Laboratorium Informatyki II
Ćwiczenie 2

1 Metody obsługi strumienia wejściowego

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.

1.1 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();
}
```

2 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.

3 Wskaźnik i referencja

Jest to specyficzna zmienna określonego typu która umożliwia przechowanie adresu do komórki pamięci w której jest przechowywana wartość danego typu. Wskaźnik deklaruje się podając typ jaki będzie wskazywał oraz jego nazwę poprzedzoną znakiem `*`. Do przypisania adresu często wykorzystuje się operator referencji `&`, który "wyciąga" adres w pamięci pod którym dana zmienna przechowuje swoją wartość. W celu "wyciągnięcia" wartości wskazywanej przez wskaźnik należy jego nazwę poprzedzić operatorem wyłuskania `*`.

```
#include <iostream>
using namespace std;
int main()
{
    int x = 25;
    int *wsk = &x;
    cout << "x=" << x << "\t*wsk=" << *wsk << "\twsk=" << wsk << endl;
    *wsk = 34;
    cout << "x=" << x << "\t*wsk=" << *wsk << "\twsk=" << wsk << endl;
}
```

4 Klasa

Klasa jest typem deklaratywnym o budowie zbliżonej do struktury, która poza danymi dostarcza także narzędzi (metod) do przetwarzania tych danych. Program obiektowy jest zbiorem obiektów wymieniających i przetwarzających informacje pomiędzy sobą.

W klasie definiuje się dane zwane atrybutami lub polami za pomocą których opisywany jest jego stan, oraz funkcji, procedur zwanych metodami które dostarczają narzędzi interfejsu do przetwarzania atrybutów obiektu.

Dostęp do poszczególnych atrybutów i metod obiektu jest precyzyjnie definiowany. W ramach projektowanej klasy definiuje się które z jej elementów są ogólnie dostępne, a do

których dostęp jest ograniczony lub zabroniony. W klasie można określić trzy poziomu dostępu do jej elementów:

- `public` - publiczny, można odwołać się bezpośrednio do tych elementów obiektu z poziomu programu. W obszarze tym definiuje się interfejs klasy

- `private` - prywatny, dostęp do elementów z tej grupy możliwy tylko dla metod klasy. W obszarze tym deklaruje się atrybuty klasy (zazwyczaj) oraz metody wewnętrzne klasy.

- `protected` - chroniony, dostęp tylko dla metod klasy i klas dziedziczących na osobno określonych zasadach. W obszarze tym deklaruje się wszystkie elementy (atrybuty i metody) do których dostęp mają mieć klasy pochodne.

Poziom `private` jest traktowany jako domyślny i nie musi być formalnie deklarowany, ale ze względu na czytelność kodu zaleca się umieszczanie go w kodzie. Deklaracja trybu dostępu polega na umieszczeniu jego nazwy w kodzie klasy zakończonej dwukropkiem Wszystko co zostanie umieszczone po zgłoszeniu trybu, aż do zgłoszenia innego trybu lub końca deklaracji klasy, będzie z nim związane. Zazwyczaj projektując klasę określa się elementy publiczne i prywatne, natomiast chronione tylko wtedy gdy dodatkowo projektujemy klasy dziedziczące po bazowej. Poniżej zapisano przykład programu obiektowego.

```
#include <iostream>
using namespace std;
class dane{
private:
   int a;
   float b;
public:
    void set(char *nazwa)
    {
        cout << "Podaj " << nazwa << ".a= "; cin >> a;
        cout << "Podaj " << nazwa << ".b= "; cin >> b;
    }
    int get_a(){
        return a;
    }
    float get_b(){
        return b;
    }
};
int main()
{
    dane X,Y;
    X.set("X");
    cout << "X.a= " << X.get_a() << " : X.b= " << X.get_b() << endl;
    cout << "Y.a= " << Y.get_a() << " : Y.b= " << Y.get_b() << endl;
    Y=X;
    cout << "Y.a= " << Y.get_a() << " : Y.b= " << Y.get_b() << endl;
}
```

5 Wskaźnik *this*

W pewnych sytuacja zachodzi konieczność aby wewnątrz klasy odwołać się do atrybutów klasy które mają taką samą nazwę jak atrybuty metody. Wskaźnik *this* pozwala na precyzyjne odniesienie się do obiektu w kodzie obiektu. Poniżej pokazano kod programu obiektowego wykorzystującego wskaźnik *this*.

```
#include <iostream>
using namespace std;
class dane{
private:
   int a;
   float b;
public:
    void set(int a, float b)
    {
        this->a = a;
        this->b = b;
    }
    void draw()
    {
        cout << "a = " << a << " : b = " << b << endl;
    }
};
int main()
{
    dane X;
    X.set(5,13.23);
    X.draw();
}
```

Zadania

W oparciu o informacje wykładowe i treść niniejszej instrukcji napisać program gromadzący informacje o punktach nawigacyjnych wycieczki górskiej. Program powinien spełniać spełniać poniższe warunki:

a) program obiektowy, oparty na zaprojektowanej klasie,

b) program gromadzi dane o nazwie punktu, współrzędnych punktu i jego wysokości npm,

c) można wprowadzić dowolną, określoną przez użytkownika liczbę punktów,

d) w oprogramowaniu operacji wejścia\wyjścia oprzeć o metody obiektów `cin` i `cout`,

e) po zebraniu danych program wyświetla dane o przebiegu poszczególnych etapów wycieczki oraz dane podsumowujące.

Kod klasy zdefiniować w osobnych plikach (.cpp i *.hpp).*