Информация для размещения на официальном сайте ГБПОУ «Светлоградский региональный сельскохозяйственный колледж»

Для электронного обучения

Группа	421
Дата	27.11.2021 г
Время	10-10-11-00
Наименование УД/МДК/УП/ПП	МДК 01.02
Ф.И.О. преподавателя	Сахарчук Т.В.
Электронная почта	saharchyk777@mail.ru
Основная литература	1. Интеллектуальные системы и технологии. Учебник и практикум для СПО Станкевич Л. А. Научная школа: Санкт-Петербургский политехнический университет Петра Великого (г. Санкт-Петербург). Год: 2019 / Гриф УМО СПО https://biblio-online.ru/book/intellektualnye-sistemy-i-tehnologii-445852 2. Федорова Г.Н. Разработка и администрирование баз данных (2-е изд., стер.) учебник«Академия»2019 г. 3. Федорова Г.Н. Информационные системы (6-е изд., стер.) учебник «Академия» 2019г 4. Рудаков А.В. Технология разработки программных продуктов (12-е изд.) учебник«Академия»2018 г.
Тема № 77-78	Практическое занятие на тему: Использование шаблонов классов

Задание	**Функции элементов в шаблонах классов**
	
	Функции-члены могут быть определены как внутри шаблона класса, так и за его пределами. В последнем случае они определяются как шаблоны функций.
	
	// member_function_templates1.cpp
	template<class T, int i> class MyStack
	{
	T* pStack;
	T StackBuffer[i];
	static const int cItems = i * sizeof(T);
	public:
	MyStack(void);
	void push(const T item);
	T& pop(void);
	};
	
	template< class T, int i > MyStack< T, i >::MyStack(void)
	{
	};
	
	template< class T, int i > void MyStack< T, i >::push(const T item)
	{
	};
	
	template< class T, int i > T& MyStack< T, i >::pop(void)

```
{
};

int main()
{
}
```

Обратите внимание, что как и в функциях-членах класса шаблона, определение функции-члена для конструктора класса подразумевает, что список аргументов шаблона приводится дважды.

Функции-члены сами могут быть шаблонами функций, в которых указываются дополнительные параметры, как показано в следующем примере.

```
// member_templates.cpp
template<typename T>
class X
{
public:
  template<typename U>
  void mf(const U &u);
};

template<typename T> template <typename U>
void X<T>::mf(const U &u)
{
}

int main()
{
}
```

Шаблоны вложенных классов

Шаблоны можно определить в классах или шаблонах классов (в этом случае они называются шаблонами членов). Шаблоны членов, которые являются классами, называются шаблонами вложенных классов. Шаблоны элементов, которые являются функциями, обсуждаются в шаблонах функций элементов.

Шаблоны вложенных классов объявляются как шаблоны классов внутри области внешнего класса. Их можно определить во включающем классе или вне его.

В следующем примере кода демонстрируется шаблон вложенного класса внутри обычного класса.

```
// nested_class_template1.cpp
// compile with: /EHsc
```

```
#include <iostream>
using namespace std;

class X
{

  template <class T>
  struct Y
  {
    T m_t;
    Y(T t): m_t(t) { }
  };

  Y<int> yInt;
  Y<char> yChar;

public:
  X(int i, char c) : yInt(i), yChar(c) { }
  void print()
  {
    cout << yInt.m_t << " " << yChar.m_t << endl;
  }
};

int main()
{
  X x(1, 'a');
  x.print();
}
// nested_class_template2.cpp
// compile with: /EHsc
#include <iostream>
using namespace std;

template <class T>
class X
{
  template <class U> class Y
  {
    U* u;
  public:
    Y();
    U& Value();
    void print();
    ~Y();
  };

  Y<int> y;
public:
  X(T t) { y.Value() = t; }
  void print() { y.print(); }
};
```

```
template <class T>
template <class U>
X<T>::Y<U>::Y()
{
  cout << "X<T>::Y<U>::Y()" << endl;
  u = new U();
}

template <class T>
template <class U>
U& X<T>::Y<U>::Value()
{
  return *u;
}

template <class T>
template <class U>
void X<T>::Y<U>::print()
{
  cout << this->Value() << endl;
}

template <class T>
template <class U>
X<T>::Y<U>::~Y()
{
  cout << "X<T>::Y<U>::~Y()" << endl;
  delete u;
}

int main()
{
  X<int>* xi = new X<int>(10);
  X<char>* xc = new X<char>('c');
  xi->print();
  xc->print();
  delete xi;
  delete xc;
}

//Output:
X<T>::Y<U>::Y()
X<T>::Y<U>::Y()
10
99
X<T>::Y<U>::~Y()
X<T>::Y<U>::~Y()
```

Локальные классы не могут иметь шаблоны элементов.

