БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
МЕХАНИКО-МАТЕМАТИЧЕСКИЙ ФАКУЛЬТЕТ
Кафедра веб-технологий и компьютерного моделирования

А. С. Кравчук, А. И. Кравчук, Е. В. Кремень

Тест

Объектно ориентированное программирование на C++. Библиотека STL

Учебные материалы
для студентов специальности 6-05-0533-07
«Математика и компьютерные науки
(по профилизациям)»

МИНСК
2023

УДК 004.432.045(075.8)+004.428(075.8)
ББК 32.973-018.1я73-1
К78

Рекомендовано советом
механико-математического факультета БГУ
30 мая 2023 г., протокол № 9

Рецензент
профессор кафедры экономической информатики
Белорусского государственного экономического университета
кандидат технических наук, доцент *А. М. Седун*

Кравчук, А. С.

К78 Тест. Объектно ориентированное программирование на C++. Библиотека STL : учеб. материалы для студентов специальности 6-05-0533-07 «Математика и компьютерные науки (по профилизациям)» / А. С. Кравчук, А. И. Кравчук, Е. В. Кремень. – Минск : БГУ, 2023. – 42 с.

Приводятся вопросы по основам объектно ориентированного программирования на C++, шаблонам, исключениям, библиотеке STL. Они предназначены для оценки и закрепления теоретических знаний по соответствующим темам.

УДК 004.432.045(075.8)+004.428(075.8)
ББК 32.973-018.1я73-1

Оглавление

Введение

В последнее время появляется тенденция к развитию в образовательных учреждениях системы аттестационного контроля с помощью тестирования, которая связана с тем, что проведение теста является наиболее эффективным средством оценки успеваемости учащихся. Тестирование позволяет повысить уровень объективности устанавливаемой оценки по результатам обучения и, как следствие, исключить возникновение конфликтов на почве образовательного процесса.

Именно в области контроля знаний информационные технологии развиваются активней и глубже. В западных странах тестирование повсеместно вытеснило традиционные формы контроля такие как устная сдача экзаменов, письменные работы, собеседования. В последнее время белорусская система образования активно приближается к европейским стандартам.

Объектно-ориентированное программирование – это способ создания программного кода, при применении которого важнейшими составляющими программ выступают объекты. В данном случае под объектом понимается комплекс свойств (данных, типичных для него), их методов обработки, а также средств обработки событий, на которые этот объект может отзываться.

Актуальность изучения концепции ООП заключается в том, что оно является востребованной современной парадигмой программирования. Это требует от студента ее глубокого освоения. Этому должна способствовать предлагаемая система тестов.

Следует обратить внимание, что изложенные в данном пособии тематические вопросы является лишь базой, которую каждый преподаватель может изменить и/или дополнить в соответствии со своим видением методики преподавания предмета.

Основы объектно-ориентированного программирования

1. Класс – это:
   - любой тип данных, определяемый пользователем;
   - тип данных, определяемый пользователем и начинающийся со служебного слова class;
   - тип данных, определяемый пользователем и начинающийся со служебного слова struct, имеющий защищенные от внешнего доступа поля (свойства);

2. Структура – это:
   - любой тип данных, определяемый пользователем;
   - тип данных, определяемый пользователем и начинающийся со служебного слова stuct;
   - тип данных, определяемый пользователем и начинающийся со служебного слова struct, не имеющий защищенные от внешнего доступа поля;

3. Членами класса могут быть:
   - как поля, так и компонентные функции, объявленные как private, protected, так и public;
   - только поля, объявленные как private;
   - только компонентные функции, объявленные как private;
   - только поля и компонентные функции, объявленные как private;
   - только поля и компонентные функции, объявленные как public;

4. Что называется конструктором?
   - метод, имя которого совпадает с именем класса и который вызывается автоматически при инициализации объекта класса;
   - метод, имя которого совпадает с именем класса и который вызывается автоматически при объявлении класса (до создания объекта класса);
   - метод, имя которого необязательно совпадает с именем класса и который вызывается при создании объекта класса;
   - метод, имя которого совпадает с именем класса и который всегда необходимо явно вызывать из головной программы при объявлении объекта класса;

5. Объект – это:
    - переменная, содержащая указатель;
    - экземпляр класса;
    - класс, который содержит в себе свойства и методы их обработки;

6. Что называется деструктором?
    - метод, который удаляет объект;
    - метод, который освобождает память, занимаемую объектом;
    - системная функция, которая освобождает память, занимаемую объектом;

7. Что называется наследованием?
    - это механизм, посредством которого производный класс получает элементы родительского и может дополнять либо изменять его свойства и методы;
    - это механизм только переопределения методов базового класса;
    - это механизм, посредством которого производный класс получает только поля базового класса;
    - это механизм, посредством которого производный класс получает элементы родительского, может их дополнить, но не может переопределить;

8. Выберите правильное утверждение.
    - деструктор не наследуется;
    - деструктор наследуется, но должен быть перегружен;

9. Выберите правильное объявление производного класса.
    - class MoreDetails::Details;
    - class MoreDetails: public class Details;
    - class MoreDetails: public Details {};
    - class MoreDetails: class(Details);

10. Выберите правильное утверждение.
    - если элементы класса объявлены как private, то они доступны только наследникам класса, но не внешним функциям;
    - если элементы класса объявлены как private, то они недоступны ни наследникам класса, ни внешним функциям;
    - если элементы объявлены как public, то они доступны наследникам класса, но не внешним функциям;

11. Возможность и способ обращения производного класса к элементам базового определяется …
    - спецификаторами доступа: private, public, protected в теле производного класса;
    - только спецификатором доступа protected в заголовке объявления производного класса;
    - спецификаторами доступа: private, public, protected в заголовке объявления производного класса;
    - спецификаторами доступа: private, public, protected в теле базового класса;

12. Выберите правильное соответствие между спецификатором базового класса, спецификатором доступа при наследовании и правами доступа производного класса к элементам базового:
    - спецификатор доступа при наследовании - public; в базовом классе: private; права доступа в производном классе – protected;
    - спецификатор доступа при наследовании - protected или public ; в базовом классе: protected; права доступа в производном классе – protected;
    - спецификатор доступа при наследовании - private; в базовом классе: public; права доступа в производном классе – public;

13. Дружественная функция – это:
    - функция другого класса, среди аргументов которой есть элементы данного класса;
    - функция, объявленная в классе с атрибутом friend, но не являющаяся членом класса;
    - функция, являющаяся членом данного класса и объявленная с атрибутом friend;
    - функция, которая в другом классе объявлена как дружественная данному;

14. Выберите правильное утверждение.
    - одна функция может быть дружественной нескольким классам;
    - дружественная функция не может быть обычной функцией, а только методом другого класса;
    - дружественная функция не может быть методом другого класса;

15.Шаблон функции – это...

