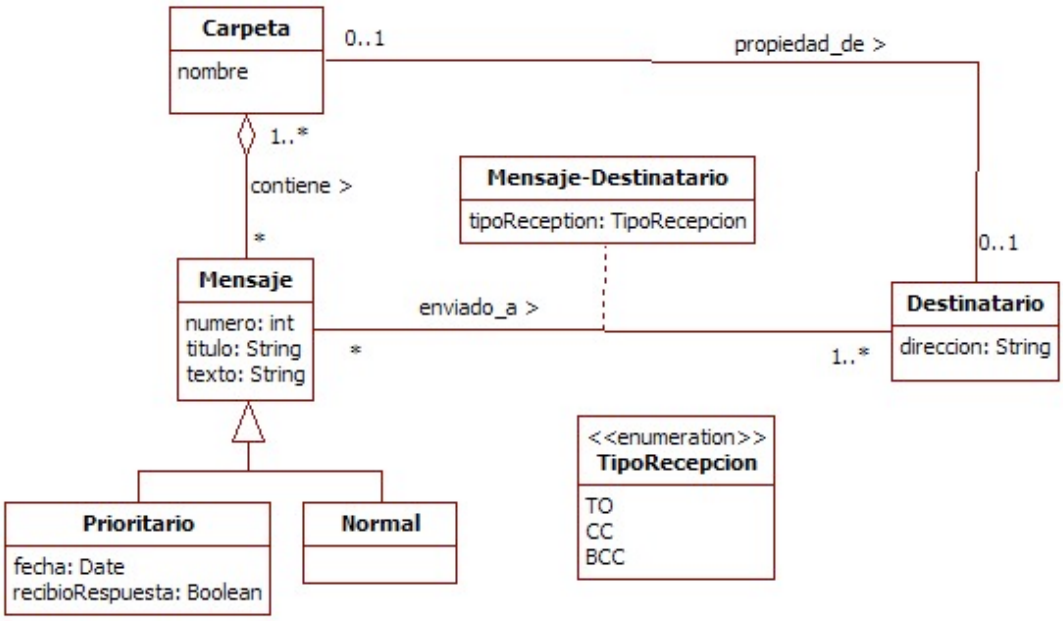
Problema 1 (25 puntos)

a) Se está desarrollando un software para gestionar el envío de mensajes (mail) de un usuario y en este proceso se ha relevado lo siguiente:

El sistema apunta a soportar las funcionalidades clásicas de envío de mensajes. En este sentido, cada mensaje tendrá un título y un texto, así como un número único asignado por el sistema. Además, cada mensaje será enviado a un conjunto de destinatarios (al menos uno) de los que se conocerá su dirección electrónica. A esta dirección serán enviados los mensajes y al mismo tiempo servirá para identificar a los destinatarios. No todos los destinatarios recibirán el mensaje de la misma forma. En particular, existen tres tipos de recepción de un mensaje: normal (TO), con copia (CC), con copia oculta (BCC). Por ende, para cada mensaje se desea conocer de qué forma recibe el mensaje cada destinatario (un destinatario podrá recibir cada mensaje sólo de una forma).

Por otro lado, el sistema apunta a incluir algunas funcionalidades que faciliten el almacenamiento y búsqueda de los mensajes. Para ello se desea poder crear carpetas en las cuales se puedan almacenar los diferentes mensajes que se envían. Cada carpeta será identificada por un nombre y particularmente existirá una carpeta (de nombre ENVIADOS) en la que se guardarán por defecto todos los mensajes enviados. El sistema tendrá la particularidad de que un mensaje podrá ser almacenado en más de una carpeta. Adicionalmente, se podrán definir carpetas cuyo contenido sea propiedad de un destinatario, es decir, que contengan mensajes enviados (de cualquier manera) a un destinatario particular. Estas carpetas las definirá el usuario en la medida de lo necesario. Una última funcionalidad permitirá determinar la prioridad de un mensaje y controlar si se recibió alguna respuesta al mismo. Para ello es necesario determinar cuándo un mensaje es prioritario, y en este caso, asociar al mismo la fecha de envío.

a) Se pide: modelar la realidad planteada mediante un Diagrama de Modelo de Dominio UML y expresar todas las restricciones del modelo en lenguaje natural.



