zera: tak więc symbol ***pon*** odpowiadać będzie liczbie 0, a np. symbol ***sob*** - liczbie 5.

Elementom wyliczenia można nadać odpowiadające im wartości liczbowe :

```
enum Dni {pon, wto=0, sro=0, czw=0, pia=0, sob, nie};
```

przy czym obowiązują następujące zasady:

- pierwszy element będzie odpowiadał wartości 0 jeśli nie nadano mu innej wartości poprzez jawne przypisanie w definicji wyliczenia. Tak więc w powyższym przykładzie symbol ***pon*** odpowiada wartości liczbowej 0.
- każdemu innemu elementowi zostanie przypisana wartość o jeden większa od wartości elementu poprzedniego, chyba że poprzez jawne przypisanie w definicji określono inną wartość. Tak więc w naszym przykładzie symbolom od ***wto*** do ***pia*** zostanie również przypisana wartość 0. Symbolowi ***sob*** nie została jawnie przypisana żadna wartość, a zatem niejawnie otrzyma on wartość 1 (jako następujący po ***pia***, którego wartość jest 0). Podobnie, ***nie*** odpowiadać będzie liczbie 2.

enum – typ wyliczeniowy

Przypisane elementom wyliczenia wartości nie tylko *nie* muszą być różne dla różnych elementów wyliczenia, ale nie muszą też być wartościami kolejnymi ani rosnącymi; mogą występować „dziury”:

```
enum Dni {pon, wto=0, sro=0, czw=0, pia=0, sob, nie=3};
```

Po takiej definicji ***sob*** odpowiada w dalszym ciągu wartości 1, ale niedzieli (zmiennej ***nie***) odpowiada teraz 3; wartości 2 nie odpowiada zaś teraz żaden element wyliczenia.

W nowym standardzie (C++11), możemy jawnie określić typ stałych wyliczeniowych (musi to być typ całkowitoliczbowy). Robimy to poprzez podanie nazwy typu po nazwie wyliczenia i dwukropku:

```
enum kolor : unsigned {pik, kier, karo, trefl};
```

Według nowego standardu, jeśli nie określimy typu stałych wyliczniowych, to będzie nim `int`.

enum – typ wyliczeniowy

Napisz funkcję zwracającą pełną nazwę dnia tygodnia w zależności od wartości zmiennej typu wyliczeniowego

```
enum Dni {pon, wto, sro, czw, pia, sob, nie};
string nazwaDnia(Dni dz)
{
    switch(dz)
    {
        case pon :  return "poniedzialek";
        case wto :  return "wtorek";
        case sro :  return "sroda";
        case czw :  return "czwartek";
        case pia :  return "piatek";
        case sob :  return "sobota";
        case nie :  return "niedziela";
    }
}
```

enum – typ wyliczeniowy

Napisać f-cje która dla danego numeru dnia (poniedziałek – 1,.., niedziela – 7) zwraca dzień tygodnia – wartość typu wyliczeniowego `Dni`

```
Dni zwrocDzien(int nr)
{
  switch(nr)
  {
   case 1: return pon;
   case 2: return wto;
   case 3: return sro;
   case 4: return czw;
   case 5: return pia;
   case 6: return sob;
   case 7: return nie;
  }
}
```

enum – typ wyliczeniowy

```
int main()
{   Dni d = wto;
    cout<<d<<endl;//1
    cout<<nazwaDnia(d)<<endl<<endl;//wtorek

     //petla wyswietlajaca wszystkie dni tygodnia
    for (Dni dz= pon; dz<=nie; dz=(Dni)(dz+1))
              cout<<nazwaDnia(dz)<<endl;

    //pobieramy numer dnia tygodnia
    int nr;
    cout<<"podaj numer dnia:";  cin>>nr;
 //na podstawie numeru wyznaczamy wartosc typu wyliczeniowego
    d = zwrocDzien(nr);
    cout<<nazwaDnia(d)<<endl<<endl; //wyswietlamy nazwe dnia d
    return 0;
}
```

Przykład 6 wersja z dokładnym określeniem rodzaju błędu

```
enum ResultType{OK, OPEN_ERROR, EMPTY_FILE, READ_ERROR};
//funkcja oblicza sumę liczb w pliku, dokładnie określany //jest rodzaj błędu


ResultType SumaPlikEnum(string nazwa, double &suma)
{   ifstream pwe(nazwa.c_str());
    //w C++11: ifstream pwe(nazwa);
    if(!pwe.is_open())
        return OPEN_ERROR;
    suma=0.0;
    double x;
    if (!(pwe>>suma)){
        ResultType r= pwe.eof()?EMPTY_FILE : READ_ERROR;
        pwe.close();
        return r;
                        //plik pusty lub błąd
    }
```

Przykład 6 wersja z dokładnym określeniem rodzaju błędu

```
    while(pwe>>x)
         suma += x;

    if(!pwe.eof()){  //jesli nie doszliśmy do końca pliku
      pwe.close();
      return READ_ERROR;
    }
    pwe.close();
    return OK;
}
```