Друзья в шаблоне

Шаблоны классов могут иметь друзей. Дружественными объектами класса-шаблона могут быть классы или шаблоны классов, функции или шаблоны функций. Ими также могут быть специализации (кроме частичных) шаблонов классов или шаблонов функций.

В следующем примере дружественная функция определена как шаблон функции в шаблоне класса. Этот код создает по одной версии дружественной функции для каждого экземпляра шаблона. Такую конструкцию можно использовать в тех ситуациях, когда дружественная функция зависит от тех же параметров шаблона, что и класс.

```
// template_friend1.cpp
// compile with: /EHsc

#include <iostream>
using namespace std;

template <class T> class Array {
  T* array;
  int size;

public:
  Array(int sz): size(sz) {
    array = new T[size];
    memset(array, 0, size * sizeof(T));
  }

  Array(const Array& a) {
    size = a.size;
    array = new T[size];
    memcpy_s(array, a.array, sizeof(T));
  }

  T& operator[](int i) {
    return *(array + i);
  }

  int Length() { return size; }

  void print() {
    for (int i = 0; i < size; i++)
      cout << *(array + i) << " ";

    cout << endl;
  }

  template<class T>
```

```
  friend Array<T>* combine(Array<T>& a1, Array<T>& a2);
};

template<class T>
Array<T>* combine(Array<T>& a1, Array<T>& a2) {
  Array<T>* a = new Array<T>(a1.size + a2.size);
  for (int i = 0; i < a1.size; i++)
    (*a)[i] = *(a1.array + i);

  for (int i = 0; i < a2.size; i++)
    (*a)[i + a1.size] = *(a2.array + i);

  return a;
}

int main() {
  Array<char> alpha1(26);
  for (int i = 0 ; i < alpha1.Length() ; i++)
    alpha1[i] = 'A' + i;

  alpha1.print();

  Array<char> alpha2(26);
  for (int i = 0 ; i < alpha2.Length() ; i++)
    alpha2[i] = 'a' + i;

  alpha2.print();
  Array<char>*alpha3 = combine(alpha1, alpha2);
  alpha3->print();
  delete alpha3;
}
//Output:
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
a b c d e f g h i j k l m n o p q r s t u v w x y z
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z a b c d e f
g h i j k l m n o p q r s t u v w x y z
```

В следующем примере создается дружественная функция, которая имеет специализацию шаблона. Если исходный шаблон функции является дружественным объектом, то и его специализация автоматически становится дружественной.

Кроме того, как отмечается в комментарии перед объявлением дружественной функции в следующем коде, вы можете взять только специализированную версию шаблона и объявить ее в качестве дружественной. В этом случае определение дружественной специализации шаблона необходимо поместить за пределами класса шаблона.

```
// template_friend2.cpp
// compile with: /EHsc
#include <iostream>
```

```
using namespace std;

template <class T>
class Array;

template <class T>
void f(Array<T>& a);

template <class T> class Array
{
   T* array;
   int size;

public:
   Array(int sz): size(sz)
   {
      array = new T[size];
      memset(array, 0, size * sizeof(T));
   }
   Array(const Array& a)
   {
      size = a.size;
      array = new T[size];
      memcpy_s(array, a.array, sizeof(T));
   }
   T& operator[](int i)
   {
      return *(array + i);
   }
   int Length()
   {
      return size;
   }
   void print()
   {
      for (int i = 0; i < size; i++)
      {
         cout << *(array + i) << " ";
      }
      cout << endl;
   }
   // If you replace the friend declaration with the int-specific
   // version, only the int specialization will be a friend.
   // The code in the generic f will fail
   // with C2248: 'Array<T>::size' :
   // cannot access private member declared in class 'Array<T>'.
   //friend void f<int>(Array<int>& a);

   friend void f<>(Array<T>& a);
};

// f function template, friend of Array<T>
```

```
template <class T>
void f(Array<T>& a)
{
   cout << a.size << " generic" << endl;
}

// Specialization of f for int arrays
// will be a friend because the template f is a friend.
template<> void f(Array<int>& a)
{
   cout << a.size << " int" << endl;
}

int main()
{
   Array<char> ac(10);
   f(ac);

   Array<int> a(10);
   f(a);
}
//Output:
10 generic
10 int
```

