Szablony klas

Szablony klas umożliwiają definiowanie ogólnej klasy czyli mogącej przetwarzać dane różnego typu (tzw. parametryzacja typów).

Użyteczność takich klas jest szczególnie widoczna przy implementacji tzw. kontenerów. Intensywnie używa ich biblioteka STL

Szablony klas a dziedziczenie
– szablony klas to inne podejście do idei tzw. powtórnego użycia kodu (ang. code reuse). Zamiast dziedziczenia (powtórne użycie kodu obiektowego), wzorce powtórnie używają kodu źródłowego.

Przykład
Klasa zwykła (tylko dla „int”):

```
class Przechowaj
{
      int a;
      public:
      Przechowaj(int A):a(A){}
      void Print() { cout<<a<<endl; }
};

int main()
{
      Przechowaj p(2); //możemy przechowywać tylko zmienne typu „int"
      p.Print();
      return 0;
}
```

Klasy szablonowe – zmieniamy konkretny typ „int” na ogólny typ „T”:

Klasa szablonowa (zadziała dla dowolnego typu):

```
template<class T>
class Przechowaj
{
      T a;
      public:
      Przechowaj( T A):a(A){}
      void Print() { cout<<a<<endl; }
};

int main()
{
      Przechowaj<int> p(2); Przechowaj<char> p2('a');
      p.Print(); p2.Print();
      return 0;
}
```

Deklarujemy, że mamy do czynienia z szablonem

Ogólny typ T (nazwa T - dowolna)

Przy deklaracji musimy jasno powiedzieć, co będziemy przechowywać, np. int.

Klasa szablonowa (jeśli ciało funkcji na zewnątrz):

```
template<class T>
class Przechowaj
{
        T a;
        public:
        Przechowaj( T A);
        void Print();
};

template<class T>
Przechowaj<T>::Przechowaj( T A):a(A){}

template<class T>
void Przechowaj<T>::Print() { cout<<a<<endl; }
```

Przy każdej funkcji trzeba dodać formułkę:
template<class T>

Zmiana operatora zakresu – musimy dodać <T>:
NazwaKlasy<T>

Uwaga, cały kod piszemy w pliku h!