Labb 5 String - Deep Copy Problem

Denna laboration går ut på att tydliggöra problemen som uppstår när ett objekt:

- ”äger” minne på heapen
- ska kunna kopieras och tilldelas värden

samt ge ett enkelt exempel av hur container klasser fungerar för att ge bättre förståelse för STL:s container classes. Vi gör dock en container för char så vi slipper trassla med templates. Nästa laborationsuppgift ”Iterator” kommer att bygga vidare på denna klass. Titta gärna på den uppgiften när du implementerar denna. I denna laboration ska en String klass (stort S för att skilja från std::string) implementeras, den ska fungera som en förenklad std::string och std:vector<char>.

String klassen ska ha:

Funktionerna ska fungera som i std::string	**Kommentar**
~String()	
String()	
String(const String& rhs)	
String(String&& rhs)	bara för VG
String(const char* cstr)	
String& operator=(const String& rhs)	
String& operator=(String&& rhs)	bara för VG
explicit operator bool()	True om != “”. Är detta en bra ide?
char& at(size_t i)	indexerar med range check *“Bounds checking is performed, exception of type std::out_of_range will be thrown on invalid access.”*
char& operator[](size_t i)	indexerar utan range check
const char* data() const;	gives a reference to the internal array holding the string, it must also be null character terminated (meaning that there must be an extra null character last in your array)
int size() const;	finns i container klasserna i STL, se basic_string
void reserve(size_t);	finns i container klasserna i STL, se basic_string
int capacity() const;	finns i container klasserna i STL, se basic_string
void shrink_to_fit()	till skillnad från std så krävs här att utrymmet krymps maximalt så String tar så lite utrymme som möjligt.
void push_back(char c)	lägger till ett tecken sist
Void resize(size_t n)	Ändrar antalet tecken till n, om n>length så fylls det på med ”char()”
String& operator+=(const String& rhs)	tolkas som konkatenering.
operator+	ok med medlemsfunktion
friend bool operator==(const String& lhs, const String& rhs)	global function
friend bool operator!=(const String& lhs, const String& rhs)	global function

```
friend std::ostream& operator<<(std::ostream& cout, String& rhs) {
   for (size_t i = 0; i < rhs.????; ++i)
      cout << rhs.???? [i];
   return cout;
}
```

Byt ???? mot något lämpligt

Tips: Om ni gör ”inline” funktioner så kostar funktionsanropet inget så det är möjligt att utnyttja att flera av konstruktorerna och operatorerna gör liknande saker. T.ex. så kan man implementer operator+ med hjälp av operator += . Observera också värdesemantiken:

```
String c(”huj”), d(”foo”);
c=d;
```

om vi nu ändrar på c eller d så ska det inte påverka värdet på den andra variabeln.

Testprogrammet i Main.cpp
Observera att det testprogram som finns i Main.cpp bara är en hjälp och varken fullständigt eller garanterat helt korrekt. Det är möjligt att testprogrammet kör felfritt fast er lösning är felaktig. Det är även möjligt – men inte troligt – att er lösning är korrekt fast tesprogrammet inte kör/kompilerar felfritt.

”Const problemet”

Man får inte ändra på en ”const variabel” och därför så behövs två versioner av de funktioner som lämnar ut en reference. T.ex.
char& operator[](int i) {….}” för anrop av normala String.
const char& operator[](int i) const {….}” för anrop av const String.m
Kompilatorn kommer att använda const varianten när man indexerar på en const String.

Exceptions safty:

Programmet ska uppfylla det striktare kravet på exceptionsäkerhet, dvs. det som också kallas rollback semantik. Detta är möjligt att göra då vi vet har operation på typen char inte kan ge exception utan det är bara minnesallkoeringar som kan ställa till med problem.

1. Krav för G

Implementera specifikationen ovan. Tänk på att ha med const där det är lämpligt. Det får inte finnas minnesläckor och det ska vara exceptionsäkert.

2. Krav för VG

Lägg märke till att några av kraven för VG delvis motsäger det som står ovan.
Förutom kraven för G så ska ni:

- Ha alla ”const” exakt rätt.
- För en del funktioner bör man även ha en const och en icke const version.
- Implementera en så kallad move konstruktor se: http://en.cppreference.com/w/cpp/language/move_constructor. Den ska vara maximalt effektiv.
- Implementera även en move assignment operator.
- Det hela ska vara ”maximalt” effektivt – fast gå inte till överdrift ⏪:
    - Om ni t.ex. samlat större delen av koden för konstruktorerna i en hjälpfunktion så kostar det inte mycket - särskilt om ni ”inlinar” den.
    - Ni ska däremot tänka er att ni har mycket långa strängar, då kostar onödig kopiering av dem.
    - All onödig allokering av dynamiskt minne kostar!
    - Om ni förlänger strängen t.ex. genom upprepade push_back tills den har längden N så skall det kosta max O(N) amorterat, t.ex. så om ni gör tusen push_back så ska det inte bli fler än ca 2000 (=1000*2) char som har kopierats.

- operator[](int i) skall uppfylla *”if pos == size(), a reference to the character with value CharT() (the null character) is returned.”*

Invarianter och assert.

Ni skall dokumentera er klass invarianter med en privat funktion Invariant, t.ex. för FltPtr:

```
bool Invariant()  { return ptr==nullptr || *ptr == *ptr; }
```

ni ska sedan på några ställen i början och/eller slutet av metoder kolla anropa den med assert.

```
assert(Invariant());
```

syftet är att ni nu har en dokumentation som är korrekt.