

Шаблони класів

- Шаблон класу – схема побудови інших класів на основі цього шаблону.
- Шаблони дають змогу створити один опис класу, який можна використовувати для роботи з різними типами даних(vector, list...).
- Шаблони значно скорочують код програми, тому що достатньо написати один шаблонний клас, який зможе працювати з різними типами даних.

Синтаксис оголошення шаблонного класу

- Оголошення шаблону починається зі слова template, яке вказує компілятору на те, що далі йде оголошення шаблону.
- Після `template` в `< >` вказуються шаблонні параметри `<class T1, class T2>`
- В даному випадку ключове слово class не означає те, що T1 обов'язково має бути класом, T1 може бути простим типом даних (`int`, `char...`)
- Також замість слова клас можна використовувати слово typename
- `Template <class T1, typename T2> class A {...}`
- Після цього у описі класу можна використовувати T1, T2 так само як і звичайні типи даних.

Приклад шаблонного стеку

```
template <class Type>
class Stack {
    private:
        enum {MAX = 10}
        Type items[MAX];
        int top;
    public:
        Stack();
        bool IsEmpty();
        bool IsFull();
        bool Push(const Type& item); //передаємо дані типу T для занесення в стек
        bool Pop(Type& item);
};
```

```
....
//використання шаблонного класу
Stack<int> stack_int;
Stack<string> stack_string;
```

Реалізація методів шаблонного класу

- Методи шаблонного класу стають шаблонними методами.
- Можуть бути як вбудованими (inline) так і реалізовані за межами класу.
- Синтаксис реалізації шаблонного методу:
 - *Template <class T ...> className<class T...>::methodName(parameters).*
- Якщо метод реалізовується в класі, то синтаксис реалізації є таким самим як у методів звичайного класу.

Приклад реалізації

```
template <class Type>
Stack<Type>::IsEmpty() {
    return top == 0;
}
```

```
template <class Type>
bool Stack<type>::IsFull() {
    return top == MAX;
}
```

```
template <class Type>
bool Stack<Type>::push(const Type& item) {
    if (top < MAX) {
        items[top++] = item;
        return true;
    }
    else {
        return false;
    }
}
```

```
template <class Type>
bool Stack<Type>::pop(Type& item) {
    if (top > 0) {
        item = items[--top];
        return true;
    }
    else {
        return false;
    }
}
```

Параметри за замовчуванням в шаблонах

- Шаблонні класи можуть приймати параметри за замовчуванням (функції з параметрами за замовчуванням).
- *Template<class T1, class T2 = int> class A {...}*
- Якщо при створенні об'єкта класу А, другий параметр не буде вказаний, то компілятор буде підставляти замість T2 int.
- *A<double, double> a1 //T1 = double, T2 = double*
- *A<double>a2//T1 = double, T2 = int.*

Спеціалізація шаблонів

- Явна спеціалізація – це визначення конкретного типу, який має використовуватись замість загального шаблону.
- Явна спеціалізація шаблону використовується тоді, коли шаблонний клас має вести себе по-іншому для цього типу даних.
- Синтаксис оголошення спеціалізованого шаблону:

Template<> class Ім'яКласу <Ім'я спеціалізованого типу>

Template <class T> class A {...} //загальний клас

Template<> class A<int> {...} //спеціалізований клас.

- Часткова спеціалізація шаблонів – використовується тоді, коли у шаблонному класі є декілька типів-параметрів (T1, T2) і клас має працювати по-іншому, якщо один з параметрів є деяким типом.

- Приклад

template <class T1, class T2> class A {...} //загальний шаблон

template <class T1> class A<T1, int> {...} //часткова спеціалізація

- Якщо у компілятора є вибір, то він приймає найбільш спеціалізований шаблон. Тобто якщо ви створюєте об'єкт класу з параметрами <double, double> то клас буде генеруватись на основі загального шаблону, але якщо у вас другим параметром буде int, то буде використовуватись частково спеціалізований шаблон.

Рекурсивне використання шаблонів

- Шаблон класу A, може виступати параметром для того ж класу A.
- Приклад:

```
template <class T, int n> class Array//шаблон масиву розміром n.
```

```
Array <int, 10> a; // створення об'єкта типу Array<int, 10>.  
//Масив цілих чисел розміром 10.
```

```
Array<Array<int, 5>, 10> b; //масив з 10 елементів, кожен з яких //є масивом з 5 елементів.
```

Шаблони як члени класів

- Шаблони можуть бути членами структури, класу чи іншого шаблону. Ця властивість шаблонів часто використовується в STL.

- Приклад

- `template <class T>`

```
class beta {
```

- `private:`

- `template <class V> // область видимості класу hold – клас beta.`

- `class hold {`

- `private:`

- `V val;`

- `public:`

- `hold(V v = 0) : val(v) {}`

- `void show() const {cout << val << endl;}`

- `V Value() const {return val;}`

- `};`

- `hold<T>q; //об'єкти класу hold в класі beta.`

- `hod<int> n;`

- `//...`

- `}`

Шаблони як параметри

- Шаблони можуть мати параметри, які також є шаблонами. Це також використовується для реалізації бібліотеки STL.
- Приклад

```
template <template <class T> class Thing>
```

```
Class A {...}
```

- Тут `<template <class T> class Thing` – параметр шаблону, при чому `template <class T> class` - тип, `Thing` – параметр.
- Для створення об'єкту класу `A`, йому потрібно передати параметр, який відповідає типу `class T> class` .

Шаблонні класи і друзі

- Шаблонні класи можуть мати друзів як і звичайні класи.
- Друзі можуть бути шаблонними або простими функціями, шаблонними або простими класами.
- Дружні функції і класи можуть належати до одної з трьох груп:
 - Нешаблонні друзі
 - Зв'язні шаблонні друзі – тип друга визначається типом класу при створенні екземпляра.
 - Незв'язні шаблонні друзі – всі спеціалізації друга є друзями для всіх спеціалізацій шаблонного класу.

Нешаблонні дружні функції

```
template <class T>
```

```
class HasFriend {
```

```
    public:
```

```
    // дружня функція для всіх екземплярів класу HasFriend
```

```
    friend void counts()
```

```
    friend void report(HasFriend<T>&); зв'язний друг
```

```
    //....
```

```
    //незв'язна дружня шаблонна функція, є дружньою для всіх екземплярів.
```

```
    template <typename C, typename D> friend void show(C &, D &)
```

```
}
```

HasFriend<int> //функція counts() є дружньою для цього класу

HasFriend<string> //і для цього теж

HasFriend<double> //ф-я report(HasFriend<double>&) дружня для цього класу

HasFriend<int> //ф-я report(HasFriend<int>&) дружня для цього класу

report(..) – не є шаблонною функцією, вона лише приймає шаблонний параметр.

Це означає, що потрібно створювати спеціалізації для параметра

```
void report(HasFriend<short>&) {..}
```

```
void report(HasFriend<string>&) {...}
```

Наслідування і шаблони

- Шаблонні класи підтримують наслідування так само як і звичайні класи.
- Все що справедливо для звичайних класів працює і з шаблонними.
- Повноцінна підтримка шаблонами механізму наслідування дає можливість створювати навіть абстрактні шаблонні класи.
- Шаблонна функція шаблонного класу не може бути віртуальною, але звичайна шаблонна функція шаблонного класу може бути віртуальною.