- определение функции, в которой типу обрабатываемых данных присвоено условное обозначение;
- прототип функции, в котором вместо имен параметров указан условный тип;
- определение функции, в котором указаны возможные варианты типов обрабатываемых параметров;
- определение функции, в котором в прототипе указан условный тип, а в определении указаны варианты типов обрабатываемых параметров;

16.Выберите правильное утверждение:

- по умолчанию члены класса, объявленного с помощью служебного слова class имеют атрибут private;
- по умолчанию члены класса, объявленного с помощью служебного слова struct, имеют атрибут public;
- функции-члены класса имеют доступ только к элементам public;

17.Переопределение (перегрузка) операций имеет вид:

- имя_класса, ключевое слово operation, символ операции;
- имя_класса, ключевое слово operator, символ операции, в круглых скобках могут быть указаны аргументы;
- имя_класса, ключевое слово operator, список аргументов;
- имя_класса, два двоеточия, ключевое слово operator, символ операции;

18.Для доступа к элементам объекта используются:

- при обращении через имя объекта – точка, при обращении через указатель – операция «->»;
- при обращении через имя объекта – два двоеточия, при обращении через указатель – операция «точка»;
- при обращении через имя объекта – точка, при обращении через указатель – два двоеточия;
- при обращении через имя объекта – два двоеточия, при обращении через указатель – операция «->»;

19.Полиморфизм – это:

- средство, позволяющее использовать одно имя для обозначения действий, общих для родственных классов;

- средство, позволяющее в одном классе использовать методы с одинаковыми именами;
- средство, позволяющее в одном классе использовать методы с разными именами для выполнения одинаковых действий;
- средство, позволяющее перегружать функции для работы с разными типами аргументов или их разным количеством;

20.Полиморфизм реализован через механизмы:
- перегрузки функций, виртуальных функций, шаблонов;
- перегрузки функций, наследования методов, шаблонов;
- наследования методов, виртуальных функций, шаблонов;
- перегрузки функций, наследования, виртуальных функций;

21.Виртуальными называются функции:
- базового класса, имеющие спецификатор virtual;
- базового класса без спецификатора, но переопределенные в производном классе
- базового класса без спецификатора, но не использующиеся в производном классе
- базового класса, которые не могут быть переопределены в базовом классе
- производного класса, имеющие спецификатор virtual и переопределенные относительно базового класса

22.Полиморфизм в объектно-ориентированном программировании реализуется:
- через механизмы перегрузки (функций и операций), виртуальные функции и шаблоны;
- через механизмы перегрузки (функций и операций) и шаблоны;
- через виртуальные функции и шаблоны;
- через механизмы перегрузки (функций и операций) и виртуальные функции;

23.Пусть описан класс:

```
class ClassA {int a; public: ClassA(); ClassA(int ); };
```

Какое из следующих выражений противоречит синтаксису C++
- ClassA B;
- ClassA B=ClassA(2);
- ClassA B(2);
- ClassA B();

24.Выберите правильные утверждения:

- у конструктора могут быть параметры;
- конструктор наследуется, но должен быть перегружен;
- конструктор должен явно вызываться всегда перед объявлением объекта;
- конструктор вызывается автоматически при инициализации объекта;
- объявление каждого класса должно содержать явно прописанный конструктор;
- если конструктор не создан, компилятор создаст его автоматически;

25.Отметьте правильные утверждения:

- конструкторы класса не наследуются;
- синтаксис конструкторов класса никак не регламентируется;
- конструкторов класса может быть несколько, но их синтаксис должен подчиняться правилам перегрузки функций;
- конструктор возвращает указатель на объект;
- конструктор не возвращает значение;

26.Дано определение класса:

```
class monstr {
   int health, armo;
public:
   monstr(int he=50, int arm=10);
   int color;
}
```

Укажите свойства и методы, доступные внешним функциям

- health, armo, monstr();
- monstr();
- color; monstr();
- health, armo, color, monstr();

27.Стиль ООП рекомендует объявлять свойства классов:

- со спецификатором доступа private;
- со спецификатором доступа public;
- без спецификаторов доступа;
- со спецификатором доступа local;
- со спецификатором доступа global;

28. Могут ли совпадать имена параметров метода и имена полей (свойств) класса?
    - да;
    - нет;

29. При описании метода перед его именем указывается:
    - имя типа возвращаемого значения;
    - имя экземпляра объекта;
    - имя поля объекта;
    - имя объекта-предка;
    - ничего не указывается;

30. Выберите правильные утверждения.
    - Действие спецификатора доступа private распространяется…
    - до другого спецификатора доступа;
    - до спецификатора доступа public;
    - до спецификатора доступа protected;
    - до конца файла;
    - до начала описания следующего поля;

31. Стиль ООП рекомендует объявлять методы со спецификатором доступа…
    - private;
    - public;
    - local;
    - global;

32. В каком месте приведенного ниже класса следует инициализировать переменную i?

```
class A{
   const int i;
   static void init(int b);
   void get(int b = 0) const;
public:
   A(int b = 0);
};
```

    - в строке объявления этого константного свойства;
    - внутри константного метода get();
    - в теле конструктора;
    - внутри статического метода init();
    - внутри функции main();

33. Что неверно при инициализации объекта y в приведенном ниже коде?

```
class SomeClass {
   int a;
public:
   SomeClass (int b) : a(b) {};
};
…
SomeClass x(10);
SomeClass y(x);
```

- это неверно, потому что SomeClass не содержит public конструктора копирования;
- это неверно, потому что SomeClass не содержит виртуального деструктора;
- это неверно, потому что SomeClass не содержит конструктора по умолчанию;
- здесь нет ничего неверного. С инициализацией y все в порядке;
- это неверно, потому что SomeClass не содержит конструктора копирования;

34. Какое определение правильно описывает класс со статическим полем и правильно инициализирует его?

