МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
"ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ"
ДГТУ

Кафедра "Программное обеспечение вычислительной техники и автоматизированных систем"

ШАБЛОНЫ В C++

Методические указания

Ростов-на-Дону 20 г.

УДК 004.438

Составитель: Б.В. Габрельян

Шаблоны в C++: метод. указания. - Ростов н/Д: Издательский центр ДГТУ, 20 . - 12 с.

Дано описание шаблонов функций и классов в C++. Приведены примеры объявления и использования собственных шаблонов функций и классов. Представлены задания по выполнению работы. Дан список контрольных вопросов. Приведен список рекомендуемой литературы.

Предназначены для студентов направлений 09.03.04 «Программная инженерия» и 02.03.03 «Математическое обеспечение и администрирование информационных систем».

Печатается по решению методической комиссии факультета «Информатика и вычислительная техника»

Научный редактор д-р техн. наук, проф. Р.А. Нейдорф

Рецензент канд. техн. наук, доц. В.Н. Землянухин

Цель работы: изучение шаблонов функций и классов в C++

I. Шаблоны функций

Если один и тот же алгоритм надо применять к данным разных типов, можно создать несколько функций с аргументами разных типов (перегрузить функции). C++ позволяет переложить работу по созданию таких перегруженных функций на компилятор. Для этого необходимо определить шаблон функций, в котором реальные типы данных заменены параметрами. Тогда при обращении к функции компилятор сможет сгенерировать подходящий код по шаблону, заменяя параметры конкретными типами фактических аргументов функции. Шаблоны создаются с использованием зарезервированного в C++ слова template. Параметры шаблона – это допустимые идентификаторы. При их объявлении используется зарезервированное слово typename (или class). Например,

```
template<typename T>   // шаблон с одним параметром
T Max( T x, T y ) {        /* имя функции Max, два аргумента одинакового типа, возвращаемое значение такого же типа */
        return x > y ? x : y;
}

int main() {
        int a = 5, b = 10, c;
        c = Max( a, b );              // создается функция Max(int,int)
        double d1 = 1.73, d2 = -1.5, d3;
        d3 = Max( d1, d2 );        // создается функция Max(double, double)
        return 0;
}
```

Такая форма вызова функции возможна, если все параметры шаблона, хотя бы единожды входят в список аргументов функции. Если это не так нужно явно искажать имя функции, добавляя к нему имя конкретного типа. Например,

```
template<typename T>
T fun() {
	return T();    // конструкция нулевой инициализации
}

int main() {
	int x;
	x = fun<int>();
	return 0;
}
```

Если есть шаблон, то по нему компилятор может создать функцию для любого допустимого типа, если в алгоритме шаблона не используются операции, запрещенные для этого типа. Т.е. не всегда один и тот же алгоритм подходит для всех допустимых типов, более того, возможны ситуации, когда синтаксически шаблон использовать можно, но смысл выполняемых действий отличается от ожидаемого. Например, если применить заданный выше шаблон Max для символьных Си-строк:

```
int main() {
	const char *str1 = "Hello", *str2 = "World", *s3;
	s3 = Max( str1, str2 );
	return 0;
}
```

то синтаксической ошибки не будет, но сравниваться будут адреса начальных символов строк. Чтобы исправить ситуацию нам нужно иметь средство для реализации особых случаев применения алгоритма. Это можно сделать с помощью специализированных шаблонов и специализированных функций. Например,

```
template<const char *>   // специализированный шаблон для типа char *
const char *Max(const char *s1, const char *s2) {
```

```
    if( strcmp( s1, s2 ) >= 0 ) return s1;
    return s2;
}
```

или, что эквивалентно, так:

```
template<>          // специализированный шаблон для типа char *
const char *Max(const char *s1, const char *s2) {
    …
}
```

Специализированная функция будет иметь следующий вид:

```
const char *Max(const char *s1, const char *s2) {
    if( strcmp( s1, s2 ) >= 0 ) return s1;
    return s2;
}
```

Специализированная функция имеет приоритет перед шаблоном, т.е. если есть такая функция, то используется она, а не шаблон.