Restricciones:

- El atributo `nombre` identifica a la `Carpeta`
- El atributo `direccion` identifica al `Destinatario`
- El atributo `numero` identifica al `Mensaje`
- La `Carpeta` asociada a un `Destinatario` está compuesta por mensajes enviados a dicho `Destinatario`
- Existe una `Carpeta` de nombre `ENVIADOS` que contiene todas las instancias de `Mensaje`

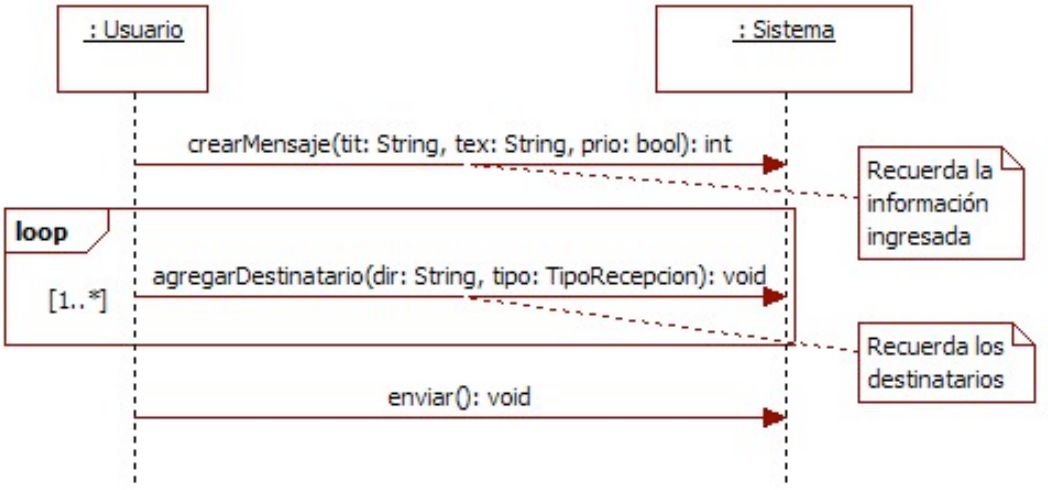
b) Como parte del relevamiento se especificó el siguiente caso de uso:

Nombre	Envío de Mensaje
Actores	Usuario
Sinopsis	El caso de uso comienza cuando el usuario desea enviar un mensaje. Para ello el usuario crea un nuevo mensaje indicando el título del mismo y el texto que contiene, así como indica si el mensaje es prioritario o no. Ante esto, el sistema registra el mensaje asignándole un número que lo identifica. Luego, el usuario ingresa el conjunto de destinatarios que recibirán el mensaje, indicando para cada uno el tipo de recepción del mensaje. El sistema registra la información y controla que haya al menos un destinatario para el envío del mensaje. Finalmente, el usuario envía el mensaje y el sistema almacena el mismo en la carpeta por defecto, así como en las propias de los destinatarios, en caso de que existan.

Se pide:

i. Realizar el Diagrama de Secuencia del Sistema (DSS) correspondiente al caso de uso.

DSS con Memoria



ii. Expresar las pre- y post-condiciones, en lenguaje natural, de los contratos correspondientes a las operaciones del DSS anterior, de acuerdo al modelo de dominio realizado en la parte a).

Operación	`crearMensaje(tit:string, tex:string, prio:bool):int`
Pre	
Post	• Si `prio=false`, entonces se crea una instancia de `Normal` con `titulo=tit`, `texto=tex` y `numero` generado • Si `prio=true`, entonces se crea una instancia de `Prioritario` con `titulo=tit`, `texto=tex`, `numero` generado y `fecha` de hoy • Se retorna el `numero` generado • Se recuerda la instancia de `Mensaje`

Operación	`agregarDestinatario(dir:string, tipo:TipoRecepcion)`
Pre	• Existe una instancia de `Mensaje` recordada • Existe una instancia de `Destinatario` con `direccion=dir` • No existe un link entre la instancia de `Mensaje` recordada y la de `Destinatario` con `direccion=dir`
Post	• Se crea una instancia de `Mensaje-Destinatario` con `tipoRecepcion=tipo` y se asocia con la instancia de `Mensaje` recordada y la de `Destinatario` con `direccion=dir`

