Лабораторная работа № 3

«Объектно-ориентированное программирование»

Бригадные задания

Варианты 1 — 4 предлагают доработать классы строк, рассмотренные на лекции и предоставляемые в готовом виде для ознакомления. Разрешается изменять класс строки произвольным образом, в том числе переписывать имеющиеся методы.

1. Дополните класс `AdvancedString` следующими методами:
   а) Проверка, что строка начинается на заданную подстроку:
   ```
   const bool startsWith(const String& substring) const;
   ```
   б) Проверка, что строка заканчивается на заданную подстроку:
   ```
   const bool endsWith(const String& substring) const;
   ```
   в) Получение подстроки по индексу начала и длине:
   ```
   AdvancedString substring(
      const size_t start, const size_t length) const;
   ```
   г) Разбиение исходной строки на подстроки по заданной строке-разделителю. Пример: строка `"1,2,3"` с разделителем `","` дают строки `"1"`, `"2"` и `"3"`. Сигнатуру метода выберите самостоятельно и объясните свой выбор.

2. Расширьте класс `AdvancedString` следующими методами:
   а) Дополнение строки до требуемой длины заданным символом (слева):
   ```
   void padLeft(const size_t requiredLength, const char padding);
   ```
   Пример: дополнение `"123"` до длины 5 символом `'0'` — `"00123"`.
   б) Форматирование значений с использованием строки `pattern` как форматной:
   ```
   static AdvancedString format(const char* pattern, ...);
   ```
   Пример: `AdvancedString::format("%.2f p.", 5.5f) == "5,00 p."`.

   Указание. Работа с переменным числом параметров реализуется посредством стандартного заголовочного файла `<cstdarg>`.

   в) Замена всех вхождений заданной подстроки `what` на подстроку `with`:
   ```
   void replace(const String& what, const String& with);
   ```
3. Расширьте класс `AdvancedString` следующими методами:
   а) Удаление заданной позицией начала и длиной порции строки:
   ```
   void remove(const size_t start, const size_t length);
   ```
   б) Удаление из начала и с конца строки символов из указанного набора:
   ```
   void trim(const char[] undesired);
   ```

Перегрузите метод вариантом без параметров, удаляющим пробельные символы.

в) «Интеллектуальное» преобразование строки в аббревиатуру:

```
String toAcronym() const;
```

Примеры: `"Control & Informatics"` становится `"C&I"`, `"As soon as possible"` становится `"ASAP"`, `"Moscow Power Engineering Institute (Technical University)"` становится `"MPEI (TU)"`.

Указание. Изучите стандартные заголовочные файлы `<cctype>` и `<cstring>`.

4. Реализуйте класс `FastSearchString`, наследуемый от `AdvancedString`, использующий для поиска подстроки в методе `indexOf()` алгоритм Кнута — Морриса — Пратта (Knuth—Morris—Pratt algorithm). Вспомогательная таблица (префикс-функция) должна рассчитываться только один раз, пока строка не меняется. Разберитесь в работе алгоритма.

   Указание. В ссылках к статье об алгоритме есть пример реализации на C++.

Во всех вариантах 5 — 11 приведенный интерфейс класса — минимальный. Всюду опущены конструкторы копирования и перемещения, деструктор, операторы присвоения (копирующий и перемещающий) и сравнения, однако реализовать их необходимо.

5. Реализуйте класс для хранения динамического массива действительных чисел со следующим интерфейсом:

```
class DynamicArray {
public:
    // Создает объект и копирует в него count элементов из data.
    DynamicArray(const double *data, const size_t count);

    // Возвращает текущую длину массива.
    const size_t getLength() const;

    // Возвращает элемент по заданному индексу.
    const double operator[](const size_t index) const;

    // Устанавливает значение элемента по заданному индексу.
    void setAt(const size_t index, const double item);

    // Вставляет элемент на место, заданное индексом.
    void insertAt(const size_t index, const double item);

    // Удаляет элемент с заданным индексом.
    void removeAt(const size_t index);

    // Возвращает индекс первого элемента, равного item, или -1.
    const int find(const double item) const;
```

```
        // Вычисляет левоассоциативную свертку списка (см. ниже).
        // Первый функции свертки аргумент – элемент, второй –
        // её предыдущее значнеие при переборе списка слева направо.
        const double foldLeft(
            const double *processor(const double, const double));
};
```

Примечание. О свертке списка: http://ru.wikipedia.org/wiki/Свёртка_списка.

