More on C++ Features

Overloading Operators in C++

What is Overloading Operators?

- Remember the following String class defined in previous lectures:

```
class String {
  public:
    String();
    String(char *s);
          …
          …


          …
  private:
    char * storageM ;
    int length;
};
```

Why Overloading Operators?

```
int main() {
   String s1 (“ABC”);
   String s2 (“XY”);
   String s3;
   s3 = s1 + s2;
   // POINT TWO
   ….
}
```

Not allowed unless that operator + is overloaded for class String.

Operator Overloading

- A class designer can provide a set of operators to work with objects of the class.
- This can be achieved by defining an operator function.
- An operator function need not to be a member function, but it must take at least one class argument. This prevents the programmer from overloading the behavior of operators for built-in data types.
- Only predefined set of C++ operators can be overloaded.

Operator Overloading

- Function Definition: An overloaded function can be defined same as ordinary member or non member functions, except that an “operator” reserved word followed by operator symbol will be used as function’s name.
- An operator function should not change the nature of an operator. For example the overloaded operator function cannot convert a unary operator to a binary or vice versa.

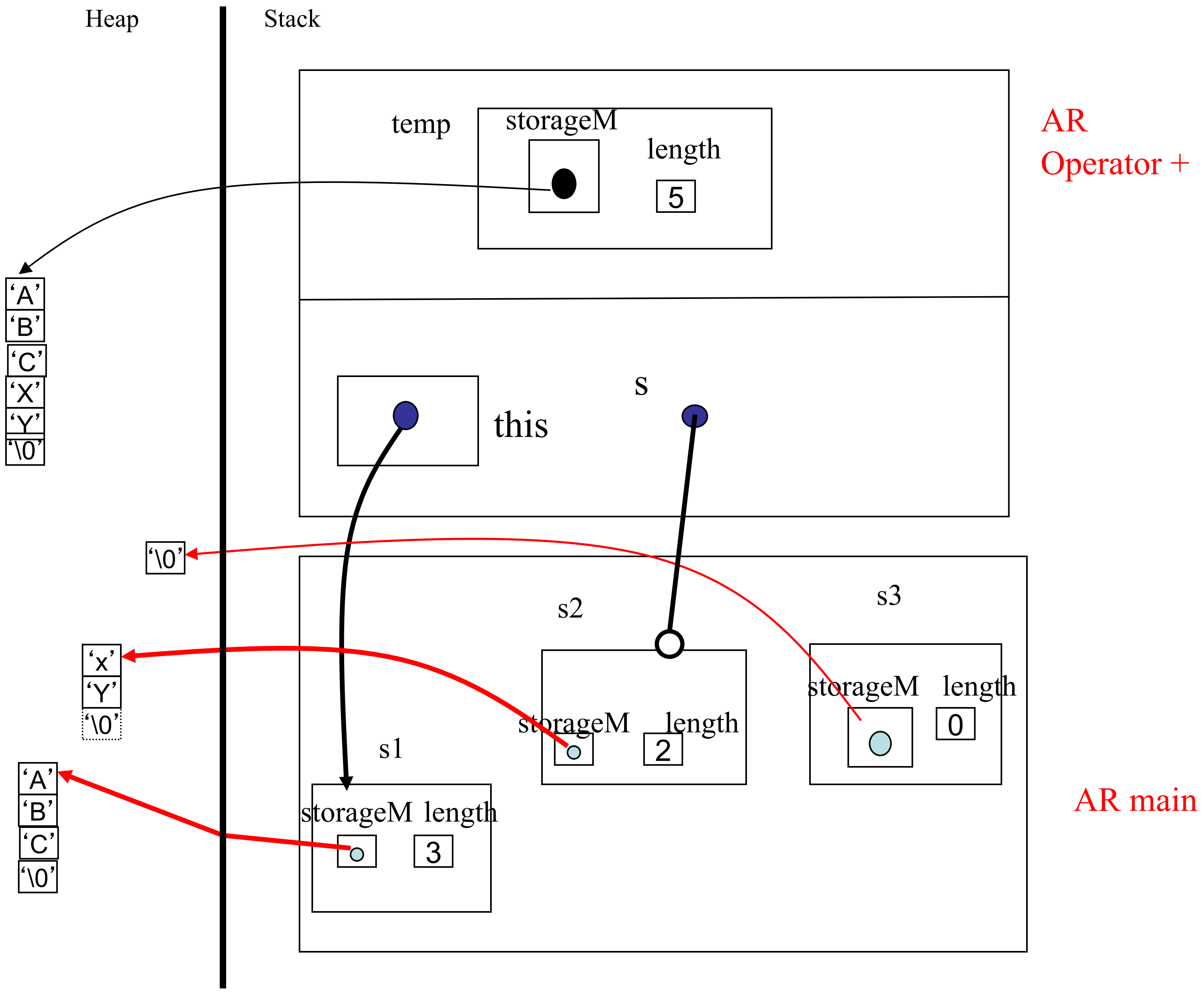
Overloading +

Overloading + Operator for Class String

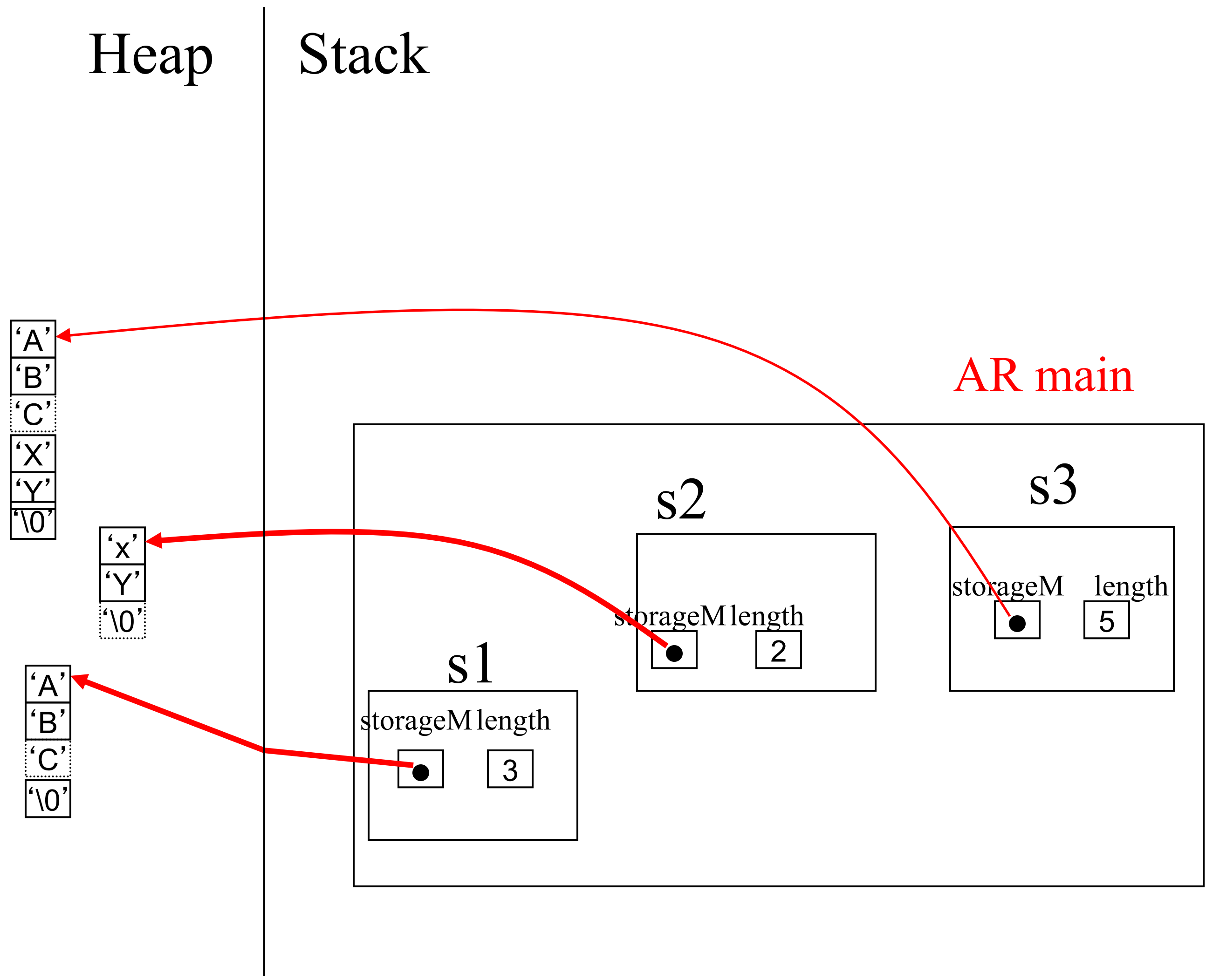
```
class String
{
  public:
      …
      …
       …

     String operator +(const String& s);

  private:
    char * storageM;
    int length;
};
```

```
String String::operator +(const String& s)
{
String  temp;
temp.length = length + s.length;
delete [] temp.storageM;
temp.storageM = new char[temp.length+1];
strcpy(temp.storageM,  storageM);
strcat(temp.storageeM,   s.storageM);
// POINT ONE
return temp;
}
```