- class bad {static int i;}; int bad::i = 99;
- class bad {static i;}; int bad:i = 99;
- class bad {int i = 99;}; int bad::i = 99;
- class bad {static int i = 99;};
- class bad {static int i = 99;}; int i::bad = 99;

35. Какие из следующих утверждений верные?

- константный метод объявляется с ключевым словом const после списка параметров;
- константный объект может вызывать только константные методы;
- константный метод может изменять значения полей класса;
- константный метод может вызывать любые методы класса;
- константный метод может вызываться только для константных объектов;
- константный метод может изменять значения константных полей класса;

36. Какие из следующих утверждений верные?
    - константный метод объявляется с ключевым словом const перед указанием типа возвращаемого значения;
    - константный метод не может изменять значения полей класса;
    - константный метод может вызываться для любых (не только константных) объектов;
    - константный метод не может вызывать константные методы класса;
    - константный метод может изменять значения константных полей класса;

37. Какая операция используется для доступа к открытым полям (свойствам) объекта?
    - операция «->»
    - операция «.»
    - операция «,»
    - операция «::»
    - операция «*»

38. Какая операция используется для доступа к открытым полям объекта через указатель на объект?
    - операция «->»
    - операция «.»
    - операция «,»
    - операция «::»
    - операция «*»

39. Что можно сказать о приведенном примере?

```
class A{
   int a;
public:
   A(int b){a = b;}
};
int main(){
   A c(3);
   cout << c.a << endl;
   return 0;
}
```

    - программа будет работать нормально;
    - при компиляции программы возникнет ошибка;
    - программа будет компилироваться, но не будет компоноваться;
    - при выполнении программы возникнет ошибка;

40. Какое из следующих утверждений относительно классов и структур истинно?
    - структура может включать в себя только базовые типы данных (int, char и т.д.);
    - структура не может иметь конструкторов и деструкторов;
    - структура не может быть наследником класса;
    - класс не может быть наследником структуры;
    - поля структуры открыты для внешнего доступа по определению;

41. Какие из следующих утверждений правильные?
    - поля класса могут быть описаны с использованием ключевого слова static;
    - поля класса могут быть описаны с использованием ключевого слова const;
    - поля класса могут иметь тип самого класса;
    - поля класса не могут быть указателями на объект этого же класса;

42. Почему конструктор копирования должен иметь в качестве параметра ссылку?
    - потому что иначе при передаче параметра по значению конструктор будет вызываться рекурсивно;
    - потому что функция-член класса может иметь в качестве параметра только ссылку;
    - потому что переменная, передаваемая по ссылке, может быть изменена;
    - потому что использование ссылки дает более эффективный код;

43. Какие из следующих утверждений неверные?
    - конструкторы глобальных объектов вызываются до вызова функции main();
    - локальные объекты создаются, как только порядок выполнения программы доходит до их объявления;
    - возможно получение указателя на конструктор;
    - конструктор запускается при создании каждого локального объекта;

44. Какие из следующих утверждений неверные?
    - конструктор возвращает значение типа void;
    - класс может иметь несколько конструкторов с разными параметрами для разных видов инициализации;
    - конструктор не может иметь параметров;
    - нельзя получить указатель на конструктор;

45. Какое из следующих утверждений верное?
    - конструктор не возвращает значение;
    - конструктор возвращает значение типа void;
    - конструкторы наследуются;
    - в конструкторах нельзя задавать значения параметров по умолчанию;

46. Какой конструктор вызывается при описании нового объекта с инициализацией другим объектом этого же типа?
    - конструктор по умолчанию;
    - конструктор копирования;
    - конструктор с параметром-объектом;
    - в этом случае никакой конструктор не вызывается;

47. Какой конструктор вызывается при выполнении операции присваивания между существующими объектами одного типа?
    - конструктор по умолчанию;
    - конструктор копирования;
    - конструктор с параметром-объектом;
    - в этом случае никакой конструктор не вызывается;

48. Выберите правильные варианты продолжения: «Конструктор копирования вызывается при ...»
    - при передаче объекта в функцию по ссылке;
    - описании нового объекта с инициализацией другим объектом;
    - передаче объекта в функцию по значению;
    - возврате адреса объекта из функции;
    - возврате объекта как результата работы функции;
    - выполнении операции присваивания;

49. Выберите правильные утверждения.
    - при перегрузке операций не сохраняется количество операндов, используемых в базовых типах данных;

- при перегрузке операций сохраняется количество операндов, используемых в базовых типах данных;
- при перегрузке операции с помощью метода класса первый операнд может быть любого типа;

50. Выберите правильные утверждения.
    - для базовых типов данных переопределять операции нельзя;
    - функции-операции могут иметь аргументы по умолчанию;
    - любые функции-операции не наследуются;

51. Какая из следующих операций при перегрузке в классе не должна иметь возвращаемое значение того же типа, что и сам класс?
    - *
    - -
    - <
    - =

52. Какая из следующих операций при перегрузке в классе не должна иметь возвращаемое значение того же типа, что и сам класс?
    - *=
    - -=
    - +=
    - ==

53. Какая из следующих операций при перегрузке в классе не должна иметь возвращаемое значение того же типа, что и сам класс?
    - /
    - -
    - +=
    - <=

54. Какая из следующих операций при перегрузке в классе не должна иметь возвращаемое значение того же типа, что и сам класс?
    - !=
    - *=
    - +=
    - -=

55. Какая из следующих операций не может быть перегружена?
    - +
    - -
    - ? :
    - =

56. Какой знак не может быть использован при перегрузке операции?
    - @
    - -
    - <
    - *=

57. Какая из следующих операций не может быть перегружена?
    - *
    - -
    - sizeof()
    - >=

