Labb 1: Replace

Övning i pekare och c-strängar. Ni ska implementera en Replace funktion som byter ut ett ord mot ett annat i en sträng. Den ska fungera ungefär som strings replace metod i C#.
Observera att de två första labbarna är en ”uppmjukning” och betydligt enklare än de labbar som kommer senare.

En implementation av Replace för std::string kan se ut så här:

```
string Replace(const string str, const string searchFor, const string changeTo) {
    string result(str);
    for (int i = 0; i < result.length();) {
        if (result.compare(i, searchFor.size(), searchFor) == 0) {
            result.replace(i, searchFor.size(), changeTo);
            i += changeTo.size();
        }
        else
            ++i;
    }
    return result;
}
```

Att göra det med c-strängar är besvärligare! Det saknas motsvarigheter till string::compare och string::replace. Sen måste ni ta hand om allokeringen också. Exempel på anrop:

Replace("Hej på dej", "ej", "ig") vilket ska ge resultatet "Hig på dig".

Man ska inte byta ut i det som redan är utbytt dvs.

Replace("Hejjj", "ej", "eje") ger resultatet "Hejejj".

Replace(”abcabcbc”, ”abc”, ”ya”) ska ge resultatet ”yayabc”

Deklaration:

```
char * Replace(const char* str, const char* searchFor, const char* changeTo);
```

Samtliga indata ska vara oförändrade vilket är markerat med const.

Utdata ska allokeras på heapen med new.

Algoritm:

1. Räkna hur många gånger `searchFor` gånger förekommer.
2. Använd resultatet för att räkna ut hur långt resultatet blir och allokera utdatasträngen på heapen.
3. Gå igenom hela strängen med en pekare på indata och på utdata:
    a. Kolla om `searchFor` gånger finns på denna plats
    b. i så fall lägg `changeTo` finns i resultatet
    och sätt pekarna i in och ut data rätt.
    c. annars flytta över input tecknet till resultatet och inkrementera pekarna
    d. ta nästa tecken.

Det är lämpligt att göra några hjälp funktioner, jag använde:

```
bool Cmp(const char *str, const char* searchFor);

int Count(const char* str, const char* searchFor);
```

Special krav: Ni ska inte använda indexering och forloopar med heltal som kontrollvariabler utan i stället använda pekare genomgående.

Programmeringsstil och minnesläckor:

För alla program i denna kurs så krävs det att koden är rimligt snygg, i flera fall kommer vi även att kräva att den är rimligt effektiv. Det är tre saker som absolut inte får förekomma:

- missledande namn, dvs. namn som leder tankarna fel! Det är bättre att ha en variabel ”i” än en som heter ”storlek” när det är styrvariabeln i en forloop vi pratar om.

- Inkonsekvent formatering och dylikt i programmet. Ni får göra hur nu vill OM ni är konsekventa.

- Minnesläckor.

För att undvika minnesläcker så är det enklast att:

1. Först i alla kompileringsenheter (dvs. normalt antingen i första .h filen ni inkluderar eller först i själva .cpp filen) så skriver ni:

```
#ifdef _DEBUG
#define new new ( _NORMAL_BLOCK , __FILE__ , __LINE__ )
#endif
```

2. Låt början av main vara:

```
int main() {
    _CrtSetDbgFlag(_CRTDBG_ALLOC_MEM_DF | _CRTDBG_LEAK_CHECK_DF);
```

När ni nu har avslutat ert program så titta i output-förnstret, där står eventuell minnesläcka beskriven. Ni kan dubbelklicka på raden med filnamnet så kommer ni att hamna på den raden som gjorde ”new” på det minnesblock som aldrig avallokerades. Skapa en minnesläcka i ert program och se hur det ser ut - det är bara att skriva ”new int;” så har ni skapat en minnesläcka.

Vi kommer att kräva att ni har minnesläckcheck på i era program för att blir godkända.