Labb 5 Deep Copy problemet

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.

Uppgift:
Implementera en String klass (stort S för att skilja från STL klassen), den ska fungera ungefär som en förenklad std::string och std:vector<char>.

String klassen ska ha:

Funktion som fungera som i 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
`String& operator=(const char* cstr)`	
`String& operator=(char ch)`	
`char& at(int i)`	indexerar med range check *“Bounds checking is performed, exception of type std::out_of_range will be thrown on invalid access.”*
`char& operator[](int 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 length() const;`	finns i container klasserna i STL, se basic_string
`void reserve(int);`	finns i container klasserna i STL, se basic_string
`int capacity() const;`	finns i container klasserna i STL, se basic_string
`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
`resize(int n)`	Ändrar antalet tecken till n, om n>length så fylls det på med ”char()”
`String& operator+=(const String& rhs)`	tolkas som konkatenering.
`String& operator+=(char* cstr)`	tolkas som konkatenering.
`operator+`	ok med medlemsfunktion
`friend bool operator==(const String& lhs, const String& rhs)`	global function
`operator<<`	För test: Görs enklast genom att konvertera till std::string och skriva ut den.

Tips: Om ni gör ”inline” funktioner så kostar funtkonsanrop inget så det är möjligt att utnyttja att flera av konstruktorerna och operatorerna gör liknande saker. T.ex. så gör operator+ och

operator += nästan samma sak. Det brukar vara bäst att implementera += först och använda den för att göra +.

Semantik för klassen.

```
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 tesprogram 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.

1. Krav för G

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

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, se nedan.
- 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 kan däremot tänka er att ni har mycket långa strängar, då kostar onödig kopiering av dem.
    - All onödig allokering av dynamiskt minne kostar!
- operator[](int i) ändra så den uppfyller *”if pos == size(), a reference to the character with value CharT() (the null character) is returned.”*

1.1 ”Append problemet”

Om man i en ”container” lägger till saker sist så tar för eller senare ”capaciteten” slut och nytt minne måste allokeras och data kopieras. Ni ska göra en lösning på detta som gör att tidskomplexiteten blir linjär och inte kvadratisk vid många tillägg.

1.2 ”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.