58. Какие из следующих утверждений верные?
    - деструктор наследуется;
    - деструктор может быть виртуальным;
    - класс может иметь несколько деструкторов с различными именами;
    - деструктор может иметь только один параметр;

59. При компиляции следующего примера возникает ошибка. Чтобы ее исправить, необходимо ...

```
class A{
   int x;
public:
   A(int i){};
};
class B: virtual A{
public:
   B(){};
};
…
B b;
```

    - добавить конструктор класса "B" который получает параметр типа int;
    - добавить вызов конструктора класса A без параметров к конструктору класса B;

- добавить вызов конструктора класса предка с константным значением, например, A(10) к конструктору класса B;
- добавить виртуальный деструктор для класса B;
- добавить конструктор копирования для класса A;

60. Выберите верное утверждение.
    - дружественная функция объявляется внутри класса, к элементам которого ей нужен доступ, с ключевым словом friend;
    - в качестве параметра дружественной функции должен передаваться объект класса;
    - одна функция может быть дружественной сразу нескольким классами;
    - на дружественную функцию не распространяется действие спецификаторов доступа;

61. Выберите верное утверждение.
    - дружественной функции должна передаваться ссылка на объект класса;
    - дружественной функции передается указатель this;
    - дружественная функция может быть обычной функцией;
    - дружественная функция может быть методом другого, ранее определенного класса;

62. Возможность иметь в одном классе несколько методов с одним именем это…
    - Overloading (перегрузка);
    - Inheritance (наследование);
    - Encapsulation (инкапсуляция);

63. Какой из операторов нужно использовать для того, чтобы инициализировать значение p адресом объекта ob для приведенного ниже кода?

```
class myclass {
   int a;
public:
   myclass(int b);
};
myclass::myclass(int b) { a = b;}
int main()
{
   myclass ob(120);
   myclass *p;
…
```

- *p = myclass *ob;
- *p = *ob;
- p = &ob;
- p = *ob;
- p = ob;

64. В чем, кроме имени, различие между методами f1() и f2() в приведенном ниже коде?

```
struct A
{
   int f1() {
      return 0;
   }
   int f2();
};
int A::f2() {
   return 0;
};
```

- различий нет;
- f1() - это метод класса;
- f1() - это статическая функция;
- f2() - это статическая функция;
- f2() - имеет спецификатор доступа private;

65. Какой из предложенных вариантов записи абстрактного класса является правильным?
    - class A {virtual int f () = 0;}
    - abstract class A {virtual int f () = 0;}
    - class A {virtual int f () = 0;} abstract

Шаблоны

1. Для чего предназначена специализация шаблона?
    - для специализации обобщённых шаблонных классов под конкретные задачи;
    - для специализации под конкретное подмножество своих параметризованных типов данных;
    - для кодирования обобщённых алгоритмов, без привязки к типам данных;

2. Чем механизм шаблонов отличается от механизма перегрузки
   - перегрузка не требует единообразия алгоритмов перегружаемых функций;
   - ничем не отличается, это просто разные наименования одного и того же;
   - шаблон создается для функций, различающихся типами данных;

3. Какие основные виды шаблонов существуют в языке C++?
   - шаблоны конструкторов и деструкторов;
   - шаблоны функций и классов;
   - шаблоны классов и структур;

4. Определена шаблонная функция

```
template <class T>
bool cmp(T a, T b)
{
   return a>b? true : false;
}
```

   Как осуществить вызов функции cmp() для параметров int a1, double b1?
   - int b = 1;
     ```
     double a = 2.5;
     cout<<cmp(a, b);
     ```
   - double b = 1;
     ```
     int a = 2.5;
     cout<<cmp(a, b);
     ```
   - double b = 1;
     ```
     int a = 2.5;
     cout<<cmp(a, int(b));
     ```

5. Какое объявление функции-шаблона func верно?
   - template <class T>
     void func(T a, T b);
   - template <T>
     void func(a, b);
   - void func<T>(T a, T b);

6. Выберите правильное (синтаксически верное) объявление функции-шаблона func, принимающей два параметра параметризующего типа template <class T>

- template <struct T>
  void func(T a, T b);
- template <T>
  void func(T a, T b);
- template <class T>
  void func(T a, T b);

7. Шаблон A и его специализации объявлены следующим образом:

```
template <class T>
class A
{public:
   A(){ cout<< 1 << " ";}
};
template <>
class A<int>
{public:
   A(){cout<< 2 << " ";}
};
template <>
class A<char*>
{public:
   A(){cout<< 2 << " ";}
};
```

Какой будет результат после выполнения кода

```
A<int> a;
A<char> a1;
A<long> a2;
```

- ошибка компиляции в строке «template <> class A<int>»
- 2 3 1
- 1 1 1
- 2 1 1

8. Какой правильный вариант описания шаблона семейства классов?

- template {class T}class Array{ … }
- template <class T>class Array{ … };
- template (class T) class Array{ … };

9. Какой правильный заголовок шаблона
   - template <class t1, class t2>
   - template <class t1,t2>
   - template <class t, class t>
   - template <class t,t>

10. Имеется шаблон функции

```
template <class T>
T func1(T a,T b)
{
   if(a) a = a % b;
   return a;
}
```

Верен ли код?

```
double a = 5, b = 6, c;
c = func1(a, b);
```

   - да, все верно;
   - нет, ошибка связана с типом операндов в операторе `if`;
   - нет, ошибка связана с типом операндов в операторе `%`;

11. Существует ли в C++ готовый набор шаблонов:
   - нет;
   - да, существует специальная библиотека STL;
   - зависит от версии компилятора;

12. Можно ли в шаблоне класса определить статический метод?
   - да;
   - нет, будет ошибка компиляции;
   - да, но результат работы программы непредсказуем;

13. Отметьте все **не**верные утверждения:
   - память, занимаемая объектом класса, сгенерированного из шаблона, освобождается автоматически;
   - шаблон не является классом;
   - все классы-шаблоны наследуют класс Template;
   - объекты классов, сгенерированных из шаблонов, занимают в памяти меньше места, чем объекты других классов;