6. Реализуйте класс для хранения связанного списка действительных чисел со следующим интерфейсом:

```
class List {
public:
    // Объект для перебора списка. Изначально «находится»
    // на первом элементе, и может перемещаться далее.
    class Enumerator {
    public:
        // Возвращает true, если возможно продвижение вперед
        // по списку, иначе возвращает false.
        const bool canAdvance() const;
        // Выполняет продвижение на один элемент вперед.
        void advance();
        // Возвращает данные текущего элемента.
        const double getCurrent() const;
    };
public:
    // Возвращает текущую длину списка.
    const size_t getLength() const;
    // Добавляет элемент с указанными данными в конец списка.
    void append(const double item);
    // Добавляет элемент в начало списка.
    void prepend(const double element);
    // Удаляет все элементы с указанными данными из списка.
    void remove(const double item);
    // Очищает список.
    void clear();
    // Создает объект для перебора списка (см. выше).
    Enumerator getEnumerator() const;
};
```

Примечание. При изменении списка все объекты класса `List::Enumerator`, связанные с данным списком и созданные до этих изменений, считаются недействительными, и работу их методов гарантировать не нужно.

7. Реализуйте класс двусвязной очереди (deque) действительных чисел с заданным интерфейсом:

```
class Deque {
public:
    // Помещает элемент в начало очереди.
    void pushFront(const double item);
    // Извлекает из начала очереди элемент и возвращает его.
    const double popFront();
    // Возвращает элемент в начале очереди, не извлекая его.
    const double peekFront() const;
    // Помещает элемент в конец очереди.
    void pushBack(const double item);
    // Извлекает с конца очереди элемент и возвращает его.
    const double popBack();
    // Возвращает элемент в конце очереди, не извлекая его.
    const double peekBack() const;
    // Возвращает признак, что очередь пуста.
    const bool isEmpty() const;
    // Очищает очередь.
    void clear();
};
```

   Способ размещения элементов в памяти выберите самостоятельно.

8. Реализуйте класс комплексного числа (`Complex`) со всеми арифметическими операциями, а также с вычислением модуля и аргумента. На основе `Complex` реализуйте класс двухполюсника с импедансом (комплексным сопротивлением), для которого определены операции последовательного (`+`) и параллельного (`||`) включения. На основе класса двухполюсника реализуйте классы сопротивления, емкости и индуктивности, у которых доступны значения соответствующих электротехнических свойств.
9. Реализуйте классы матрицы и *n*-мерного вектора (размерности задаются при создании) со всеми арифметическими операциями: сложением, вычитанием, умножением на число, умножением матрицы на вектор. Реализуйте скалярное произведение и вычисление длины (для векторов), а также вычисление определителя и обращения (для матриц).
10. Реализуйте класс для работы с файлами на базе типа `FILE` из `<cstdio>`, обеспечивающий типобезопасные операции чтения и записи нескольких простых типов, как минимум: целых чисел (**`int`**), вещественных чисел (**`float`**), а также

строк C (`char*`) и строк демонстрационного класса `String`. Реализация безопасного копирования приветствуется, но допускается и запрет копирования.

11. Реализуйте класс `BigInteger`, при помощи которого можно было бы производить точную арифметику над неограниченно большими (по модулю) целыми числами. Требуемые операции: `+`, `-` (вычитание и отрицание), `*`, `/`, `%` (остаток от деления), `==`, `>`, `<`, `>=`, `<=`, `=`, представление в виде строки в десятичном виде.

Указание. Проще всего использовать внутреннее представление числа как динамического массива с элементами-десятичными цифрами. Возможны и гораздо более эффективные реализации.

12. Реализуйте класс т. н. «умного указателя» (smart pointer) на массив целых чисел в динамической памяти (`int*`). Поведение объекта должно быть таким, чтобы его можно было использовать как названный указатель, передав в конструкторе реальный указатель на выделенный блок памяти. Освобождение памяти должно выполняться автоматически в деструкторе объекта. Поведение при копировании и перемещении реализуйте так, как сочтете нужным, но не запрещайте эти операции.

13. Среди демонстрационного кода ЛР имеется проект `Expressions`, содержащий каркас для калькулятора арифметических выражений.
    а) Реализуйте интерфейс `IExpressionFactory`, создав систему классов, реализующих интерфейс `IExpression`, для представления различных арифметических выражений: числа, отрицания, сложения, умножения, деления, возведения в степень. Описания интерфейсов даны в комментариях. После выполнения этого этапа вычисление выражений должно работать и проходить тесты в `main()`.
    б) Решите задачу освобождения памяти под дерево выражения. Конкретный способ выберите самостоятельно и объясните свой выбор.
    в) Ознакомьтесь с реализацией класса `ExpressionParser`. Расширьте возможности разбора выражения вычислением функций *sin*, *cos*, *ln*, *exp*, *arctan* (усовершенствовав код разбора выражения и введя новый тип выражения). Создайте несколько проверок новой функциональности.
    г) Укажите преимущества и недостатки архитектуры решения задачи.

