# Laboratorium Informatyki II
# Ćwiczenie 5

## 1 Metody obsługi strumienia

Obiekty `cin` i `cout` odpowiadają strumieniom wejściowemu i wyjściowemu realizując zadania wysyłania na ekran monitora i odczytywania znaków wprowadzonych z klawiatury. Podstawowa obsługa realizowana jest przez przesyłanie do lub z obiektów danych za pomocą operatorów `<<` i `>>`. Obiekty te dostarczają także szeregu metod wspierających obsługę obiektów wejściowego i wyjściowego. Wywołanie metody realizuje się poprzez podanie jej nazwy po nazwie obiektu, rozdzielone kropką, jak pokazano poniżej.

```
cout.put('A');
cin.get(x);
```

### 1.1 Strumień wejściowy

Obiekt `cin` dostarcza szeregu metod obsługi strumienia wejściowego. Podstawową z nich jest metoda `.get()` pozwalająca na pobranie znaku lub ciągu znaków z bufora wejściowego. W odróżnieniu od odczytu danych za pomocą operatora przekierowania strumienia `>>`, metoda `.get()` potrafi odczytać także znaki białe jak spacje, tabulacje czy znak przejścia do nowego wiersza. Dostępnych jest kilka implementacji metody w kodzie w zależności od typu zmiennej do jakiej zapisane zostaną odczytane dane, jak pokazano poniżej.

```
cin.get(char znak);                    //typ char
chat znak = cin.get();                 //typ char
cin.get(char* tab,int n);              //typ tablica char
cin.get(char* tab,int n ,char znak);   //typ tablica char
```

Dwie pierwsze pozwalają na odczytanie z bufora wejściowego pojedynczego znaku i przypisanie go do zmiennej typu `char`. Trzecia z wymienionych powyżej implementacji odczytuje z bufora wejściowego określoną liczbę znaków lub wszystkie znaki do momentu wykrycia znaku odpowiadającego naciśnięciu klawisza `ENTER`. Pobrane dane zapisane zostaną do tablicy typu `char` wraz z znakiem końca ciągu tekstowego `'\0'`. Ostatnia forma pozwala dodatkowo określić trzecie kryterium końca pobierania danych, dowolny znak podany w apostrofach lub odpowiadająca mu wartość z strony kodowej. Metoda pobiera z bufora maksimum `n-1` znaków, ze względu na to że ostatnim znakiem przypisanym do tablicy będzie `'\0'`. Obiekt `cin` posiada jeszcze inne metody odczytu danych z bufora wejściowego:

- `.getline(char*tab,int limit);` - odczytuje znaki z bufora wejściowego i przypisuje je do tablicy znakowej. Limitem jest liczba znaków lub znak `ENTER`. Pobrany ciąg zakończony będzie znakiem `'\0'`.

- `.getline(char*tab,char znak);` - odczytuje znaki z bufora wejściowego i przypisuje je do tablicy znakowej. Limitem jest określony znak lub znak `ENTER`. Pobrany ciąg zakończony będzie znakiem `'\0'`.

- `char znak = cin.peek()` - odczytuje znak z bufora wejściowego nie usuwając go

- `.read(char *tab, int limit)` - odczytuje `n` znaków z bufora wejściowego. Nie dodaje znaku `'\0'` na jego końcu.

W pewnych sytuacjach może dojść do wprowadzenia przez użytkownika niekompatybilnych danych. W takiej sytuacji może dojść za zablokowania bufora wejściowego. Konieczne stanie się opróżnienie go z aktualnie zapisanych w nim danych. Zrealizować można to za pomocą metody `.ignore()`. Metodę można wywołać na trzy sposoby, jak pokazano poniżej.

```
std::cin.ignore();                 //czyści bufor z jednego znaku
std::cin.ignore(int n);            //czyści bufor z (int)n znaków
std::cin.ignore(int n,char znak); //czyści bufor z (int)n znaków lub
                                    do momentu natrafienia na określony znak
```

Metodę `.ignore()` stosuje się zazwyczaj gdy wystąpił błąd strumienia.

## 1.2 Strumień wyjściowy

Metody dla strumienia wyjściowego reprezentowanego przez obiekt `cout` pozwalają nie tylko na wyświetlanie znaków, ale także na określenie formy wyświetlania danych przez obiekt.