Przykład 6 wersja z dokładnym określeniem rodzaju błędu

```
int main()
{
   //funkcja z wykorzystaniem enum
   string nplik;
   cout << "podaj nazwe pliku: ";
   cin >> nplik;
   double suma;
   switch(SumaPlikEnum(nplik,suma))
   {
     case OK: cout<<"suma elementow= "<<suma<<endl; break;
     case OPEN_ERROR: cout<<"blad otwarcia pliku "<<endl; break;
     case EMPTY_FILE: cout<<"plik pusty"<<endl;
     case READ_ERROR: cout<<"blad odczytu z pliku"<<endl;
   }
   return 0;
}
```

Przykład 7 - zapis struktury do pliku z formatowaniem

Dane są typy strukturalne:

```
struct TPrzesylka{
    string skad, dokad;
    float waga; //w kg
    bool priorytet; //wartosc prawda dla przesyłki
                    // priorytetowej
};

struct TListaPrzesylek{
    int ile_p; //liczba przesyłek, ile_p <= 100
    TPrzesylka tab_p[100]; //tablica zawiera dane o
                           // wszystkich przesylkach
};
```

Przykład 7 - zapis struktury do pliku z formatowaniem

Napisz funkcje o nagłówku

```
int zapiszPrzesylki(const TListaPrzesylek& listap, string nazwapl);
```

która zapisze do pliku o nazwie przekazanej w parametrze `nazwapl` pełną informację o przesyłkach priorytetowych ze struktury `listap`.

Wartością funkcji jest ilość zapisanych przesyłek.

Jeśli nie uda się otwarcie pliku o zadanej nazwie, wartością funkcji ma być –1.

Jeśli w strukturze brak jest jakichkolwiek przesyłek to wartością funkcji jest -1.

Zadbaj o formatowanie danych w pliku: numer przesyłki z wyrównaniem do prawej, pola `string` wyjustowane do lewej, `waga` z dokładnością do 2-ch miejsc po przecinku z wyrównaniem do prawej.

Przykład 7 – zapis struktury do pliku z formatowaniem

```
#include <iostream>
#include <fstream>
#include <string>
#include <iomanip>
using namespace std;

struct TPrzesylka{
   string skad, dokad;
   float waga;
   bool priorytet; //wartosc prawda dla przesyłki priorytetowej
};

struct TListaPrzesylek{
   int ile_p; //liczba przesyłek, ile_p <= 100
   TPrzesylka tab_p[100]; //tablica zawiera dane o przesylkach
};
```

Przykład 7– zapis struktury do pliku z formatowaniem

```
int zapiszPrzesylki(const TListaPrzesylek& lp, string nazwapl)
{
   //pusta struktura
    if (lp.ile_p==0) return -1;


    ofstream pwy(nazwapl.c_str());
    //jesli otwarcie pliku nie powiodlo sie
    if (!pwy.is_open())
        return -1;
    int ile=0; //licznik przesylek priorytetowych
    //ustawiamy precyzje na 2 miejsca po ,
    pwy<<fixed<<setprecision(2);
```

Przykład 7– zapis struktury do pliku z formatowaniem

```
    for (int i=0; i < lp.ile_p; i++) //przechodzimy po liscie
                                     // przesylek
    {
        if(lp.tab_p[i].priorytet) //jesli jest priorytetowa
        {
            ile++;//zwiekszamy licznik i zapisujemy do pliku
            pwy<<setw(3)<<ile<<"."<<left
               <<setw(20)<<lp.tab_p[i].skad
               <<" - "<<setw(20)<<lp.tab_p[i].dokad<<" - "
               <<right<<setw(6)<<lp.tab_p[i].waga<<"kg"<<endl;
        }
    }
    pwy.close();
    return ile;
}
```

Przykład 7–zapis struktury do pliku z formatowaniem

```
int main()
{
    TListaPrzesylek lista={4,{{"Warszawa", "Koluszki", 2.55, true},
                           {"Warszawa", "Gniezno", 8, false},
                           {"Krakow", "Szczecin",45.2, true},
                           {"Katowice", "Poznan", 123.4, true}}};
    int ilep = zapiszPrzesylki(lista,"priorytetowe.txt");
    if(ilep<0)
        cout<<"blad otwarcia pliku lub pusta struktura"<<endl;
    else
        cout<<ilep<<" przesylek zapisano do pliku"<<endl;
    return 0;
}
```

Zawartość pliku:

```
  1. Warszawa          - Koluszki          -   2.55 kg
  2. Krakow            - Szczecin          -  45.20 kg
  3. Katowice          - Poznan            - 123.40 kg
```

Tryby otwarcia pliku *

- Tryb otwarcia pliku określa sposób w jaki dany plik ma być używany: do odczytu, do zapisu, w celu dołączenia itd.
- Przy kojarzeniu strumienia z plikiem (konstruktor lub open()) możemy podać drugi argument określający tryb otwarcia pliku:

```
//konstruktor z argumentem określającym typ

ifstream fin("dane_we.txt", mode1);

//metoda open() z argumentem określającym typ

ofstream fout;

fout.open("wyniki.txt", mode2);
```