14. Сколько параметров может быть у шаблона при определении шаблона функции ?
    - 1;
    - столько, сколько аргументов у функции;
    - столько, сколько типов используется для параметризации;

15. При определении класса-шаблона
    - он должен быть включен в общий контекст (`namespace`) STL;
    - он должен быть наследован от класса `template`;
    - он должен быть отмечен ключевым словом `template`;
    - он должен включать в себя, по крайней мере, один параметр-тип;

16. Отметьте правильный заголовок шаблона функции:
    - template <class T> void Sum(T x1, T x2);
    - class template <class T> Sum
    - template <T> class
    - template class <class T> Sum(T x1, T x2);

17. Какой правильный вариант создания экземпляра объекта?

```
template <class T>
class Matrix
{
public :
    Matrix(int n, int m);
   ~Matrix() { … }
};
```

    - Matrix a(4, 5);
    - Matrix <float> a(4, 5);
    - Matrix :: <float> a(4, 5);

18. Укажите правильное объявление шаблона функции, если в программе производится вызов double x = zero<double>();
    - template <class TT>
      TT zero() { return 0;}
    - template TT
      zero() { return 0;}
    - template <class TT>
      TT zero { return 0;}

19.Укажите результат на экране:

```
template <class T>
T sum(T *a, T *b)
{
  T f = 5;
  return (*a + *b) - f;
}

int main()
{ int i = 10, j = 20;  double x = 5.1, y = 2.2;
  cout << sum(&i, &j) << " "
       << sum(&x, &y);
}
```

20.Укажите результат на экране:

```
…
template<class T>
bool function(T a, T b) {
   return a > b;
}
int main() {
   cout<< function(false, true);
   return 0;
}
```

21.Укажите результат на экране:

```
…
template<class T>
char function(T a, T b) {
   return a > b ? a : b;
}
int main() {
   cout<< function('q', 'w');
   return 0;
}
```

22.Укажите результат на экране:

```
…
template<class F, class E>
char function(F a, E b) {
   return a > b ? a : b;
}
```

```
int main() {
   cout<< 's' - function('s', 't');
   return 0;
}
```

23.Укажите результат на экране:

```
…
template<class F, class E>
int function(F a, E b) {
   return a > b ? a : b;
}
int main() {
   cout<< function(2, 1);
   return 0;
}
```

24.Укажите результат на экране:

```
…
template<class F, class E, class J>
int function(F a, E b, J c) {
   return a + b + c;
}
int main() {
   cout<< function(2.5, 1.7, 1.3);
   return 0;
}
```

25.Укажите результат на экране:

```
…
template<class E>
E function(int a, E b, int c) {
   return a + b + c;
}
int main() {
   cout<< function(2.5, 1.7, 1);
   return 0;
}
```

26.Укажите результат на экране:

```
…
template<class F, class E>
int function(F a, E b, F c) {
   return a + b + c;
}
int main() {
```

```
    cout<< function(2, 1.7, 1);
    return 0;
}
```

27.Укажите результат на экране:

```
…
template<class T>
T function(T a, float b, T c) {
    return a * b * c;
}
int main() {
    cout<< function(2, 1.5, 1);
    return 0;
}
```

28.Укажите результат на экране:

```
…
template<class T>
T function(int a, T b, int c) {
    return a * b * c;
}
int main() {
    cout<< function(2.3, 1.7, 1);
    return 0;
}
```

29.Укажите результат на экране:

```
…
template<class T>
T function(T x) {
    if (x < 0) x = -x;
    return x;
}
int main() {
    cout<< function(2.5)<<" "<< function(-1);
    return 0;
}
```

30.Укажите результат на экране:

```
…
template<class T>
void function(T & x, int n) {
    T r = 5;
```

```
    for(int i =1; i < n; r = r * x, i++);
    x = r;
}
int main() {
    int a = 5;
    function(a, 2);
    cout<<a;
    return 0;
}
```

31.Укажите результат на экране:

```
…
template<class T>
double function(T x, int n) {
    return n == 0 ? 1 : x * function(x, n - 1);
}
int main() {
    cout<< function(3, 3);
    return 0;
}
```

32.Укажите результат на экране:

```
…
template<class T>
T function(T *a, int n) {
    return n == 0 ? a[0] : a[n] + function(a, n - 1);
}
int main() {
    int a[] = {3, 4, 5};
    int n = sizeof(a)/sizeof(int);
    cout<< function(a, n - 1);
    return 0;
}
```

33.Укажите результат на экране:

```
…
template<class T>
T function(T *a, int n) {
    return n == 0 ? a[0] : a[n] * function(a, n - 1);
}
int main() {
    int a[] = {2, 2, 2, 3};
    int n = sizeof(a)/sizeof(int);
```

```
    cout<< function(a, n - 1);
    return 0;
}
```

Исключения

1. Что дает использование обработки исключительных ситуации?
   - возможность корректного завершения программы или участка программы в случае возникновения исключительной ситуации;
   - возможность выдать сообщение пользователю в случае возникновения исключительной ситуации;
   - возможность разработчику контролировать непредвиденные ситуации;
   - возможность исправлять синтаксические ошибки;

2. Сколько параметров может принимать catch?
   - ни одного;
   - несколько;
   - один;

3. Укажите результат на экране:

```
#include <iostream>
using namespace std;
short j = 4, i = 0;

void fun1()      { double x = 2; if (!i) throw x; }
void fun2()      { --j; fun1(); j++; }

int main()
{
   try
   {
     fun2();
   }
   catch (double)
   {
       cout <<"Exception ";
   }
   cout << j <<" " << i;
}
```

4. Укажите результат на экране:

```
#include <iostream>
using namespace std;
int main() {
   try {
      try {
         try {  throw 1; }
         catch (int) {  cout << "Exception 1"; }
      }
      catch (int) {  cout << "Exception 2"; }
   }
   catch (int) { cout << "Exception 3"; }
   return 0;
}
```

5. Что происходит при попытке выполнить оператор return внутри блока catch?
   - выход из функции;
   - повторное создание обрабатываемой исключительной ситуации;
   - ошибка компиляции;
   - аварийная остановка программы;
   - ошибка выполнения;