Obiekt `cout` posiada dwie metody dedykowane do wyświetlania znaków

- `.put(char znak)` - wyświetlenie na ekranie pojedynczego znaku przekazanego jako znak lub wartość kodowa znaku (tablica ASCII)

- `.write(char* tablica,int liczba_znaków)` - wyświetlenie określonej liczby znaków z tablicy znakowej

Metody można łączyć w kaskadę, jak pokazano w przykładzie pokazanym poniżej.

```
#include <iostream>
using namespace std;
int main()
{
    char tekst1[] = "Testowy ciąg znakowy.";
    cout.put(124).put('\t').put('@').put('\n');
    cout.write(tekst1,sizeof(tekst1)).put('\n');
}
```

Pozostałe metody można podzielić na dwie zasadnicze grupy. Metody ustawiające formę wyświetlanych danych, oraz sterujące sposobem wyświetlania danych.

Dane liczbowe standardowo wyświetlane są w systemie dziesiętnym. Obiekt `cout` umożliwia zmianę na system ósemkowy i szesnastkowy dla liczb całkowitych. Liczby rzeczywiste mogą być wyświetlane tylko w systemie dziesiętnym. Sposób przełączania trybu pokazano poniżej.

```
#include <iostream>
using namespace std;
int main()
{
    oct(cout); cout << "oct-->" << 4345 << endl;
    dec(cout); cout << "dec-->" << 4345 << endl;
    hex(cout); cout << "hex-->" << 4345 << endl;
    cout << oct; cout << "oct-->" << 7777 << endl;
    cout << dec; cout << "dec-->" << 7777 << endl;
    cout << hex; cout << "hex-->" << 7777 << endl;
}
```

Podstawowymi elementami jakie są ustawiane to liczba znaków `.width(int liczba)` i precyzja wyświetlania liczb rzeczywistych `.precision(int liczba)`. Metody te ustawiają sposób wyświetlania pierwszej wartości przekazanej do obiektu `cout`. Precyzyjne sterowanie sposobem wyświetlania ustawia się za pomocą metody `.setf()` i dedykowanych manipulatorów. Przykładowe zastosowanie metod formatujących pokazano poniżej.

```
#include <iostream>
#include <cmath>
using namespace std;
int main()
{
    cout.setf(ios_base::fixed, ios_base::floatfield);
    for(int i=0;i<=20;i++){
        cout.width(2);
        cout << i << ": ";
        cout.width(22);
        cout.precision(i);
        cout << 1.0/pow(10,i) << endl;
    }
}
```

# 2 Stan strumienia IO

Stan strumienia **IO** jest opisywany przez trzy bity:

- `eofbit` – ustawia wartość kiedy `cin` napotka koniec pliku
- `badbit` – ustawia wartość gdy strumień z jakiegoś powodu został uszkodzony (np.: *błąd odczytu danych z pliku, niekompatybilny typ danych*)
- `failbit` - ustawia wartość kiedy metody `cin` lub `cout` wykonane zostaną z błędem (błędny typ danych, brak prawa dostępu do pliku itp.)

Jeżeli pojawi się błąd i bit błędu zostanie ustawiony, nie będzie możliwe poprawne niekorzystanie ze strumienia **IO** do czasu przywrócenia stanu poprawnego - wyczyszczeniu bitów błędów. Odczytanie stanów strumieni jest możliwe za pomocą dedykowanych metod dla obiektów IO (`cout` i `cin`):

- `.good()` – zwraca true jeżeli strumień nadaje się do użytku
- `.eof()` – wraca true jeżeli koniec pliku (ustawiony `eofbit`)

- `.bad()` – zwraca true jeżeli ustawiony jest `badbit`
- `.fail()` – zwraca true jeżeli ustawiony jest `badbit` lub `failbit`
- `.rdstate()` zwraca stan strumienia:

  0 - stan OK

  1 - `eofbit`

  2 - `failbit`

  3 - `badbit`

- `.clear()` – czyszczenie bitów stanu

W przypadku bufora wejściowego, poza przywrócenie bitów błędów metodą `.clear()` konieczne może być opróżnienie bufora wejściowego z niekompatybilnych danych, jak pokazano poniżej.

```
cin.clear();
while (!isspace(cin.get()));

cin.clear();
while (cin.get() != '\n');
```

