Cairo University
Faculty of Computers and Artificial Intelligence
Computer Science Department

The following sections provide some examples on classes. They explain also how to separate the program into .h and .cpp files:

1 The String class

```
1  // File: astring.h
2
3  #ifndef __STRING_CLASS
4  #define __STRING_CLASS
5
6  #include <iostream>
7  using namespace std;
8
9  class String
10 {
11 private:
12    char* str; // C-string containing the string appended by the null char
13    int n; // Number of characters in str, not including the null char
14
15 public:
16    String(); // Empty constructor
17    String(const char* cstr); // Constructor that takes CString
18    String(const String& s); // Copy constructor
19    ~String(); // Destructor
20
21    String operator + (const String& b) const; // Add String + String
22    String operator + (const char* b) const; // Add String + CString
23    String& operator += (const String& b);
24    String& operator = (const String& b); // Copy assignment
25
26    operator const char*() const; // Type conversion from String to CString
27    char& operator[](int i);
```

```
    // The following functions are friends, not members:
    friend String operator + (const char* a, const String& b);
    friend istream& operator >> (istream& in, String& s);
    friend ostream& operator << (ostream& out, const String& s);
};

String operator + (const char* a, const String& b);
istream& operator >> (istream& in, String& s);
ostream& operator << (ostream& out, const String& s);

#endif
```

```cpp
// File : astring .cpp

#include "astring.h"
#include <cstring>
using namespace std;

String::String()
{
   n=0;
   str=0;
}

String::String(const char* cstr)
{
   n=strlen(cstr);
   str=new char[n+1];
   strcpy(str,cstr);
}

String::String(const String& s)
{
   n=s.n;
   str=new char[n+1];
   strcpy(str, s.str);
}

String::~String()
{
   if(str) delete[] str;
}

String String::operator + (const String& b) const
{
   String r;
   r.n=n+b.n;
   if(r.n==0) return r;
   r.str=new char[r.n+1];
   if(str) strcpy(r.str, str);
   if(b.str) strcpy(r.str+n, b.str);
   return r;
}
```

```
1  String String::operator + (const char* b) const
2  {
3     String r;
4     int nb=strlen(b);
5     r.n=n+nb;
6     if(r.n==0) return r;
7     r.str=new char[r.n+1];
8     if(str) strcpy(r.str, str);
9     if(b) strcpy(r.str+n, b);
10    return r;
11 }
12
13 String& String::operator += (const String& b)
14 {
15    if(b.n==0) return *this;
16    int new_n=n+b.n;
17    char* new_str=new char[new_n+1];
18    if(str) strcpy(new_str, str);
19    if(b.str) strcpy(new_str+n, b.str);
20    n=new_n;
21    if(str) delete[] str;
22    str=new_str;
23    return *this;
24 }
25
26 String& String::operator = (const String& b)
27 {
28    n=b.n;
29    if(str) delete[] str; str=0;
30    if(!b.str) return *this;
31    str=new char[n+1];
32    strcpy(str, b.str);
33    return *this;
34 }
35
36 String::operator const char*() const { return str; }
37
38 char& String::operator[](int i)
39 {
40     if(i<0 || i>=n) throw -1;   // That will be explained in Exceptions lecture
41     return str[i];
42 }
```

```
String operator + (const char* a, const String& b)
{
   String r;
   int na=strlen(a);
   r.n=na+b.n;
   if(r.n==0) return r;
   r.str=new char[r.n+1];
   if(a) strcpy(r.str, a);
   if(b.str) strcpy(r.str+na, b.str);
   return r;
}

istream& operator >> (istream& in, String& s)
{
   char buf[200];
   buf[0]=0;
   in>>buf;
   if(s.str) delete[] s.str;
   s.n=strlen(buf);
   s.str=0;
   if(s.n==0) return in;
   s.str=new char[s.n+1];
   strcpy(s.str, buf);
   return in;
}

ostream& operator << (ostream& out, const String& s)
{
   out<<s.str;
   return out;
}
```

```cpp
// File : main.cpp

#include "astring.h"
#include <iostream>
using namespace std;

int main()
{
   String a, b, c;

   cin>>a>>b;
   cout<<a<<"-"<<b<<endl;

   a="Hello"; b="-World";

   b=a;
   cout<<b<<endl;   // Prints: Hello

   a="Hello"; b="-World";
   const char* pa="hello";
   const char* pb="-world";

   c=a+b;  cout<<c<<endl;   // Prints: Hello-World
   c=pa+b;  cout<<c<<endl;   // Prints: hello-World
   c=a+pb;  cout<<c<<endl;   // Prints: Hello-world
   c=a+=b;  cout<<a<<" "<<c<<endl;   // Prints: Hello-World Hello-World

   a = "Hello";   // Implicit conversion
   b = (String)"World";   // Explicit conversion
   c = static_cast<String>("Prog");   // Explicit conversion
   cout<<a<<" "<<b<<" "<<c<<endl; // Prints: Hello World Prog

   const char* x = a;
   const char* y = (const char*)a;
   const char* z = static_cast<const char*>(a);
   cout<<x<<" "<<y<<" "<<z<<endl; // Prints: Hello Hello Hello

   cout<<a[1]<<endl;   // Prints: e
   a[2]='x';
   cout<<a<<endl;   // Prints: Hexlo

   return 0;
}
```

