School of Computing and Informatics

BCS362 – Generic Programming with C++

Strings and String Stream Processing

1 Introduction

The class template **basic_string** provides typical string manipulation operations such as copying, searching etc. The template definition and all support facilities are defined in *namespace std*, these include the typedef statement

```
typedef basic_string<char> string;
```

Listing 1: typedef statement

that creates the alias type **string** for **basic_string**< *char* >

1.1 Initializing a string object

A **string** object can be initialized with a constructor argument as

```
  string text{"Hello UKZN"}; //creates a string from a const char*
```

Listing 2: string initialization using a constructor

which creates a string with characters **Hello UKZN**, or with two constructor arguments as

```
  string data {'B', 10}; //create a string with 10 B characters
```

Listing 3: string initialization using a constructor

which creates a string with 10 **'B'** characters. Class **string** also provide a default constructor, which creates an empty string and a copy constructor. A string can also be initialized in its definition as

```
  string month = "April"; //same as string month{"April"};
```

Listing 4: string initialization

where the = is not an assignment operator, rather it is an implicit call to the string class constructor which does the conversion.

C++ strings are not necessarily null terminated. The C++ class template provides only description of the capabilities of a class string, implementation is platform independent.

1.2 string length

The length of a string can be retrieved using member function **length** or **size**. The subscript operator [ ] which does not perform bound checking, can be used to access and modify string characters. A string object has first subscript of **0** and last subscript of **size() - 1**.

1.3 string input

The stream extraction operator (>>) is overloaded to support string input as

```
string name;
cin >> name;
```

Listing 5: string input

which reads a stream of characters from **cin** until a white-space characters is encounters. Input is delimited with white space character. The **getline()** function is also overloaded for string input and can be used as

```
getline(cin, name);
```

Listing 6: string input

which reads newline delimited string from the keyboard to variable **name**. You can specify an optional delimiter as a third argument of **getline()** function, as shown in class.

1.4 string assignment and concatenation

C++ string class provides a function **assign()** that assigns one string to another as

```
string string1 = "COMP315 is simple";
string string2;
string2.assign(string1); //assign string1 to string2
```

Listing 7: string assignment

The general syntax of **assign()** is

```
targetString.assign(sourceString, start, numberOfCharacters);
```

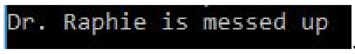
Listing 8: assign() syntax

where **start** is the starting subscript and **numberOfCharacters** is number of characters to be copied from source to target string. When only source string is given as an argument, the whole of it is assigned to target string.

We can concatenate two strings using overloaded + or += concatenation operator, or the function **append()** as

```
string string3 = "Raphie is messed up";
string string4 = "Dr. ";
string4.append(string3);
```

Listing 9: string concatenation using append()

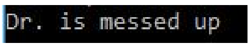
and the output of string4 is 

```
Dr. Raphie is messed up
```

. You can also append just some characters as

```
string string5 = "Dr. ";
string5.append(string3, 7, string3.size() - 7);
```

Listing 10: string concatenation using append()

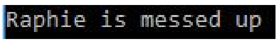
which gives the output 

```
Dr. is messed up
```

You can iterate through characters of a string using range based for loop as

```
for(char i : string3)
  cout << i;
```

Listing 11: looping through a string using range based for

which gives 

```
Raphie is messed up
```

1.5 Comparing strings

The string class provide **compare()** function for comparing strings. This function is overloaded as

```
1 int compare (const string& str) const noexcept;
2 int compare (size_t pos, size_t len, const string& str) const;
3 int compare (size_t pos, size_t len, const string& str, size_t subpos, size_t sublen) const;
```

Listing 12: comparing strings

The function at line 1 compares one string to another as

```
  //comapre string4 and string3
  string4.compare(string3);
```

Listing 13: comparing strings

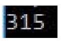
The function at line 2 and 3 can be used compares a sub-string of one string to another string and return 0 if they are equal.