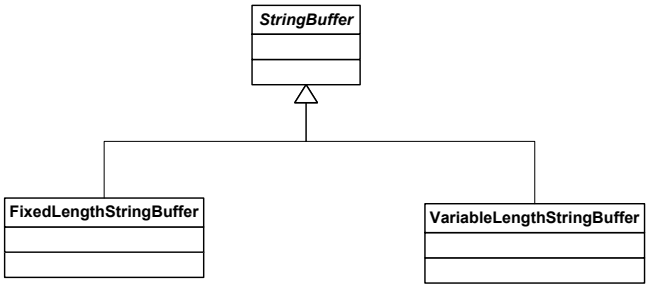
Operación	`enviar()`
Pre	• Existe una instancia de `Mensaje` recordada • Existe al menos un link entre la instancia de `Mensaje` y alguna instancia de `Destinatario`
Post	• Se crea un link entre la instancia de `Mensaje` recordada y la de `Carpeta` con `nombre="ENVIADOS"` • Para cada instancia de `Destinatario` linkeada con la instancia de `Mensaje`, si existe un link entre la instancia de `Destinatario` y una instancia de `Carpeta`, entonces se crea un link entre esa instancia de `Carpeta` y la del `Mensaje` • Se deja de recordar información

c) Ver diapositivas del Teórico del curso.

Problema 2 (15 puntos)

a) Implementar en C++ el datatype String, que represente cadenas de caracteres de largo variable.
b) Agregar la sobrecarga de los siguientes operadores:
   - Concatenación (operador +).
   - Acceso a un caracter del String (operador []).
c) Agregar la siguiente operacion:
   - *substring*, que retorna el substring que se encuentra entre dos posiciones dadas de la cadena (incluyendo los caracteres en dichas posiciones).
d) Agregar manejo de excepciones de forma que:
   - Al recibir un parámetro inválido en una operación se lance la excepción "std::invalid_argument".
e) El datatype String implementado representa una cadena de caracteres de valor variable. Si se quiere modificar dicha cadena, se puede utilizar un StringBuffer. La clase StringBuffer posee varios métodos como ser inserción, reemplazo y borrado de caracteres del buffer. Existen varias formas de representar un StringBuffer. Dos posibles tipos de buffers son:

- **FixedLengthStringBuffer**, especialización de StringBuffer de largo fijo. En los casos que las operaciones sobrepasan el largo del buffer, el mismo se trunca.
- **VariableLengthStringBuffer**, especialización de StringBuffer de largo variable. Si las operaciones sobrepasan el largo del buffer, el mismo se redimensiona al largo necesario.



Implementar en C++ la clase StringBuffer (abstracta) y las dos especializaciones definidas, con las siguientes operaciones:
- *insert*, que permite insertar un String en el buffer a partir de una posición dada.
- *capacity*, que retorna el largo del buffer.

```
#ifndef __STRING_HH__
#define __STRING_HH__

#include <iostream>
#include <stdexcept>

using namespace std;

class String{
      private:
              char* str;
              int   largo;

    public:
           String();
           String(const String&);
           String(char*);
           ~String();
           String& operator=(const String&);
           String& operator=(char*);
           String operator+(const String&);
           String operator+(char*);
           bool operator==(const String&);
           bool operator==(char*);
           bool operator!=(const String&);
           bool operator!=(char*);
           bool operator<(const String&);
           bool operator<(char*);
           bool operator>(const String&);
           bool operator>(char*);
           bool operator<=(const String&);
           bool operator<=(char*);
           bool operator>=(const String&);
           bool operator>=(char*);
           char operator[](int);
           int Length();
           String Substring(int,int);
           //extras
           char* getStr();
           void lectura(istream&);
           void escritura(ostream&);
         void setStr(char*);
         void setLargo(int);



};

// manipulaciones de char*

void copiarString(char*, char*, int);
void agregarString(char*, char*, int, int);
int comparaString(const char*, const char*,int,int);
int Strlen(const char*);
```

```
// manejo de flujo

istream& operator>> (istream&, String&);
ostream& operator<< (ostream& , const String);

// sobrecargas char*-String

String operator+(char*, String&);
bool operator==(char*, String&);
bool operator!=(char*, String&);
bool operator<(char*, String&);
bool operator>(char*, String&);
bool operator<=(char*, String&);
bool operator>=(char*, String&);

#endif
```