II. Указатели на функции, методы классов и объекты-функции

Параметр шаблона может замещаться любым типом, поддерживающим нужную функциональность. Например, для указателей на функции, на статические и нестатические методы классов и для объектов классов, в которых переопределена операция вызова функции (их называют объектами-функциями), разрешена операция вызова функции.

Указатель на функцию

```
void (*fPtr)();
```

может содержать адрес любой функции соответствующего типа, в данном случае, не имеющей аргументов и не возвращающей значения.

```
void fun();
fPtr = &fun; // или fPtr = fun;
```

теперь можно вызвать функцию не напрямую, по имени, а косвенно, через указатель

```
fPtr();
```

применив к указателю на функцию операцию вызова функции - ().

Указатель на нестатический метод класса

```
class A {
public:
    void m_fun() {
        cout << “member function” << endl;
    }
};
```

void (A::*mfPtr)();

можно связать с любым методом класса A (и только A), не имеющим явных аргументов и не возвращающим значение.

```
mfPtr = &A::m_fun;
```

и вызвать метод через указатель с помощью операции .* (для объекта класса) или операции ->* (для указателя на объект класса)

```
A obj;
A *pobj = &obj;
(obj.*mfPtr)();
(pobj->*mfPtr)();
```

Объект-функция

```
class FunObj {
public:
    void operator()() {
        cout << “operator()” << endl;
    }
};
```

теперь, если есть объект класса

```
FunObj fobj;
```

можно применить к нему операцию вызова функции, вызывая тем самым метод operator()

```
fobj();
```

Можно создать шаблон, параметром которого будет то, что ведет себя как функция (т.е. поддерживает операцию **()**).

```
template<typename T>
void test(T fun) {
    fun();
}
```

и использовать его с функцией, статическим методом или объектом-функцией

```
int main() {
    test( fun ); // см. определение функции fun выше
    test( fObj ); // объект класса FunObj объявленный выше
}
```

Чтобы вызвать нестатический метод класса нужен объект или указатель на объект этого класса, поэтому шаблон может быть таким

```
template<typename ObjType, typename Fun>
void test(ObjType *obj, Fun fun) {
    (obj->*fun)();
}
```

класс A объявлен выше

```
int main() {
    A aObj;
    test( &aObj, &A::m_fun );
}
```

III. Шаблоны классов

Шаблоны классов дают возможность автоматической (т.е. усилиями компилятора) генерации набора классов, отличающихся только типами,

используемыми при их определении. Например, контейнеры, такие как списки, массивы и т.п. обеспечивают одинаковую функциональность независимо от типа элементов, помещаемых в контейнер. Тогда тип элемента можно представить в шаблоне параметром и, при создании объектов класса для контейнеров конкретного типа, нужно будет указать этот тип в имени класса, построенного по шаблону (шаблонного класса) и компилятор сгенерирует нужный класс. Например,

```
template<typename T>
class Stack {
        T *data;
        int max_size;
        int top;
  public:
        Stack(int max_s = 10) : max_size(max_s), top(-1) {
                data = new T[max_size];
        }
        ~Stack() {
                delete[] data;
        }
        void push(T elem) {
                if( top < max_size-1 ) data[++top] = elem;
        }
        int pop() throw(int) {
                if( top == -1 ) throw -1;
                return data[top--];
        }
        int top() throw(int) {
                if( top == -1 ) throw -1;
                return data[top];
        }
```

```
        bool isEmpty() {
            return top == -1;
        }
};
int main() {
        Stack<int> st1(20); // стек для не более чем 20 целых
        st1.push(12);
        st1.push(12);
        while( !st1.isEmpty() ) cout<<st1.pop()<<endl;
        Stack<const char *> st2; // стек для не более чем 10 (по умолчанию) строк
        st1.push("Hello");
        st1.push("World");
        while( !st2.isEmpty() ) cout<<st2.pop()<<endl;
        return 0;
}
```

