| hrs | / /20 | | Exams Office Use Only |
|---|---|---|---|

University of the Witwatersrand, Johannesburg

| | |
|---|---|
| Course or topic No(s) | ELEN3009 |
| Course or topic name(s)<br>Paper Number & title | Software Development II |
| Examination/Test* to be held during month(s) of<br>(*delete as applicable) | November 2025 |
| Year of Study<br>(Art & Sciences leave blank) | Third |
| Degrees/Diplomas for which this course is prescribed<br>(BSc (Eng) should indicate which branch) | BSc(Eng)(Elec) |
| Faculty/ies presenting candidates | Engineering and the Built Environment |
| Internal examiners and telephone number(s) | Dr SP Levitt x77209 |
| External examiner(s) | Mr K Padayachee |
| Special materials required<br>(graph/music/drawing paper)<br>maps, diagrams, tables, computer cards, etc) | Computer card for multiple-choice questions |

Time allowance

| Course Nos | ELEN3009 | Hours | 3 |
|---|---|---|---|

Instructions to candidates
(Examiners may wish to use this space to indicate, inter alia, the contribution made by this examination or test towards the year mark, if appropriate)

1. Read instructions on page 1 of exam
2. Answer all 4 questions
3. Available marks: 110 - Full marks: 100
4. Basic scientific calculator permitted

Internal Examiners or Heads of School are requested to sign the declaration overleaf

1. As the Internal Examiner/Head of School, I certify that this question paper is in final form, as approved by the External Examiner, and is ready for reproduction.

2. As the Internal Examiner/Head of School, I certify that this question paper is in final form and is ready for reproduction.

(1. is applicable to formal examinations as approved by an external examiner, while 2. is applicable to formal tests not requiring approval by an external examiner—Delete whichever is not applicable)

Name:_____ Signature:_____

(THIS PAGE NOT FOR REPRODUCTION)

**Instructions**

- Answer *all* questions. The questions do not carry equal weight.
- You may assume that all code snippets given include the required header files and the directive: `using namespace std;`
- Pencil may be used.
- You only need to specify `#include`'s if specifically asked.
- For classes, you can give the implementation entirely in the header file, unless directed otherwise.
- Marks are not awarded solely for functionality but also for good design, making appropriate use of library functions, following good coding practices, and using a modern, idiomatic C++ style.
- Your code must be easily understandable or well commented.
- Reference sheets are provided.

**Question 1** [Total Marks: 40]

For each of the multiple choice questions below, there may be *more than one correct answer*. Mark the *correct answers* on the multiple choice card that is provided. Each question counts for five marks.

A *negative marking scheme* is used so any incorrect answers which are selected for a particular question will lower your mark for that question. Note, however, that you cannot score less than zero for any single question.

1.1 Assume that there are two classes that model distinct concepts but have a fair amount of implementation details in common. Which of the following approaches should be taken to best handle this scenario?

a) Capture the commonality in a separate class and let both classes be derived from this class.

b) Include the commonality in both classes and track changes to ensure consistency.

c) Capture the commonality in a separate class and provide it to the two classes via composition.

d) Make use of the *composite design pattern*.

e) Merge the two classes together into a single combined class.

1.2 Which of the following statements are *true* about exceptions?

a) Throwing an exception has a greater performance cost than returning an error code.

b) C++ does not allow you to nest `try catch` blocks.

c) The `new` operator can throw an `out_of_memory` exception.

d) An exception should never occur during the execution of a well-designed program.

e) Exception objects are not required for catching exceptions.

1.3 Examine the following C++ program:

```
int main()
{
    auto a = 0;
    for (int i = 2 ; i != 20; ++i)
    { cout << ++a << endl; }

    cout << endl;

    auto b = 0;
    for (int i = 0 ; i <= 17; i++ )
    { cout << (b += 1) << endl; }

    cout << endl;

    auto c = 0;
    for (int i = 0 ; i != 18; i++ )
    { cout << c++ << endl; }

    cout << endl;

    auto d = 0;
    for (int i = 2 ; i != 20; i++)
    { cout << (d = d + 1) << endl; }

    return 0;
}
```

Which of the following statements are *true*?

a) The program will run through each of the `for` loops 18 times.

b) Only the `for` loops with variables `a`, `d` will produce the same output.

c) Only the `for` loops with variables `b`, `c` will produce the same output.

d) Only the `for` loops with variables `a`, `b` and `d` will produce the same output.

e) Only the `for` loops with variables `b`, `c` and `d` will produce the same output.

1.4 Which of the following statements are *true*?

a) A `static` data member of a class is accessible to all objects of that class.

b) A `static` member function can read but not modify non-static data members.

c) A `static` variable and a `const` variable are essentially equivalent.

d) The memory allocated for `static` variables and objects is automatically managed.

e) A `static` member function for a class can be called without any objects of the class existing.