String.cpp

```
#include "String.h"
#include <stdexcept>
#include <iostream>

using namespace std;

// manipulaciones de char* //

void copiarString( char* a, char* b, int l){
    for (int k = 0; k<=l; k++)
     a[k] = b[k];
}
void agregarString(char *a, char*b, int largo, int l){
     if (l!=0)
        for (int k = 0; k<=l; k++)
            a[k+largo] = b[k];
}
int Strlen(const char*a){
    if (a==NULL)
        throw invalid_argument ("invalid_argument");
    int largo=0;
    while (a[largo]!='\0'){
          largo++;
          }
    return largo;
    }
int comparaString(const char* a, const char*b,int largo_a, int
largo_b){
     bool iguales= true;
     bool menor= false;
     bool mayor= false;
     int i = 0;
     int k = 0;
     for (int j=0; ((j<largo_a)&&(i<largo_b)&&(iguales)); j++){
            iguales = (a[j] == b[i]);
            if ((!iguales)&&(a[j]<b[i])){
              menor = true;
            }
            if ((!iguales)&&(a[j]>b[i])){
             mayor = true;
            }
            i++;
            k++;
     }
     if (iguales){
        if (k<largo_a){//a>b
              return 1;
        }else{
              if (i<largo_b){ //a<b
                    return -1;
              }else{
                    return 0;
```

```
                }
             }
       }else{// no son iguales, se corto el for
            if (menor){
                 return -1;
            }
            if (mayor){
                return 1;
            }
       }
}


// funciones extras de string //

char* String:: getStr() {
      return str;
}
int String::Length() {
    return largo;
}
void String::setStr(char*a){
    str=a;
}
void String::setLargo(int l){
    largo=l;
}
void String::lectura(istream& entrada){
    char car;
    String cadenaleida;
    entrada.get(car);
    while ((car==' ') || (car=='\n')|| (car=='\t')||
(car=='\r')|| (car=='\f')|| (car=='\v'))
           entrada.get(car);
    char * aux = new char [2];
    aux[0]=car;
    aux[1]='\0';
    while ((car!='\n')){
     aux[0]=car;
    cadenaleida=(cadenaleida + aux);
    entrada.get(car);
    }

    this->largo=cadenaleida.Length();
    if (this->str!=NULL)
     delete[] this->str;
    this->str= new char[this->largo+1];
    copiarString(this->str,cadenaleida.getStr(),this->largo);
    delete[] aux;


}
void String::escritura(ostream& salida){
     salida<<str;
```

```
}


// funciones de string //

String::String(){

    largo = 0;
    str = new char[1];
    str[0] = '\0';
}
String::String(const String& a){

    str = new char[a.largo+1];
    copiarString(str,a.str, a.largo);
    largo = a.largo;
}
String::String( char* a){
   if (a==NULL)
       throw invalid_argument ("invalid_argument");
   int l = Strlen(a);
   this->str = new char[l+1];
   copiarString(str,a, l);
   largo = l;
}
String::~String(){
    delete[] this->str;
}
String& String::operator=(const String& a) {
    int l =a.largo;
    delete [] str;
    str =new char [l+1];
    copiarString(str,a.str, l);
    largo=l;
    return *this;
}
String& String::operator=(char*a){
    if (NULL==a)
       throw invalid_argument("invalid_argument");
    int l= Strlen(a);
    delete [] str;
    largo=l;
    str= new char [l+1];
    copiarString(str,a,l);
    return *this;
}
String     String::operator+(const String& a){
    String retorno;
    int l=a.largo;
    retorno.largo=largo+l;
    char* aux = new char[largo+l+1];
    copiarString(aux,str,largo);
    agregarString(aux,a.str,largo,l);
    retorno.str=aux;
```

```
    return retorno;
}
String String::operator+(char*a){
       String retorno;
       int l= Strlen(a);
       retorno.largo=largo+l;
       char* aux = new char[largo+l+1];
       copiarString(aux,str,largo);
       agregarString(aux,a,largo,l);
       retorno.str=aux;
       return retorno;
}
bool String::operator==(const String& a) {
     return comparaString(str,a.str,largo,a.largo)==0;
}
bool String::operator==(char*a) {
     return comparaString(str,a,largo,Strlen(a))==0;
}
bool String::operator!=(const String&a) {
     return comparaString(str,a.str,largo,a.largo)!=0;
}
bool String::operator!=(char*a) {
     return comparaString(str,a,this->largo,Strlen(a))!=0;
}
bool String::operator<(const String&a) {
     return comparaString(str,a.str,largo,a.largo)==-1;
}
bool String::operator<(char*a) {
     return comparaString(str,a,this->largo,Strlen(a))==-1;
}
bool String::operator>(char*a) {
     return comparaString(str,a,this->largo,Strlen(a))==1;
}
bool String::operator>(const String&a) {
     return comparaString(str,a.str,largo,a.largo)==1;
}
bool String::operator<=(const String&a){
     int aux =comparaString(str,a.str,largo,a.largo);
     return ((aux==-1)||(aux==0));
}
bool String::operator<=(char*a){
     int aux =comparaString(str,a,this->largo,Strlen(a));
     return ((aux==-1)||(aux==0));
}
bool String::operator>=(const String&a){
     int aux =comparaString(str,a.str,largo,a.largo);
     return ((aux==1)||(aux==0));
}
bool String::operator>=(char*a){
     int aux =comparaString(str,a,this->largo,Strlen(a));
     return ((aux==1)||(aux==0));
}
char String::operator[](int i){
     if ((i<0) || (i>=largo))
```

```
        throw out_of_range("out_of_range");
     return (str[i]);
}
String String::Substring(int inf,int sup) {

    if (sup<inf)
            throw invalid_argument("invalid_argument");
    if ((inf<0)||(sup>=largo)||(sup<0)||(inf>=largo))
            throw out_of_range("out_of_range");

    int cant = sup - inf + 2;
    String retorno;
    retorno.str = new char[cant];
    int i=0;

    for(int j = inf; j <= sup; j++){
            retorno.str[i] = str[j];
           i++;
    }
    retorno.largo=i;
    retorno.str[i] = '\0';
    return retorno;
}


// funciones por fuera del data type string

istream& operator>> (istream& entrada, String&a){
         a.lectura(entrada);
         return entrada;
}
ostream& operator<< (ostream& salida, String a){
        a.escritura(salida);
        return salida;
}
String operator+(char*a, String& b){

   int l = Strlen(a) + b.Length();
   int l2 = Strlen(a);
   char* aux = new char[l+1];
   copiarString(aux,a,l2);
   agregarString(aux,b.getStr(),l2,b.Length());
   String ret;
   ret.setStr(aux);
   ret.setLargo(l);
   return ret;

}
bool operator==(char*a, String& st) {
     return
comparaString(a,st.getStr(),Strlen(a),st.Length())==0;
}
bool operator!=(char*a, String& st) {
     return
```

```
comparaString(a,st.getStr(),Strlen(a),st.Length())!=0;
}
bool operator<(char*a, String& st) {
     return
comparaString(a,st.getStr(),Strlen(a),st.Length())==-1;
}
bool operator>(char*a, String& st) {
     return
comparaString(a,st.getStr(),Strlen(a),st.Length())==1;
}
bool operator<=(char*a, String& st){
     int aux
=comparaString(a,st.getStr(),Strlen(a),st.Length());
     return ((aux==-1)||(aux==0));
}
bool operator>=(char*a, String& st){
     int aux
=comparaString(a,st.getStr(),Strlen(a),st.Length());
     return ((aux==1)||(aux==0));
}
```

```
#ifndef __STRINGBUFFER_HH__
#define __STRINGBUFFER_HH__

#include "String.h"


class StringBuffer{

    private:
     String str;
     int capacidad;
     int cant_chars;

    public:
     StringBuffer();
     StringBuffer(int);
     StringBuffer(String,int);
     StringBuffer(char *,int);
     virtual ~StringBuffer();
     virtual void insert (int, String) = 0;
     virtual void insert (int, char*) = 0;
     int capacity();
     int length();
     String str_de_buf();

    protected:
     String getStrBuff();
     int getCap();
     int getCantChars();
     void setCap(int);
     void setCantChars(int);
     void setStrBuff(String);

};


#endif
```