Параметры шаблона, как типы, так и нетипы, могут иметь значения по умолчанию. Для типов это имя конкретного типа, для нетипов – константа нужного типа. Например,

```
template<typename T = int, int max_size = 10>
class Stack {
        T *data;
        int top;
  public:
        Stack() : top(-1) {
            data = new T[max_size];
        }
        ~Stack() {
            delete[] data;
        }
```

```
        void push(T elem) {
                if( top < max_size-1 ) data[++top] = elem;
        }
        int pop() throw(int) {
                if( top == -1 ) throw -1;
                return data[top--];
        }
        int top() throw(int) {
                if( top == -1 ) throw -1;
                return data[top];
        }
        bool isEmpty() {
                return top == -1;
        }
};
int main() {
        Stack< > st1; // стек для не более чем 10 целых (по умолчанию)
        Stack<char *> st2; // стек для не более чем 10 (по умолчанию) строк
        Stack<double,50> st3; // стек для не более чем 50 double
        …
        return 0;
}
```

IV. Свойства типов

Так как параметр шаблона может замещаться номинально любым типом, а конкретные алгоритмы или типы могут быть ориентированы только на некоторые типы, стандартная библиотека C++ предлагает шаблоны для определения и преобразования свойств типов, замещающих параметры шаблона. Эти средства описаны в файле заголовков type_traits (и во вложенных файлах). Например, свойство is_integral<Type> позволяет узнать, является ли Type (точнее тот тип, который замещает параметр шаблона) одним из типов bool, wchar_t, char, знаковым или без знаковым char, short, int , long, long long, а также

их версиями с квалификатором const или volatile. Свойства типов, такие как is_integral организованы как шаблоны классов (структур), содержащих статическое поле value типа bool.

```
#include <type_traits>
    int main() {
    if( is_integral<int>::value == true ) cout << "1. is integral" << endl;
    if( is_integral<int *>::value == true ) cout << "2. is integral" << endl;
    if( is_integral<const int>::value == true ) cout << "3. is integral" << endl;
    if( is_integral<volatile char>::value == true ) cout << "4. is integral" << endl;
    }
```

Выводит

1. is integral

3. is integral

4. is integral

Существуют такие свойства как is_arithmetic<T>, is_pointer<T>, is_array<T>, is_void<T>, is_function<T> и т.д. см. файл type_traints в каталоге include компилятора.

Можно сравнивать типы, например, is_same<T1,T2> позволяет узнать совпадают ли типы T1 и T2.

```
    template<typename T>
    void test(T arg) {
        if( is_same<T,int>::value == true ) // или if(is_same<T,int>::value)
            cout << arg << " is int" << endl;
        else
            cout << arg << " isn’t int" << endl;
    }
    int main() {
        test(10);
        test("abc");
    }
```

Выводит

10 is int

abc isn’t int

Можно получать новые типы применяя преобразования к заданным, например, remove_pointer<T>, add_pointer<T>, remove_reference<T>, add_lvalue_reference<T> и т.д. Здесь в шаблонах классов (структур) объявлено статическое поле type, задающее новый тип.

Например,

remove_pointer<int *>::type x; // int x;

V. ЗАДАНИЯ

1. Какие из следующих определений шаблонов неправильные? Почему?

```
template <class T, class T> T f1(T x) {
        return x;
}
template <class T1, T2> void f2(T1 x, T2 y) {
        cout << "f2: x = " << x << "\ty = " << y << endl;
}
template <class T>  T  f3(int x) {
        return T();
}

inline template <class T> T f4(T x, T y) {
        cout << "f4: x = " << x << "\ty = " << y << endl;
        return x;
}
```

2. Написать программу, в которой определяется шаблон для функции Max(x,y), возвращающей большее из значений x и y. Написать специализированную версию функции Max(const char*, const char*), возвращающую "большую" из передаваемых ей символьных строк. В каждой из функций предусмотреть вывод сообщения о том, что вызвана шаблонная или специализированная функция и вывод найденного большего. Проверить работу программы на следующих примерах

```
x = Max('a','1');
y = Max(0,1);
```

```
z = Max("Hello", "World");
obj3 = Max(obj1, obj2);
```

Здесь obj1, obj2 и obj3 - объекты класса, который нужно создать.