1.6 Substrings

C++ class template string provides the function **substr(start, numberOfCharacters)** which cuts a string (**numberOfCharacters**) at position specified by **start** as

```
  string data1 = "COMP315 is amazing...neh";
  string data2 = data1.substr(4, 3);
```

Listing 14: string sub-string

The output of **data2** would be 

```
315
```

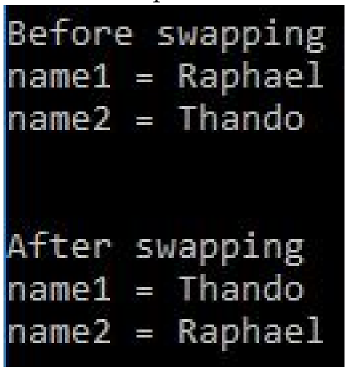
1.7 Swapping strings

You can use the function **swap()** to swap two strings as

```
  string name1 = "Raphael";
  string name2 = "Thando";
  cout << "\n\nBefore swapping\nname1 = " << name1 << endl;
  cout << "name2 = " << name2 << endl;
  name2.swap(name1);
  cout << "\n\nAfter swapping\nname1 = " << name1 << endl;
  cout << "name2 = " << name2 << endl;
```

Listing 15: string swapping

and the output of this code is



```
Before swapping
name1 = Raphael
name2 = Thando


After swapping
name1 = Thando
name2 = Raphael
```

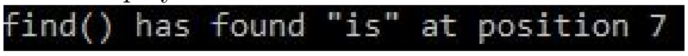
1.8 Finding substrings

You can use the function **find()** which returns the subscript (location) a string (first letter of a sub-string) is in a string as

```cpp
string name3 = "\n\nUKZN is good. KBN is not\n";
cout << "find() has found \"is\" at position " << name3.find("is") << endl;
```

Listing 16: find substrings

which displays



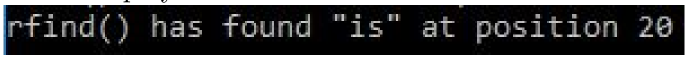
```
find() has found "is" at position 7
```

You can use the function **rfind()** which returns the subscript (location) a string (first letter of a sub-string) is in a string in reverse order as

```cpp
cout << "rfind() has found \"is\" at position " << name3.rfind("is") << endl;
```

Listing 17: find substrings

which displays



```
rfind() has found "is" at position 20
```

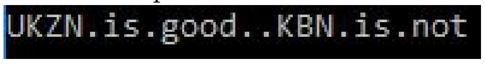
1.9 Replacing characters in a string

You can use the function **replace(exactCharToBeReplaced, numberOfChracters, charReplacing)** as

```cpp
size_t position = name3.find(" "); //find first space
while(position != string::npos)
{
  name3.replace(position, 1, ".");
  position = name3.find(" ", position + 1);
}
cout << name3 << endl;
```

Listing 18: Replacing characters

and the output is



```
UKZN.is.good..KBN.is.not
```

1.10 Insert characters in a string

Use the function **insert(location, string)**

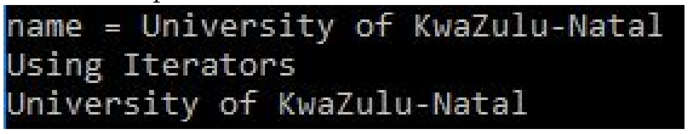
1.11 Iterators

Class string provide iterators for forward and backward traversals of strings. Iterators provide access to individual characters with a syntax similar to pointer operations. Iterators are not range checked.

```cpp
#include <iostream>
#include <string>
using namespace std;
int main()
{
  string name {"University of KwaZulu-Natal"};
  string::const_iterator iterator {name.begin()};
  cout << "name = " << name << endl;
  cout << "Using Iterators \n" ;
  //iterate through the string
  while( iterator  != name.end())
  {
    cout << *iterator; //dereference the  iterator  to get character
     iterator ++; //
  }
  cout << endl;
}
```

Listing 19: accessing characters in a string using iterators

and the output of this code is



```
name = University of KwaZulu-Natal
Using Iterators
University of KwaZulu-Natal
```

1.12 String stream processing

C++ I/O modules also support inputing from, and outputing to, **string**s in memory.These capabilities are referred to as **in-memory I/O** or **string stream processing**.

Input from a string is supported by **istringstream** class. Output to a string is supported by **ostringstream** class. The class names **istringstream** and **ostringstream** are actually aliases defined by the **typedef**s

```
typedef basic_istringstream<char> istringstream;
typedef basic_ostringstream<char> ostringstream;
```

Listing 20: class typedef

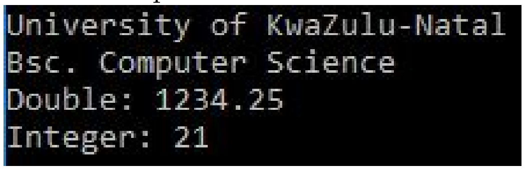
The class template **istringstream** and **ostringstream** provide same functionality as class **istream** and **ostream**, plus other member functions specific to *in-memory formatting*. Programs that use in-memory formatting must include **sstream** and **iostream** libraries.