6. Что такое исключительная ситуация (или исключение)?
   - исключительная ситуация (или исключение) — это ошибка, которая возникает во время выполнения программы;
   - синтаксическая ошибка;
   - ошибка в написании операторов;

7. Что из себя представляет обработка исключений в C++?
   - это системные средства, с помощью которых программа может справиться с ошибками времени выполнения;
   - исправление логических ошибок;
   - исправление синтаксических ошибок;

8. На каких ключевых словах основана обработка исключений в C++?
   - try;
   - catch;
   - throw;
   - new;
   - delete;

9. Какие этапы включают в себя обработка исключений?
   - в try-блок помещаются программные инструкции, которые программист считает нужным проконтролировать;
   - если исключение возникает в try-блоке, то оно дает знать о себе «выбросом» определенного рода информации (с помощью ключевого слова throw);
   - «выброшенное» с помощью throw исключение может быть перехвачено программным путем с помощью catch-блока и обработано соответствующим образом;
   - catch-блок всегда должен содержать оператор безусловного перехода goto;

10. Для чего в языке C++ используется оператор thow?
    - для «сигнализации» о возникновении исключения или ошибки;
    - для «выброса» кодового значения;
    - для определения константы;

11. Что такое «генерация исключения»?
    - сигнализация о том, что произошла ошибка;
    - «выброс» кода исключения;
    - текстовое сообщение пользователю об ошибке;
    - нет правильных ответов;

12. Что может следовать за ключевым словом throw?
    - указывается значение любого типа данных (код ошибки);
    - текстовое описание проблемы;
    - создание объекта специального пользовательского класса;
    - копирование наиболее важных данных;

Библиотека STL

1. Из чего состоит STL?
   - набор согласованных обобщенных алгоритмов, контейнеров, средств доступа к их содержимому и различных вспомогательных функций;
   - набор средств работы с массивами в императивном программировании;
   - библиотека программирования интерфейса пользователя;

2. Какие алгоритмы существуют в STL?

- функции для поиска членов коллекций по признаку;
- функции выполнения перестановок в коллекциях;
- функции сортировки членов коллекции;
- функции для выполнения определенных арифметических действий над членами коллекции;
- нет верных ответов;

3. В чем преимущество использования алгоритмов перед собственноручно написанными функциями?
   - сокращение времени написания программы;
   - сокращение кода;
   - возможность универсализации кода (например, возможность его использования для различных, в том числе пользовательских типов данных);
   - нет правильного ответа;
   - преимуществ нет;

4. Опишите класс complex?
   - один из классов библиотеки STL;
   - для его использования нужно включить заголовочный файл #include <complex>;
   - предназначен для хранения двух элементов одного типа, представляющих комплексное число в его декартовой форме;
   - комплексное число может быть представлено суммой действительного числа и мнимой части;
   - является шаблоном;
   - можно конкретизировать (указать) тип вещественной и мнимой части;
   - не требует подключения дополнительных заголовочных файлов;
   - можно раздельно конкретизировать типы вещественной и мнимой частей;

5. Опишите структуру pair?
   - это структура, имеющая два поля;
   - имена полей предопределены, первое называется first, второе называется second;
   - спецификатор доступа к обоим полям – public;
   - необходимо раздельно конкретизировать типы полей first и second;
   - нельзя с помощью cin ввести сразу значения двух полей;

- при использовании операций сравнения (типа меньше, больше и т.д.) пары сравниваются сначала по полю first, а если поля first равны, то по полю second;
- требует подключения дополнительных заголовочных файлов;

6. Опишите структуру tuple?
   - структура из любого количества полей;
   - для доступа к полям tuple используется конструкция get;
   - параметр, передаваемый функции get<>() в угловых скобках – это индекс поля;
   - значение индекса поля должно быть константой, нельзя в качестве этого значения использовать переменную;
   - структуры tuple сортируются также в лексикографическом порядке сначала по нулевому полю, при равенстве нулевого поля по первому, затем по второму и т.д.;
   - требует подключения дополнительных заголовочных файлов;

7. Опишите класс string?
   - требует подключения дополнительного заголовочного файла;
   - все объявления в заголовочном файле string сделаны в именованной области пространства имен с названием std;
   - имеет несколько конструкторов: один без параметров и остальные с параметрами;
   - определены некоторые арифметические и логические операции;
   - не требует использования инструкции using namespace std;

8. Укажите допустимые операции над строками:
   - присваивание значения (=);
   - конкатенация двух строк, конкатенация строки и символа (+ или +=);
   - посимвольное сравнение (==, !=);
   - лексикографическое сравнение (<, >, <=, >=);
   - операция обращения к отдельным символам ([ ]);
   - умножение двух строк (*);
   - отрицание строки (!);

9. При вводе строки с помощью std::cin:
   - строка вводится вне зависимости от количества пробелов;
   - строка вводится до появления первого пробела;

10. Что такое контейнер?
    - класс (шаблон) STL, реализующий функциональность некоторой структуры данных;
    - участок кода с перегруженными функциями;
    - совокупность классов связанных наследованием;

11. Что такое итератор?
    - структура данных, которая «указывает» на некоторый элемент контейнера, и (для некоторых контейнеров) умеет переходить к предыдущему/следующему элементу;
    - это объект, который позволяет перемещаться по элементам некоторой последовательности;
    - индекс элемента контейнера;

12. Операции, которые можно выполнять с любыми итераторами:
    - проверка двух итераторов на равенство (==);
    - проверка двух итераторов на неравенство(!=);
    - инкремент (увеличение итератора), то есть переход к следующему элементу контейнера (++);
    - декремент (уменьшение итератора), то есть переход к предыдущему элементу контейнера (--);
    - операция разыменования (*);
    - побитовый сдвиг влево или право (<<, >>);

13. Выберете верные утверждения:
    - до стандарта 2011 года ключевое слово auto использовалось для явного указания, что переменная должна иметь автоматическую продолжительность жизни;
    - в настоящее время ключевое слово auto при инициализации переменной может использоваться вместо типа переменной, чтобы сообщить компилятору, что он должен присвоить тип переменной исходя из инициализируемого значения;
    - в настоящее время можно использовать в качестве типа возвращаемого функцией значения (но это не рекомендуется);
    - в настоящее время можно нельзя использовать ключевое слово auto в параметрах функций;