AR Diagram for Point ONE



AR Diagram for Point TWO



Overloading +=

Overloading +=

- += operator in the String class can be overloaded to be used for string concatenation (see the function definition in the next slide):
  - String s1 = “ Hello “;
  - String s2 = “World”;
  - s1 += s2;
- += operator is used to concatenate the strings s1 and s2. Therefore s1will change to: “Hello World”.
- **Let’s Write the definition of overloaded operator +=.**

Overloading +=

```
String& String::operator += (const String& s) {
    length += s.length;
    char *p = new char[length+1];
    assert (p !=0);
    strcpy (p , storageM);
    strcat(p , s.storageM);
    delete storageM;
    storageM = p;
    return *this;
}
```

Member or Non-member Overloaded Operators

Member or nonmember?

- If the first parameter of an overloaded function must be an object of another class, the function must be a nonmember.
  - However, if the function needs direct access to the data members, it can be also defined as a friend.
- **Let’s take a look at overloading operator << for class String objects.**

```
ostream&  operator << (ostream& os, String& s)
{
    return os << s.storageM;
}
```

Member or nonmember (Continued)

- The assignment "=", subscript "[]", call "()" and member selection "->" operators are required by language to be defined as class member
- **Let's write the definition of the overloaded operator [] for class String.**

```
char& String::operator [ ] (int elem) {
    assert (elem >= 0 && elem <length);
    return storageM[elem];
}
```

Overloading Increment and Decrement Operators

Overloading ++ and --

- **Now let's try to overload prefix and post fix increment operators ++ and -- for class String.**
- How compiler is supposed to recognize a post-fix from refix?
  - Post-fix uses a dummy argument of type int.

```
class String {
 public:
  String();
  String(char *s);
     ...
 private:
  char* cursor;
  char * storagM ;
  int length;
};
```

```
String::String(const char *s)
 : length((int)strlen(s))
{
 charsM = new char[length + 1];
 strcpy(storageM, s);
 cursor = storageM;
}
```

```
// prefix
 char String::operator ++ ()
 {

    ...
 }

// post-fix
 char String::operator++(int)
 {
  char ret = *cursor;
  cursor++;
  return ret;
 }
```

What is a ‘friend’ in C++

Friend functions and classes

- The following components of a program can be a friends to a class:
  - A global function, visible by a class.
  - A member function of other classes in the program, visible to the class.
  - Another class, visible by a class

Friend Functions

```
void f();

class A{
        int a;
        friend void f();
        public:
        A( );
        void print();
};
```

Friend Classes

```
class B{
    int b;
    friend class A;
    friend class C;
    public:
    B();
    void print();
    void f();
};
```

Example

```
class List;   // Forward declaration of List
class Node{
 private:
     int data;
    Node* next;
    fiend class List; //
 public:
    Node(int a) {data = a;}
    int get_dat() {return data;}
};

class List{
    …         // methods of List can have access
              // to private members of Node
};
```

Friend Member Functions

```
class C {

    friend void B::f();
    int c;
    public:
    C( );
    void print();
};
```