Задача №1

Реализация на абстрактния тип данни `String`

(80 точки)

Предварителни данни

Целта на задачата е да реализирате абстрактния тип данни **`String`**. Класът String трябва да поддържа динамично заделен буфер от символи, като при нужда размера на буфера трябва да се променя. Примерна организация на работата е класа е:

```
class String {
     int capacity_;
     int size_
     char* buffer_;
public:
     String(int capacity);
     String(const char* str);
     ~String();
     ...
};
```

Примерна реализация на конструкторите и деструктора е дадена по-долу. Обърнете внимание, че в буфера се съхраняват символни низове в C-стил — т.е. след последния символ се добавя терминиращ символ **`'\0'`**. Този подход е избран за да може при реализацията на класа да се използват стандартните функции за работа със низове — **`strcpy()`**, **`strlen()`**, **`strcat()`**, **`strcmp()`** и т.н.

```
String::String(int capacity)
: capacity_(capacity),
  size_(0),
  buffer_(new char[capacity])
{}
String::String(const char* str)
: capacity_(0),
  size_(0),
  buffer_(0)
{
     size_=strlen(str);
     capacity_=size_+1;
     buffer_=new char[capacity_];
     strcpy(buffer_,str);
}
String::~String() {
     delete [] buffer_;
}
```

Основни методи (20 точки)

Основните методи на класа String, които трябва да се реализират са следните:

- **`int size() const;`**
  Връща броя символи в символния низ (дължината на символния низ)
- **`int length() const;`**
  Този методи е еквивалентен на метода **`size()`**.
- **`int capacity() const;`**
  Връща капацитета (вместимостта) на символния низ, т.е. до какъв максимален размер може да достигне символния низ без да се налага динамично заделяне на нов буфер.
- **`bool empty() const;`**
  Връща стойност true, когато символния низ е празен.
- **`void clear();`**
  Изчиства символния низ и го превръща в празен символен низ.
- **`char& operator[](int index);`**
  Връща препратка към символа с индекс **`index`**. Този метод не проверява стойността на параметъра **`index`** за валидност.
- **`char& at(int index);`**
  Връща препратка към символа с индекс **`index`**. Този метод проверява стойността на параметъра **`index`** за валидност и ако тя не е валидна генерира изключение.
- **`ostream& operator<<(ostream& out, const String& str);`**
  Оператор за изход на обекти от типа **`String`** стандартните изходни потоци.
- Реализирайте операторите за сравнение като използвате **`strcmp()`**:

```
bool operator==(const String& other);
bool operator!=(cosnt String& other);

bool operator<(const String& other);
bool operator>(const String& other);

bool operator<=(const String& other);
bool operator>=(const String& other);
```

Копиращ конструктор и оператор за присвояване (20 точки)

За класа String трябва да се дефинират копиращ конструктор и оператор за присвояване.

- **`String(const String& other);`**
  Копиращ конструктор.
- **`String& operator=(const String& other);`**
  Оператор за присвояване.

Клас iterator (20 точки)

В класа **`String`** трябва да се дефинира вътрешен клас **`iterator`**, който да позволява

обхождане и манипулиране на символите в низа. В класа String трябва да се добавят методи **begin()** и **end()**, които връщат итератори насочени към първия елемент и с едно след последния елемент съответно.

Примерна дефиниция на класа **iterator** е дадена по-долу:

```
class String{
...
public:
     class iterator {
     ...
     public:
          iterator operator++();
          iterator operator++(int);
          bool operator==(const iterator& other) const;
          bool operator!=(const iterator& other) const;
          char& operator*();
     };
...
     iterator begin() const;
     iterator end() const;
};
```

Конкатенация (20 точки)

Трябва да се реализират набор от методи, които добавят символи към края на символния низ.

- **String& append(const String& other);**
  Методът **append()** добавя символния низ **other** към опашката на низа.
- **String& operator+=(const String& other);**
  Реализацията на **operator+=()** трябва да работи аналогично на метода **append()**.
- **void push_back(char ch);**
  Метода **push_back()** добавя нов символ на опашката на символния низ.
- **operator+();**
  След събиране на два символни низа трябва се създаде нов символен низ, който да съдържа конкатенацията на символните низове, които са предадени като аргументи.

Вход и изход

Програмата трябва да получи входните си данни от командният ред. Изпълнимият файл трябва да се казва prog01. Командният ред който обработва програмата трябва да изглежда по следния начин:

```
prog01 "Hello world" "Good bye"
```

Програмата трябва да извърши следните неща:

1. Да създаде две променливи **str1** и **str2** от типа **String**, които да имат за стойности низовете получени като аргументи на командния ред.
2. Да изведе на стандартния изход стойностите на двата стринга, като използва дефинирания оператор за изход:
   string 1: <Hello world>
   string 2: <Good bye>
3. Да изведе дължината на двата низа, които е получила като аргументи:
   string 1 lenght: 11
   string 2 length: 8
4. Да обходи двата низа като използва класа **iterator** и да преброи интервалите в двата низа:
   string 1 spaces: 1
   string 2 spaces: 1
5. Да сравни двата низа като използва операторите за сравнение и да изведе съобщение от вида:
   <Hello world> is greater than <Good bye>
   Първият аргумент трябва да е от лявата страна на сравнението.
6. Като използва метода **push_back()** да добави символа **'!'** в края на двата низа и да ги изведе:
   string 1: <Hello world!>
   string 2: <Good bye!>
7. Да създаде низ **str**, който да е конкатенация на низовете **str1** и **str2**:
   concatenation: <Hello world!Good bye!>
8. Да изведе дължината на кокатенирания низ:
   concatenation lenght: 21
9. Да преброи броя на интервалите в конкатенирания низ:
   concatenation spaces: 2