14. Какие основные элементы STL хранят различные значения и объекты?
    - контейнеры STL;
    - обобщенные алгоритмы STL;
    - итераторы STL;

- адаптеры STL;
- функциональные объекты;

15. Какие основные элементы STL абстрагируют перемещение по коллекциям объектов?
    - контейнеры STL;
    - обобщенные алгоритмы STL;
    - итераторы STL;
    - адаптеры STL;
    - функциональные объекты;

16. Каковы основные особенности использования контейнера STL вектор (vector)?
    - необходимо присоединить одноименный заголовочный файл;
    - произвольный доступ к элементам;
    - при создании контейнера с указанием только количества элементов, он будет создан пустым (нулевым в случае числовых значений);
    - размер vector-а в любое время может динамически изменяться операциями добавления в конец или удаления последнего элемента;
    - непрерывное размещения объектов контейнера в памяти;
    - размер vector-а в любое время может динамически изменяться операциями добавления в начало или удаления нулевого элемента;

17. Что необходимо для создания двумерного вектора?
    - рекурсивно использовать vector в качестве значения параметра типа в контейнере vector;
    - создать vector состоящий из vector-ов;
    - передать количество строк и столбцов двумерного массива специальному конструктору контейнера vector;

18. Какие утверждения про итераторы верны?
    - итераторы - это обобщённые указатели;
    - итераторы предназначены для обхода последовательности объектов в обобщённом контейнере;
    - итераторы можно сравнивать;
    - итераторы можно вычитать;
    - итераторы можно умножать;
    - итераторы можно склабывать;

19. Каковы основные особенности использования контейнера STL стек (stack)?
    - структура данных, в которой доступ осуществляется только к последнему элементу;
    - новые элементы добавляются в конец стека;
    - элементы удаляются также из конца стека, то есть при удалении элемента из стека удаляется последний добавленный элемент;
    - для его использования необходимо подключить одноименный заголовочный файл;
    - новые элементы можно добавлять в начало стека и удалять первый элемент стека;
    - структура данных, в которой можно осуществлять только прямой доступ к элементам;

20. Каковы основные особенности использования контейнера STL очередь (queue)?
    - структура данных, в которой доступ осуществляется только к самому раннему добавленному элементу (первому);
    - новые элементы добавляются в конец очереди, а удаляются из начала очереди;
    - для его использования необходимо подключить одноименный заголовочный файл;
    - новые элементы можно добавлять в начало стека и удалять последний элемент очереди;
    - структура данных, в которой можно осуществлять прямой доступ к элементам;

21. Каковы основные особенности использования контейнера STL дек (deque)?
    - структура данных, которая позволяет добавлять элементы и в конец, и в начало, а также удалять элементы из конца и из начала;
    - для его использования необходимо подключить одноименный заголовочный файл;
    - структура данных, в которой можно осуществлять прямой доступ к элементам;
    - структура данных, в которой новые элементы добавляются только в конец очереди, а удаляются только из начала очереди;

22. Каковы основные особенности использования контейнера STL список (list)?
    - реализует структуру данных типа «двусвязный список»;

- имеет возможность добавления (и удаления) элементов и в начало, в конец и в середину списка;
- для его использования необходимо подключить одноименный заголовочный файл;
- имеет дополнительную возможность прямого доступа, которое есть у вектора и дека;

23. Являются ли синонимами понятия «произвольный доступ к элементам» и «прямой доступ к элементам»?
    - да;
    - нет;

24. Что такое произвольный доступ к элементам данных?
    - возможность обратиться к любому элементу последовательности данных за равные промежутки времени, не зависящие от размеров последовательности;
    - чем дальше (от начала) расположен элемент совокупности данных, тем больше требуется времени для доступа;

25. Какие виды доступа встречаются в контейнерах?
    - произвольный;
    - последовательный;
    - по признаку;
    - адресный;

26. Какие методы с одинаковыми названиями имеются в контейнерах vector, queue, deque, list?
    - size();
    - empty();
    - resize();
    - push_back();
    - pop_back();

27. Что из себя представляет контейнер множество (set)?
    - структура данных, эквивалентная множествам в математике;
    - состоит из различных элементов одного заданного типа;
    - поддерживает операции добавления элемента во множество, удаления элемента из множества, проверка принадлежности элемента множеству;
    - одно и то же значение хранится во множестве только один раз;
    - множества хранятся в виде упорядоченной структуры;
    - поддерживает произвольный доступ к элементам;

28. Особенности работы с итераторами контейнера «множество»?
    - разыменование итератора (применение унарной операции (*)) возвращает значение элемента множества, на который указывает итератор;
    - определена операция инкремента (что означает переход к следующему элементу);
    - определена операция декремента (переход к предыдущему элементу);
    - итераторы можно сравнивать на равенство и неравенство;
    - определены операции сравнения итераторов при помощи <, <=, >, >=;
    - определены операции прибавления/вычитания из итератора целого числа;

29. Особенности использования алгоритмов стандартной библиотеки шаблонов (STL)?
    - при их использовании необходимо подключить заголовочный файл <algorithm>;
    - большинство из алгоритмов принимают в качестве параметров два итератора;
    - в качестве итераторов могут использоваться методы begin() и end();
    - если к итераторам определенных контейнеров применимы операции сложения и вычитания целых чисел, то алгоритмы могут применяться к части контейнера;
    - они определяют операции сравнения итераторов при помощи <, <=, >, >=;

30. Укажите результат на экране:

```
…
template<class T>
void function (vector<T> (&a)) {
   for(int i =  a.size() - 1; i >= 0; i--)
   {
      cout<< a[i] << " ";
   }
}
int main() {
   vector <int> a = {9, 7, 5, 3};
   function(a);
   return 0;
}
```