3. Создать шаблон функции, выводящей на экран сумму двух, переданных ей аргументов. Аргументы могут быть типов int, float, double а так же массивами элементов типов int, float, double. В последнем случае выводится сумма нулевых элементов массивов. Если типы аргументов не подходящие - выводится сообщение "не арифметические типы и не массивы элементов арифметических типов". (Только для повышенной оценки) Решите эту задачу для всех арифметических типов (и массивов элементов таких типов) средствами стандарта C++17 (см. Приложение).

4. Создать шаблон класса матриц. Переопределить операции сложения, вычитания и ввод/вывод в поток. Проверить работоспособность шаблона класса матриц для данных разных числовых типов: int, float, complex.

5. Создать (и протестировать работу) шаблон класса для реализации текстового меню, позволяющий связывать с пунктами меню указатели на функции, на статические методы и объекты-функции (функции и методы не получают аргументов).

Создать (и протестировать работу) другой шаблон класса для реализации текстового меню, позволяющий связывать с пунктами меню указатели на нестатические методы классов (без явных аргументов).

VII. Контрольные вопросы

1. Для решения каких задач удобно использовать шаблоны функций?
2. Чем шаблоны функций отличаются от перегрузки функций?
3. Как задать шаблон функции?
4. Сколько параметров может быть у шаблона функции?
5. Что такое специализированный шаблон?
6. Чем специализированный шаблон отличается от специализированной функции?

7. В программе заданы следующие конструкции:

```
template<typename T>
void f(T x, T y) { cout<<"x="<<x<<"\ty="<<y<<endl; }

void f(double x, double y) { cout<<"x="<<x<<"\ty="<<y<<endl; }

template<typename T>
void f(double x, double y) { cout<<"x="<<x<<"\ty="<<y<<endl; }

int main() {
        int a = 5;
        double d = 1.5;
        f( a, d );
        return 0;
}
```

Что произойдет: синтаксическая ошибка, вызов функции, будет создана и вызвана шаблонная функция?

8. Для решения каких задач удобно использовать шаблоны классов?
9. Как задать шаблон класса?
10. Что такое параметры-типы и параметры-не типы?
11. Что такое параметры-шаблоны?
12. Как создать специализированную версию шаблона класса?
13. Какими должны быть значения по умолчанию для параметров-типов?
14. Какими должны быть значения по умолчанию для параметров-не типов?

Приложение

Стандарт C++17 содержит новую управляющую конструкцию

if constexpr (выражение) оператор;

и версию с конструкцией else.

Она позволяет компилировать (значение выражения == true) или нет отдельные участки кода, т.е. обрабатывается не во время выполнения программы, как обычный if, а на этапе компиляции. Соответственно и выражение должно быть таким (константным), чтобы его можно было вычислить при компиляции, а не при выполнении программы.

В MS Visual Studio 2017 с последним обновлением можно включить опцию, заставляющую компилятор рассматривать конструкции стандарта C++17 через свойства проекта:

Properties->C/C++->Language->C++ Language standard - ISO C++17 Standard (/std:c++17).

Литература

1. Г. Шилдт. "Самоучитель C++". – СПб.: БХВ-Петербург. – 2001, 688 с.

2. Стенли Б. Липпман, Жози Лажойе, Барбара Э. Му «Язык программирования C++. Базовый курс», 5-е изд., М.:Вильямс, 2014, 1120 с.

3. Стивен Пратта «Язык программирования C++ (C++11). Лекции и упражнения», 6-е изд., М.:Вильямс, 2014, 1248 с.

Редактор Т.В. Колесникова

В печать

Объем 0.75 усл.п.л. Офсет. Формат 60x84x16.

Бумага тип №3. Заказ № . Тираж 50 экз. Цена свободная

Издательский центр ДГТУ

Адрес университета и полиграфического предприятия:

344000, г. Ростов-на-Дону, пл. Гагарина, 1.