Изисквания към решението

1. Програмата трябва да бъде написана на езика **C++** съгласно **ISO/IEC 14882:1998**.
2. Задължително към файловете с решението да е приложен и **Makefile**. Изпълнимият файл, който се създава по време на компилация на решението, трябва да се казва **prog01**.
3. При проверка на решението програмата ви ще бъде компилирани и тествана по следния начин:
   ```
   make
   prog01 "Hello world" "Good bye"
   ```
   Предходната процедура ще бъде изпълнена няколко пъти с различни входни данни за да се провери дали вашата програма работи коректно.
4. Реализацията на програмата трябва да спазва точно изискванията описани по-горе. Всяко отклонение от изискванията ще доведе до получаване на 0 точки. По-точно: имената на всички класове и методи както и типовете на параметрите и резултатите трябва да

съвпадат с описаните по-горе.

5. Работи, които са предадени по-късно от обявеното (или не са предадени), ще бъдат оценени с 0 точки.
6. Правилата за оценяване са следните. Приемаме, че напълно коректна и написана спрямо изискванията програма получава максималния брой точки — 100%. Ако в решението има пропуски, максималният брой точки ще бъде намален съгласно следните правила:
   - Програмата ви трябва да съдържа достатъчно коментари. Оценката на решения без коментари или с недостатъчно и/или мъгляви коментари ще бъде намалена с 30%.
   - Всеки файл от решението трябва да започва със следният коментар:

```
//-----------------------------------------
// NAME: Ivan Ivanov
// CLASS: Xia
// NUMBER: 13
// PROBLEM: #1
// FILE NAME: xxxxxx.yyy.zzz (unix file name)
// FILE PURPOSE:
//   няколко реда, които описват накратко
//   предназначението на файла
//-----------------------------------------
```

   - Всяка функция във вашата програма трябва да включва кратко описание в следния формат:

```
//-----------------------------------------
// FUNCTION: xxyyzz (име на функцията)
//   предназначение на функцията
// PARAMETERS:
//   списък с параметрите на функцията
//   и тяхното значение
// FUNCTIONS CALLED:
//   списък на функциите, които се извикват
//   от описваната функция
//-----------------------------------------
```

   - Лош стил на програмиране и липсващи заглавни коментари ще ви костват 30%.
   - Програми, които не се компилират получават 0 точки. Под „**не се компилират**“ се има предвид произволна причина, която може да причини неуспешна компилация, включително липсващи файлове, неправилни имена на файлове, синтактични грешки, неправилен или липсващ **`Makefile`**, и т.н. Обърнете внимание, че в UNIX имената на файловете са case sensitive.
   - Програми, които се компилират, но не работят, не могат да получат повече от 50%. Под „**компилира се, но не работи**“ се има предвид, че вие сте се опитали да решите проблема до известна степен, но не сте успели да направите пълно решение. **Безсмислени или мъгляви програми ще бъдат оценявани с 0 точки, независимо че се компилират**. Често срещан проблем, който спада към този случай, е че вашият **`Makefile`** генерира изпълним файл, но той е именуван с име, различно от очакваното (т.е. **`prog01`** в разглеждания случай).
   - Програми, които дават неправилни или непълни резултати, или програми, в които изходът и/или форматирането се различава от изискванията ще получат не повече от 70%.

- Всички наказателни точки се сумират. Например, ако вашата програма няма задължителните коментари в началото на файлове и функциите се отнемат 30%, ако няма достатъчно коментари се отнемат още 30%, компилира се, но не работи правилно — още 30%, то тогава резултатът ще бъде:
  `50*(100-30-30-30)%= 50*10%= 5 точки`.
- Работете самостоятелно. Групи от работи, които имат твърде много прилики една с друга, ще бъдат оценявани с 0 точки.
- Вашата програма трябва да съдържа следните файлове:
  1. Заглавен файл **`String.hh`**, който съдържа дефинициите на класа **`String`** и класа **`String::iterator`**.
  2. Файл **`String.cc`**, който съдържа всички имплементации на методите, декларирани в заглавния файл **`String.hh`**. Заглавния файл не трябва да съдържа дефиниции на методи.
  3. Файл **`main.cc`**, в който е дефинирана главната функция на програмата **`main()`**.
  4. Файл **`Makefile`**, който компилира описаните по-горе файлове и създава изпълним файл с име **`prog01`**. Обърнете внимание, че ако вашето решение не съдържа **`Makefile`** или съдържа некоректен **`Makefile`**, то най-вероятно ще попадне в категорията „**не се компилира**“.