В следующем примере показан дружественный шаблон класса, объявленный в пределах шаблона класса. Затем этот шаблон класса используется в качестве аргумента шаблона для дружественного класса. Дружественные шаблоны классов должны определяться за пределами шаблона класса, в котором они объявлены. Все специализации или и частичные специализации дружественного шаблона также являются дружественными объектами исходного шаблона класса.

```
// template_friend3.cpp
// compile with: /EHsc
#include <iostream>
using namespace std;

template <class T>
class X
{
private:
  T* data;
  void InitData(int seed) { data = new T(seed); }
public:
  void print() { cout << *data << endl; }
  template <class U> friend class Factory;
};

template <class U>
class Factory
```

```
{
public:
  U* GetNewObject(int seed)
  {
    U* pu = new U;
    pu->InitData(seed);
    return pu;
  }
};

int main()
{
  Factory< X<int> > XintFactory;
  X<int>* x1 = XintFactory.GetNewObject(65);
  X<int>* x2 = XintFactory.GetNewObject(97);

  Factory< X<char> > XcharFactory;
  X<char>* x3 = XcharFactory.GetNewObject(65);
  X<char>* x4 = XcharFactory.GetNewObject(97);
  x1->print();
  x2->print();
  x3->print();
  x4->print();
}
//Output:
65
97
A
a
```

Повторное использование параметров шаблона

Параметры шаблона могут повторно использования в списке параметров шаблона. Например, приведенный ниже код допустим:

```
// template_specifications2.cpp

class Y
{
};
template<class T, T* pT> class X1
{
};
template<class T1, class T2 = T1> class X2
{
};

Y aY;

X1<Y, &aY> x1;
```

	X2<int> x2;
	
	int main()
	{
	}

Контрольный тест	Все программы представленные в лекции набрать на языке С++ и результаты показать преподавателю. Ответьте на вопросы: 1. Сферы применения языка С++. Пример простой программы 2. История создания языка и его эволюция С++ 3. Структура простой программы на языке С++, использования транслятора и запуск программы на выполнение 4. Имена или идентификаторы языка С++ 5. Переменные и Константы языка С++ 6. Все операции языка С++. 7. Операторы-выражения. Объявления имен 8. Операторы управления. Условные операторы. Операторы выбора. 9. Операторы цикла. Оператор возврата. Оператор перехода 10. Вызов функций. Имена функций 11. Необязательные аргументы функций. Рекурсия 12. Встроенные типы языка С++ 13. Определение методов класса. Переопределение операций 14. Подписи методов и необязательные аргументы. Запись классов 15. Классы и объекты 16. Производные типы данных 17. Массивы. Структуры. 18. Битовые поля. Объединения. Указатели 19. Связь между массивами и указателями. 20. Безтиповый (нетипизированный) указатель. Нулевой указатель. Строки и литералы 21. Распределение памяти 22. Автоматические переменные. Статические переменные. 23. Динамическое выделение памяти. Выделение памяти под строки. 24. Рекомендации по использованию указателей и динамического распределения памяти. 25. Распределение памяти при передаче аргументов функции. Рекомендации по передаче аргументов. Ссылки 26. Наследование, виды наследования. Виртуальные методы 27. Абстрактные классы. Множественное наследование 28. Производные классы, наследование 29. Интерфейс и состояние объекта. Объявление friend. 30. Использование описателя const. Доступ к объекту по чтению и записи 31. Контроль доступа к объекту 32. Конструкторы классов. Возможности инициализации объектов 33. Деструкторы классов 34. Копирующий конструктор. Операции new и delete 35. Переопределение операций. Как определять операции. 36. Преобразования типов. Явные преобразования типов 37. Стандартные преобразования типов. Преобразования указателей и ссылок. 38. Преобразования типов, определенных в программе 39. Компоновка нескольких файлов в одну программу. Проблема использования общих функций и имен. 40. Препроцессор. Определение макросов. Условная компиляция 41. Файлы и переменные. Общие данные. 42. Глобальные переменные. Повышение надежности обращения к общим данным 43. Попытка классификации ошибок. Сообщение об ошибке с помощью возвращаемого значения. 44. Исключительные ситуации. Обработка исключительных ситуаций,

	операторы try и catch. 45. Потоки в языке C++. 46. Манипуляторы и форматирование ввода-вывода 47. Строковые потоки. Ввод-вывод файлов. 48. Понятие шаблона. 49. Функции-шаблоны 50. Использование шаблонов классов

Дата____27.11.2021 г _____ _____ _____

Подпись Ф.И.О. преподавателя