Poniżej zapisano przykład odczytu danych z bufora wejściowego z kontrolą i naprawą stanu strumienia wejściowego.

```
#include <iostream>
using namespace std;
void test();
void czysc();
int suma();
int main(){
    cout.setf(ios_base::boolalpha);
    cout << "Dane wejściowe: ";
    int sum = suma();
    cout << "Suma=" << sum << endl;
    test();
    cout << "Nowe dane wejściowe: ";
    sum = suma();
    cout << "Suma=" << sum << endl;
    test();
}
void test()
{
    cout << "Stan: " << cin.rdstate() << endl;
    cout << "test .good() : " << cin.good() << endl;
    cout << "test .bad()  : " << cin.bad() << endl;
    cout << "test .fail() : " << cin.fail() << endl;
    if(!cin.good())
        czysc();
}
int suma()
```

```
{
    int sum = 0, x=1;
    while(cin >> x && x!=0)
        sum+=x;
    return sum;
}
void czysc()
{
    cout << "Czyszczenie" << endl;
    cin.clear();
    while(cin.get()!='\n');
    cout << "Kontrola:" << endl;
    test();
}
```

# 3 Wyjątki

Każda tworzona aplikacja narażona jest na błędy:

- niekompatybilne dane wejściowe
- błędy przy przetwarzaniu danych
- inne nieprzewidziane sytuacje

Reakcję programu na tego typu sytuacje należy starać się zaprogramować w kodzie. Część sytuacji możemy oprogramować z wykorzystanie konstrukcji warunkowej. Jednakże zazwyczaj takie rozwiązanie jest mało efektywne. W języku C++ problem ten zdecydowanie wygodniej jest rozwiązać w oparciu o mechanizm wyjątków. Składa się on z dwóch elementów:

- testowania i przechwycenia błędu,
- wysłania komunikatu błędu.

Większość klas standardowych (np.: `iostream`) posiada wbudowany mechanizm wysyłania komunikatów o błędach. Za pomocą mechanizmu przechwycenia można go odczytać, zinterpretować i zaprogramować odpowiednie działanie.

```
try{
   testowany kod;
}
catch(const typ nazwa){
   reakcja na przechwycony błąd;
}
catch(...){
   reakcja na przechwycenie dowolnego,
   innego od wcześniej przechwyconych błędów;
}
```

Testowany w bloku `try{}` kod w przypadku wystąpienia błędu o zaprogramowanej detekcji może wysłać komunikat na pomocą instrukcji `throw komunikat`. Klasy i funkcje

standardowe (`std`) mają wbudowane mechanizmy detekcji z zwracania komunikatów błędach które może przechwycić funkcja `catch(){}`. Poniższy przykład obrazuje zastosowanie mechanizmu wyjątków w kontroli i reakcji na błędy. Poniżej pokazano przykład implementacji wyjątków w kodzie programu.

```
#include <iostream>
#include <cmath>
using namespace std;
int main()
{
    int x, n=0,Sg=1;
    cout << "Podaj ciąg liczb: ";
    try{
        while(true){
            if(!(cin >> x)) throw 101;
            if(!x) break;
            if(x<0) throw 201;
            Sg*=x;
            n++;
        }
        if(n>0){
            cout << "Średnia geometryczna Sg = " << pow(Sg,1.0/n) << endl;
            cout << "Liczba wrowadzonych liczb n = " << n << endl;
        }else
            cout << "Zbiór pusty." << endl;
    }
    catch(const int x){
        cout << "Błąd: ";
        switch(x){
        case 101:
            cout << x << ", dane niekompatybilne.";
            break;
        case 201:
            cout << x << ", wprowadzono liczbę ujemną." << endl;
            break;
        default:
            cout << " nieznany!" << endl;
        }
    }
}
```

Program wykrywa wprowadzenie liczby ujemnej lub wartości znakowej (niekompatybilnej). Wysyłany jest za pomocą funkcji `throw` komunikatu o wybranej wartości, przerywając jednocześnie kod realizowany w bloku `try{}`. Jest on następnie przechwytywany przez instrukcje `catch(){}` i może być w tym bloku przetwarzany.

# Zadania

W oparciu o informacje wykładowe i treść niniejszej instrukcji napisać programy. Zastosować dynamiczną alokację pamięci za pośrednictwem operatorów `new` i `delete`. Napisać program "Spis książek.

- Program gromadzi dane o książkach w domowej biblioteczce.
  - tytuł książki
  - autor (autorzy)
  - tematyka (kryminał, podręcznik, poezja itd)
  - liczba stron
- Zaimplementować kontrolę błędów przy wprowadzaniu danych (wyjątki)
- Program posiada interfejs pozwalający na dodanie nowej pozycji lub wyświetlenie listy wprowadzonych pozycji
- Spis pozycji wyświetlany jest w postaci sformatowanej tabeli