31. Укажите результат на экране:

```
…
template<class T>
T function(vector<T> (&a)) {
   T r = *a.begin();
   for(auto it = a.begin() + 1; it < a.end(); it++) {
      if (r > *it) r = *it;
   }
   return r;
}
int main() {
   vector <double> a = {1.5, 2.3, 3.2, 4.7};
   cout<<function(a);
   return 0;
}
```

32. Укажите результат на экране:

```
…
template<class T>
int function(vector<T> (&a)) {
   auto it_res = a.begin();
   for(auto it_current = a.begin() + 1; it_current < a.end();it_current ++) {
      if (*it_res > *it_current) it_res = it_current;
   }
   return it_res - a.begin();
}
int main() {
   vector <double> a = {0.9, 1.8, 2.5, 3.8};
   cout<<function(a);
   return 0;
}
```

33. Укажите результат на экране:

```
…
template<class T>
T function(vector<T> (&a)) {
   T r = *a.begin();
   for(auto it = a.begin() + 1; it < a.end(); it++) {
      if (r < *it) r = *it;
   }
   return r;
}
int main() {
```

```
    vector <double> a = {2.5, 2.3, 3.2, 3.7};
    cout<<function(a);
    return 0;
}
```

34.Укажите результат на экране:

```
…
template<class T>
bool then(T a, T b) {
    return a < b;
}
template<class T>
T function(vector<T> (&a), bool (*ptr)(T, T)) {
    T r = *a.begin();
    for(auto it = a.begin() + 1; it < a.end(); it+=2) {
        if (ptr(r, *it)) r = *it;
    }
    return r;
}
int main() {
    vector <double> a = {3, 4, 5, 7};
    cout<<function(a, then);
    return 0;
}
```

35.Укажите результат на экране:

```
…
template<class T>
bool then(T a, T b) {
    return a > b;
}
template<class T>
T function(vector<T> (&a), bool (*ptr)(T, T)) {
    T r = *a.begin();
    for(auto it = a.begin() + 1; it < a.end(); it+=2) {
        if (ptr(r, *it)) r = *it;
    }
    return r;
}
int main() {
    vector <double> a = {3, 4, 5, 7};
    cout<<function(a, then);
    return 0;
}
```

36. Укажите результат на экране:

```
…
template<class T>
int function(vector<T> (&a)) {
   auto itRes  = a.begin();
   for(auto itCurrent = a.begin() + 2; itCurrent < a.end(); itCurrent +=2) {
      if (*itRes  < *itCurrent) itRes  = itCurrent;
   }
   return itRes - a.begin();
}
int main() {
   vector <int> a = {-1, 2, 1, 0};
   cout<<function(a);
   return 0;
}
```

37. Укажите результат на экране:

```
…
template<class T>
int function(vector<T> (&a)) {
   auto itBegin = a.begin();
   for( ; (itBegin < a.end()) && (*itBegin > 0); itBegin++);
   for(auto itCurrent = itBegin; itCurrent < a.end();itCurrent +=1) {
      if ((*itCurrent < *itBegin) && (*itCurrent < 0)) itBegin = itCurrent;
   }
   return itBegin - a.begin();
}
int main() {
   vector <int> a = {1, 4, -1, 2, -5};
   cout<<function(a);
   return 0;
}
```

38. Укажите результат на экране:

```
…
template<class T>
int function(vector<T> (&a)) {
   auto itBegin = a.begin();
   for( ; (itBegin < a.end()) && (*itBegin < 0); itBegin++);
   for(auto itCurrent = itBegin; itCurrent < a.end();itCurrent +=1) {
      if ((*itCurrent < *itBegin) && (*itCurrent > 0)) itBegin = itCurrent;
   }
```

```
    return itBegin - a.begin();
}
int main() {
    vector <int> a = {1, 4, -1, 2, -5};
    cout<<function(a);
    return 0;
}
```

39. Укажите результат на экране:

```
…
template<class T>
int function(vector<T> (&a)) {
    auto itCurrent = a.begin();
    for( ; (itCurrent < a.end()) && (*itCurrent != 0); itCurrent++);
    return itCurrent - a.begin();
}
int main() {
    vector <int> a = {1, 0, -1, 0, -5};
    cout<<function(a);
    return 0;
}
```

40. Укажите результат на экране:

```
…
template<class T>
int function(vector<T> (&a)) {
    auto itCurrent = a.end() - 1;
    for( ; (itCurrent >= a.begin()) && (*itCurrent != 0); itCurrent--);
    return itCurrent - a.begin();
}
int main() {
    vector <int> a = {1, 0, -1, 0, -5};
    cout<<function(a);
    return 0;
}
```

Литература

1. Павловская, Т.А. C/C++. Программирование на языке высокого уровня / Т.А. Павловская. - Санкт-Петербург [и др.] : Питер, 2017. - 460 с.
2. Смалюк, А.Ф. Объектно-ориентированное программирование (язык С++) / А.Ф. Смалюк, Д.В. Макарчук, Д.С. Карпович. – Институт повышения квалификации и переподготовки кадров по новым направлениям развития техники, технологии и экономики БНТУ. – Мн.: БНТУ, 2006. – 72 с.
3. Подбельский В.В. Язык Си++ / В.В. Подбельский – М.: Финансы и статистика, 2000. – 560 с.

Учебное издание

Кравчук Александр Степанович
Кравчук Анжелика Ивановна
Кремень Елена Васильевна

Тест
Объектно ориентированное программированиена C++. Библиотека STL

Учебные материалы
для студентов специальности 6-05-0533-07
«Математика и компьютерные науки
(по профилизациям)»

В авторской редакции

Ответственный за выпуск *Е. В. Кремень*

Подписано в печать 28.08.2023. Формат 60×84/16. Бумага офсетная.
Усл. печ. л. 2,56. Уч.- изд. л. 2,39. Тираж 50 экз. Заказ

Белорусский государственный университет.
Свидетельство о государственной регистрации издателя, изготовителя,
распространителя печатных изданий № 1/270 от 03.04.2014.
Пр. Независимости 4, 220030, Минск.

Отпечатано с оригинал-макета заказчика
на копировально-множительной технике
механико-математического факультета
Белорусского государственного университета.
Пр. Независимости 4, 220030, Минск.