Варианты 14 — 15 предлагают создать обертку (wrapper) на C++ в объектно-ориентированном стиле над реальной библиотекой с интерфейсом для C.

Необходимо поместить заголовочные файлы и файлы `*.lib` в каталог проекта (можно в подкаталоги), а файлы `*.dll` — в каталог исполняемых файлов проекта. Для подключения библиотек к проекту необходимо указать на них компоновщику: в свойствах проекта (в его контекстном меню) на вкладке «*Project settings*» вызвать диалог «*Project's build options…*», где на вкладке «*Linker settings*» добавить нужные библиотеки в список «*Link libraries*».

14. Библиотека `libcurl` (http://curl.haxx.se/libcurl/) — это кроссплатформенное решение для передачи файлов по сети с использованием распространенных открытых протоколов (HTTP, FTP и многих других). Необходимо написать систему классов для выполнения простых действий с ее помощью:

```
// Буфер данных.
class Buffer {
public:
	// Возвращает текущий размер буфера.
	const size_t getSize() const;
	// Возвращает текущие данные буфера.
	const void* getData() const;
	// Добавляет newData размером newDataSize в конец буфера.
	void append(const void* newData, const size_t newDataSize);
	// Очищает буфер.
	void clear();
};

// Контейнер для функций работы с файлами по сети.
class NetworkFile {
public:
	// Возвращает содержимое удаленного файла по адресу url.
	static const Buffer download(const char *url);
	// Помещает данные data в удаленный файл по адресу url.
	static void upload(const char *url, const Buffer *data);
};
```

Указание. Раздел «Example sources» сайта библиотеки содержит примеры выполнения нужных действий. Ваша основная задача — инкапсулировать работу с cURL.

15. Библиотека `libxml2` (http://www.xmlsoft.org/) — популярное решение для работы с открытым форматом обмена данными XML. Необходимо реализовать систему

классов для загрузки документа XML и удобного доступа к его данным с интерфейсом:

```
// Базовый класс для узлов документа XML.
class XmlNode {
public:
    // Возвращает наименование узла.
    virtual const char* getName() const = 0;
    // Возвращает значение узла.
    virtual const char* getValue() const = 0;
};
// Атрибут дескриптора XML.
class XmlAttribute : public XmlNode {
};
// Дескриптор XML.
class XmlTag : public XmlNode {
public:
    // Возвращает количество дочерних дескрипторов.
    const size_t getChildrenCount() const;
    // Возвращает дочерний дескриптор с заданным индексом.
    const XmlTag& getChild(const size_t index) const;
    // Возвращает количество атрибутов дескриптора.
    const size_t getAttributeCount() const;
    // Возвращает атрибут с заданным индексом.
    const XmlAttribute& getAttribute(const size_t index) const;
    // Возвращает значение атрибута с заданным наименованием
    // или 0, если такого атрибута нет.
    const char* getAttributeValue(const char *name) const;
};
// Документ XML.
class XmlDocument {
public:
    // Загружает документ XML из указанного файла.
    // Возвращает признак успешной загрузки.
    bool load(const char *path);
    // Возвращает корневой элемент или 0, если документ
    // не загружен, или при загрузке возникла ошибка.
    const XmlTag* getRootNode() const;
};
```

Указание. Воспользуйтесь примером разбора документа на сайте библиотеки: http://www.xmlsoft.org/examples/index.html#tree1.c.

16. Библиотека `libiconv` (http://www.gnu.org/software/libiconv/) — распространенное кроссплатформенное решение для преобразования строк между кодировками. Создайте класс `EncodedString` (наследник `String`) со средствами для работы с различными кодировками:

а) Установление текущей кодировки:

```
const String getEnconding() const;
```

Кодировкой по умолчанию допускается считать ASCII, в которой также возвращается наименование кодировки.

б) Принудительное указание текущей кодировки:

```
void setEncoding(const char *encoding);
```

Новая кодировка возвращается далее методом `getEncoding()` и используется для преобразований.

в) Преобразование в новую кодировку при помощи *libiconv*:

```
const bool encode(const char *encoding);
```

Если произошла ошибка, возвращает **`false`** и оставляет строку в том же состоянии, что и до попытки преобразования, иначе возвращает **`true`**.