StringBuffer.cpp

```
#include "StringBuffer.h"
#include "String.h"

using namespace std;


StringBuffer::StringBuffer(){
    capacidad=256; // lo crea con una capacidad inicial de 256
    cant_chars=0;
    str=String();
}

StringBuffer::StringBuffer(int cap){
    if (cap<0) {
     throw invalid_argument ("invalid_argument");
    }
    capacidad=cap;
    cant_chars=0;
    str=String();

}

StringBuffer::StringBuffer(String a,int cap){
    if (cap<0) {
     throw invalid_argument ("invalid_argument");
    }
    capacidad=cap;
    cant_chars=a.Length();
    str=a;

}

StringBuffer::StringBuffer(char* a,int cap){
    if (cap<0) {
     throw invalid_argument ("invalid_argument");
    }
    capacidad=cap;
    cant_chars=Strlen(a);
    str=String(a);
}

StringBuffer::~StringBuffer(){
}

int StringBuffer::capacity(){
    return capacidad;
}

int StringBuffer::length(){
    return cant_chars;
}
```

```
String StringBuffer::str_de_buf(){
    return str;
}

String StringBuffer::getStrBuff(){
    return str;
}

int StringBuffer::getCap(){
    return capacidad;
}

int StringBuffer::getCantChars(){
    return cant_chars;
}

void StringBuffer::setCap(int a){
     capacidad=a;
     }
void StringBuffer::setCantChars(int a){
     cant_chars=a;
     }
void StringBuffer::setStrBuff(String s){
     str=s;
     }
```

```
#ifndef __FIXEDLENGTHSTRINGBUFFER_HH__
#define __FIXEDLENGTHSTRINGBUFFER_HH__

#include "String.h"
#include "StringBuffer.h"

class FixedLengthStringBuffer : public StringBuffer {

    public:
     FixedLengthStringBuffer();
     FixedLengthStringBuffer(int);
     FixedLengthStringBuffer(String,int);
     FixedLengthStringBuffer(char*,int);
     ~FixedLengthStringBuffer();
     void insert(int,String);
     void insert(int,char*);
};

#endif
```