Tryby otwarcia pliku *

W klasie `ios_base` są zdefiniowane stałe trybu otwarcia (typu `bitmask`)

Stałe	Znaczenie
`ios_base::app`	(***app****end*) Dołącz na koniec pliku
`ios_base::ate`	(***at e****nd*) Po otwarciu pliku ustaw się na jego końcu
`ios_base::binary`	(***binary***) Plik binarny
`ios_base::in`	(***in****put*) Otwórz plik do odczytu
`ios_base::out`	(***out****put*) Otwórz plik do zapisu
`ios_base::trunc`	(***trunc****ate*) Zredukuj rozmiar pliku do zera jeśli istnieje

Tryby otwarcia pliku *

- W omawianych do tej pory przykładach używaliśmy tylko jednego argumentu:

```
ofstream we; we.open("plik.txt");
```

- W prototypach tych funkcji określono dla drugiego argumentu - trybu otwarcia wartości domyślne.
- W konstruktorze i metodzie `open()` klasy `ifstream` domyślną wartością drugiego argumentu jest stała `ios_base::in` (otwórz do odczytu), a w przypadku klasy `ofstream` domyślną wartością jest wyrażenie `ios_base::out | ios_base::trunc` (otwórz do zapisu i zredukuj rozmiar pliku do zera).
- Operator alternatywy bitowej (|) łączy dwie wartości bitowe w pojedynczą wartość, przy użyciu której można ustalić obydwa bity.
- W przypadku klasy `fstream` nie określono trybu domyślnego, a więc przy tworzeniu obiektu tej klasy trzeba podać go jawnie.
- Jeśli chcemy zachować zawartość pliku i dołączyć nowe dane na jego końcu, możemy użyć trybu `ios_base::app`:

```
ofstream fout("dolacz.txt", ios_base::out|ios_base::app);
```

Dołączanie pliku

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;
int main()
{   // dodawanie danych nowych osób do pliku
    ofstream fout("plik.txt", ios_base::out | ios_base::app);
    if (!fout.is_open())
    {
        cerr << "Nie mozna otworzyc pliku do zapisu.\n";
        return 1;
    }
    cout << "Podaj dane kolejnych osób
           (aby zakonczyc, wprowadz pusty wiersz):\n";
```

Dołączanie pliku

```
    string dane;
     while (getline(cin, dane) && dane.size() > 0)
     {
         fout << dane << endl;
     }
     fout.close();
     return 0;
}
```

Przykład 8- zapis macierzy do pliku z formatowaniem

Napisz i przetestuj funkcje logiczną o nagłówku

```
bool zapiszMacierz(string nazwapl, float mac[][NMAX], int w, int k );
```

która do pliku o nazwie przekazanej przez parametr `nazwapl` zapisze macierz `mac` o `w`-wierszach i `k`- kolumnach, gdzie `w,k <= NMAX.`

Jeśli operacje plikowe powiodą się zwraca `true`, a `false` w przeciwnym przypadku.

Zadbaj odpowiednie formatowanie wyników.

```
#include <iostream>
#include <fstream>
#include <string>
#include <iomanip>
using namespace std;

const int NMAX = 10;

//teraz przed funkcja main() tylko deklaracja funkcji,
//definicja po main()

//w prototypie- deklaracji funkcji – można opuścić nazwy parametrów
bool zapiszMacierz(string , float [][NMAX], int , int  );
```

Przykład 8- zapis macierzy do pliku z formatowaniem

```
int main()
{
    float t[][NMAX] = { {    1,  3,    5,        23, 16.42},
                        {4.567,  4,    6,  234.345,    98},
                        {  585, 34,    1,        67,  31.2},
                        {    1,  0,    1, 2345.967, 123.2}};
   string n;
   cout<<"podaj nazwe pliku do zapisu ";
   cin>>n;
   if( zapiszMacierz(n,t,4,5))
     cout<<"macierz zapisana do pliku"<<endl;
   else
     cout<<"blad otwarca pliku"<<endl;
   return 0;
}
```

Przykład 8- zapis macierzy do pliku z formatowaniem

```
//definicja funkcji – tu koniecznie nazwy parametrów
bool zapiszMacierz(string nazwapl, float mac[][NMAX], int w, int k )
{
    ofstream pwy(nazwapl.c_str());
    if (!pwy.is_open()) //jesli otwarcie pliku nie powiodlo sie
        return false;
    pwy<<fixed<<setprecision(3);
    for(int i=0; i<w; i++)//idziemy po wierszach od 0 do w-1
    {
        for (int j=0; j<k; j++)//wzdluz i-tego wiersza(po kolumnach)
        {
            pwy<<setw(9)<<mac[i][j];
        }
        //po wyswietleniu calego wiersza wstawiamy znak nowej linii
        pwy<<endl;
    }
    pwy.close();
    return true;
}
```