2 The Fraction class

```
1  // File : fraction .h
2
3  #ifndef __FRACTION_CLASS
4  #define __FRACTION_CLASS
5
6  #include <iostream>
7  using namespace std;
8
9  class Fraction
10 {
11 private:
12    int num;   // numerator;
13    int den;   // denominator;
14
15 public:
16    Fraction(int=0, int=1);   // Constructor with default arguments
17    operator double();   // Type conversion from Fraction to double
18
19    Fraction operator + (const Fraction&) const;
20    Fraction operator += (const Fraction&);
21    Fraction& operator ++ ();   // The prefix ++ operator
22    Fraction operator ++ (int);   // The postfix ++ operator
23
24    bool operator == (const Fraction&) const;   // Test for equality
25
26    friend istream& operator >> (istream&, Fraction&);
27    friend ostream& operator << (ostream&, const Fraction&);
28 };
29
30 istream& operator >> (istream&, Fraction&);
31 ostream& operator << (ostream&, const Fraction&);
32
33 #endif
```

```cpp
// File : fraction .cpp

#include "fraction.h"

Fraction::Fraction(int n, int d)
{
   if(d==0) d=1; // Avoid division by zero
   this->num = n; this->den = d;
}

Fraction::operator double()
{
   return (double) this->num / this->den;
}

Fraction Fraction::operator + (const Fraction& b) const
{
   Fraction c(num * b.den + b.num * den, den * b.den);
   return c;
}

Fraction Fraction::operator += (const Fraction& b)
{
   *this = *this + b;    // Use the overloaded + operator!
   return *this;
}

Fraction& Fraction::operator ++ ()
{
   num += den;
   return *this;
}

Fraction Fraction::operator ++ (int)
{
   Fraction f = *this;
   num += den;
   return f;
}

bool Fraction::operator == (const Fraction& b) const
{
   return (num * b.den == den * b.num);
}
```

```
1  istream& operator >> (istream& in, Fraction& f)
2  {
3     in >> f.num >> f.den;
4     return in;
5  }
6
7  ostream& operator << (ostream& out, const Fraction& f)
8  {
9     out << f.num << "/" << f.den;
10    return out;
11 }
```

```cpp
// File : main.cpp

#include "fraction.h"
#include <iostream>
using namespace std;

int main()
{
   Fraction a, b(2,3), d;  cin>>d;

   Fraction c=Fraction(7,4);   // The RHS constructs a temporary object of type
                               // Fraction using the two arguments constructor, then uses
                               // the copy constructor to copy the temporary object into
                               // the Fraction object 'c' in the LHS. But, since the temporary
                               // object will not be used any more, the compiler usually
                               // optimizes this line to be just the same as: Fraction c(7,4);
                               // to avoid calling the copy constructor

   cout<<a<<" "<<b<<" "<<c<<endl;   // Prints: 0/1 2/3 7/4

   d=b+c; cout<<d<<endl;   // Prints: 29/12

   cout << (double)d << endl;   // Prints: 2.42

   a=Fraction(2,3);   // The RHS constructs a temporary object of type Fraction
                      // using the two arguments constructor, then uses the
                      // assignment operator to copy the temporary object into
                      // the Fraction object 'a' in the LHS

   b=Fraction(3,5);   // Construct a temporary object then assigns it to 'b'

   a+=b; cout<<a<<" "<<b<<endl;   // Prints: 19/15 3/5

   a=Fraction(3,5);

   b=++a; cout<<a<<" "<<b<<endl;   // Prints: 8/5 8/5

   c=a++; cout<<a<<" "<<c<<endl;   // Prints: 13/5 8/5

   if(a==c) cout<<"Equal"<<endl;   // No output

   return 0;
}
```