FixedLengthStringBuffer.cpp

```
#include "FixedLengthStringBuffer.h"
#include "String.h"
#include "StringBuffer.h"
#include <stdexcept>
#include <iostream>

using namespace std;

FixedLengthStringBuffer :: FixedLengthStringBuffer() :
StringBuffer() {

}

FixedLengthStringBuffer :: FixedLengthStringBuffer(int a) :
StringBuffer(a){

}

FixedLengthStringBuffer :: FixedLengthStringBuffer(String s,int
cap) : StringBuffer(s,cap){

   if (this->getCap() < this->getCantChars()){
       String s0 = this->getStrBuff();
       s0 = s0.Substring(0,cap-1);
       this->setStrBuff(s0);
       this->setCantChars(s0.Length());
   }

}

FixedLengthStringBuffer :: FixedLengthStringBuffer(char* a,int
cap) : StringBuffer(a,cap){

    if (this->getCap() < this->getCantChars()){
       String s0 = this->getStrBuff();
       s0 = s0.Substring(0,cap-1);
       this->setStrBuff(s0);
       this->setCantChars(s0.Length());
   }
}

FixedLengthStringBuffer :: ~FixedLengthStringBuffer(){
   //cout << "Destrui el FixedLengthStringBuffer"<< "\n\n";
}

void FixedLengthStringBuffer :: insert(int pos,String s){

    int cant = this->getCantChars();

    if ((pos<0) || (pos > cant)) {
     throw out_of_range("out_of_range");
    }else{
```

```
    String s0, s1, s2, s3;
    int l0, ls, l2, cap;

    s0 = this->getStrBuff();
    l0 = s0.Length();

    if (pos == 0){
        s0 = s;
        l0 = s0.Length();

    }else if (pos == l0) {
        s0 = s0+ s;
        l0 = s0.Length();

    }else{

        s1 = s0.Substring(0,pos-1);
        l0 = s0.Length();
        s2 = s0.Substring(pos,l0-1);
        ls = s.Length();
        l2 = s2.Length();
        if (ls < l2) {
            s2 = s2.Substring(ls,l2-1);
            s = s + s2;
        }
        s0 = s1 + s;
        l0 = s0.Length();
      }
      cap = this->getCap();
      if (l0 > cap)
        s0 = s0.Substring(0,cap-1);
      this->setStrBuff(s0);
      this->setCantChars(s0.Length());

     }
}

void FixedLengthStringBuffer :: insert(int pos,char* car){

     int cant = this->getCantChars();

     if ((pos<0) || (pos>cant)) {
           throw invalid_argument ("out_of_range");
     } else{

        String s0, s1, s2, s3, s;
        int l0, ls, l2, cap;

     s = String(car);
     s0 = this->getStrBuff();
     l0 = s0.Length();

     if (pos == 0){
```

```
            s0 = s;
            l0 = s0.Length();

        }else if (pos == l0) {
            s0 = s0+ s;
            l0 = s0.Length();

        }else{

            s1 = s0.Substring(0,pos-1);
            l0 = s0.Length();
            s2 = s0.Substring(pos,l0-1);
            ls = s.Length();
            l2 = s2.Length();
            if (ls < l2) {
                s2 = s2.Substring(ls,l2-1);
                s = s + s2;
            }
            s0 = s1 + s;
            l0 = s0.Length();
        }
        cap = this->getCap();
        if (l0 > cap)
            s0 = s0.Substring(0,cap-1);
        this->setStrBuff(s0);
        this->setCantChars(s0.Length());
      }
}
```

```
#ifndef __VARIABLELENGTHSTRINGBUFFER_HH__
#define __VARIABLELENGTHSTRINGBUFFER_HH__

#include "String.h"
#include "StringBuffer.h"

class VariableLengthStringBuffer : public StringBuffer {

    public:
     VariableLengthStringBuffer();
     VariableLengthStringBuffer(String);
     VariableLengthStringBuffer(char*);
     ~VariableLengthStringBuffer();
     void insert(int, String);
     void insert(int, char*);
};


#endif
```