An **ostringstream** object uses a *string* object to store the output data. Member function **str** returns a copy of that string as

```
  ostringstream output;
  string  name {"University of KwaZulu−Natal"};
  string  prog {"Bsc. Computer Science"};
  double doub {1234.25};
  int  integer  {21};
  //write these  details  to an ostringstream object
  output << name << endl << prog << "\nDouble: " << doub << "\nInteger: " << integer << endl;
  //get a copy of the  string  written to ostringstream object
  string  out = output.str();
  cout << out;
```

Listing 21: In-memory string output

and the output of this code is



```
University of KwaZulu-Natal
Bsc. Computer Science
Double: 1234.25
Integer: 21
```

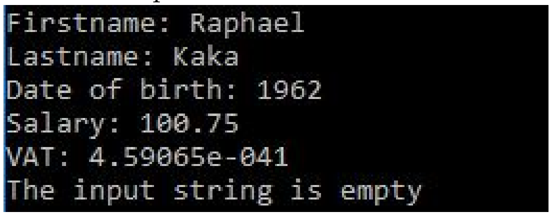
Given a string that contains **Raphael Kaka 1962 100.750** we can read from this string as

```
  string  data = "Raphael Kaka 1962 100.750";
  istringstream  inputString {data};
  string  firstname;
```

```
string lastname;
int year;
double salary;
float vat;
inputString >> firstname >> lastname >> year >> salary >> vat;
cout << "Firstname: " << firstname << "\nLastname: " << lastname << "\nDate of birth: " << year << "\nSalary: "
    << salary << "\nVAT: " << vat << endl;
if (inputString.good())
{
  cout << "The string still has data we can read\n";
}
else
{
  cout << "The input string is empty\n\n";
}
```

Listing 22: In-memory string output

and the output is



```
Firstname: Raphael
Lastname: Kaka
Date of birth: 1962
Salary: 100.75
VAT: 4.59065e-041
The input string is empty
```

1.13 Numeric conversion functions

C++ **to_string()** function from **string** library returns a string representation of its numeric arguments. C++ also provides functions that can convert strings to numeric values

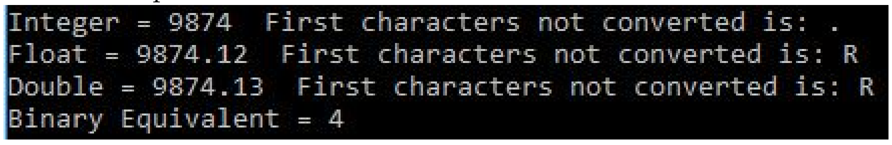
Function	**Comvert to**
stoi()	string to int – receives three arguments. First argument is the string to be converted to **int**, second argument is a pointer to **size_t** variable which stores the first character that was not converted (default is null), and the third argument is the *base* can be between 2 and 36, default is 10.
stol()	string to long.
stoul()	string to unsigned long
stoll()	string to long long
stoull()	string to unsigned long long
stof()	string to float
stod()	string to double
stold()	string to long double

For instance

```
string numString = "9874.125020347656R35246563524243564 7436UKZN";
size_t var;
int numInt = stoi(numString, &var);
cout << "Integer = " << numInt << " First characters not converted is: " << numString[var] << endl;
float numFloat = stof(numString, &var);
cout << "Float = " << numFloat << " First characters not converted is: " << numString[var] << endl;
double numDouble = stod(numString, &var);
cout << "Double = " << numDouble << " First characters not converted is: " << numString[var] << endl;
int binary = stoi("100.01", &var, 2);
cout << "Binary Equivalent = " << binary << endl; //convert binary 100 to decimal. Gives 4
```

Listing 23: String conversion to numericals

and the output of this code is



```
Integer = 9874  First characters not converted is: .
Float = 9874.12  First characters not converted is: R
Double = 9874.13  First characters not converted is: R
Binary Equivalent = 4
```