VariableLengthStringBuffer.cpp

```
#include "VariableLengthStringBuffer.h"
#include "String.h"
#include <stdexcept>

using namespace std;


VariableLengthStringBuffer::VariableLengthStringBuffer():StringB
uffer(){

}

VariableLengthStringBuffer::VariableLengthStringBuffer(String
cadena):StringBuffer(cadena,cadena.Length()){

}

VariableLengthStringBuffer::VariableLengthStringBuffer(char*
a):StringBuffer(a,Strlen(a)){

}

VariableLengthStringBuffer::~VariableLengthStringBuffer(){

}

void VariableLengthStringBuffer::insert(int pos, String strg){
    if ((pos<0) || (pos>this->getCantChars()))
    {
     throw out_of_range ("out_of_range");
    }

    if (pos==0)
    {
     if (strg.Length()>=getCantChars()){
           setStrBuff(strg);
           setCantChars(strg.Length());
           setCap(strg.Length());
           }
           else {

                String sub;
                sub =
getStrBuff().Substring(strg.Length(),getCantChars()-1);
                setStrBuff(strg+sub) ;
                setCantChars (getStrBuff().Length());
                setCap( getCantChars());
                }
    }
    else
    {
     if(pos==getCantChars()){
```

```
            String aux=getStrBuff() + strg;
            setStrBuff(aux);
            setCantChars (aux.Length());
            setCap( aux.Length());
      }
      else{
            if((getCantChars()-pos)<=strg.Length()){
                  String sub;
                  sub=getStrBuff().Substring(0,pos-1);
                  String res=sub+strg;
                  setStrBuff(res);
                  setCantChars(res.Length());
                  setCap( res.Length());
            }
            else{

                  String sub;
                  String sub1;
                  sub = getStrBuff().Substring(0,pos-1);
                  sub1=
getStrBuff().Substring(pos+strg.Length(),getCantChars()-1);
                  String s3=sub+strg+sub1;
                  setStrBuff(s3) ;
                  setCantChars(s3.Length());
                  setCap( s3.Length());
            }
      }
    }
}

void VariableLengthStringBuffer::insert(int pos, char* arreglo){
    if ((pos<0) || (pos>getCantChars()))
      throw out_of_range ("out_of_range");

    String s2 = String(arreglo);
    this->insert(pos,s2);
}
```