1.5 Which of the following statements, related to *Spartan programming*, are *true*?

a) The `const` keyword can be used to support this programming style.

b) Programming in this style means using fewer variables.

c) The use of C-style arrays rather than STL containers is in keeping with a Spartan programming style.

d) Preferring to use variables allocated on the stack rather than `static` variables conforms to this style.

e) Using range-based for loops instead of ordinary for loops is in keeping with a Spartan programming style.

1.6 Which of the following statements are *correct*?

a) A pointer, to a pointer, to a pointer to an integer is valid.

b) Code containing a vector of references to some type will compile.

c) A pointer to a reference, and reference to a pointer, are both valid.

d) It is possible to create a reference to a reference.

e) A reference is a constant pointer.

1.7 Which of the following statements are *true*?

a) There is never a case where the compiler does not know which function to execute for a given line of code.

b) Static dispatch and early binding go hand-in-hand.

c) A virtual method table is used to support static dispatch in programming languages.

d) Virtual tables or *vtables* are defined for each object of a class.

e) A normal function call is replaced with a pointer to the function's implementation during the build process.

1.8 Given the following class, which of the statements concerning it are *true*?

```
1  class A
2  {
3    public:
4      A(): a_{1}, b_{2}, c_{3} {}
5      A(int aa, int bb, int cc = -1): a_{aa}, b_{bb}, c_{cc} {}
6      A(int aa) { a_ = aa; b_ = 0; c_ = 0; }
7
8    private:
9      int a_ = 2;
10     int b_ = 2;
11     int c_ = 2;
12 };
```

a) The constructor on line 4 is the class's default constructor.

b) The constructor on line 4 makes use of in-class member initialization.

c) The constructor on line 6 first default-initialises the variables a_,b_ and c_ and then assigns them values.

d) The following line of code

```
auto a3 = A{3,4};
```

results in a_ = 3, b_ = 4, and c_ = −1.

e) All the constructors can be called without specifying a value for c.

**Question 2** [Total Marks: 22]

Answer the following questions, given the `Person` class definition below.

```
class Person
{
public:
  Person(const string &name) : name_(name)
  {
    cout << "Creating " << name_ << endl;
  }

  Person(const Person &rhs) : name_(rhs.name_)
  {
    cout << "Copying " << name_ << endl;
  }

  string name() const { return name_; }

  void name(const string &new_name) { name_ = new_name; }

  ~Person()
  {
    cout << "Destructing " << name_ << endl;
  }

private:
  string name_;
};
```

Listing 1: `Person` class

a) Give the *entire* output of the following program. (11 marks)

```
1  int main()
2  {
3      auto people = vector<Person>{};
4      people.reserve(2);
5      cout << people.size() << endl;
6      cout << people.capacity() << endl;
7      people.push_back(Person{"Lerato"});
8      people.push_back(Person{"John"});
9
10     for (const auto &person : people)
11       cout << person.name() << endl;
12
13     return 0;
14 }
```

Listing 2: A `vector` of `Person`

b) Explain what a `shared_ptr` is, when it should be used, and how it works. (4 marks)

c) Give the *entire* output of the following program which now makes use of `shared_ptr`.

(7 marks)

```
1  int main()
2  {
3     auto people = vector<shared_ptr<Person>>{};
4     people.reserve(2);
5     cout << people.size() << endl;
6     cout << people.capacity() << endl;
7     people.push_back(make_shared<Person>("Lerato"));
8     people.push_back(make_shared<Person>("John"));
9
10    for (const auto &person : people)
11       cout << person->name() << endl;
12
13    return 0;
14 }
```

Listing 3: A `vector` of `shared_ptr` to `Person`

**Question 3** [Total Marks: 22]

The following questions relate to the modelling and display of a rudimentary grocery list. *Three* different designs are given in the appendix, each being clearly indicated. In all cases, the `Canvas` class is identical and this class's interface is given below. All three designs produce the same output, that is, there is no difference in the list's appearance.

```
enum Font {Arial, TimesNewRoman, Helvetica};

// Canvas makes use of a particular graphics library/framework
// and system fonts in order to render the list on the screen
// in a window. All the necessary includes to use the library/framework
// will be present in this file.
class Canvas
{
public:
   // pass in the pen thickness to write with as well as the font
   // used to display each list item
   Canvas(float graphics_pen_thickness, Font font);
   void drawListItemWithAdjacentEmptyCheckBox(const string& item) const;
   void drawListItemWithAdjacentTickedCheckBox(const string& item) const;
   //...
};
```

Listing 4: `Canvas` public interface

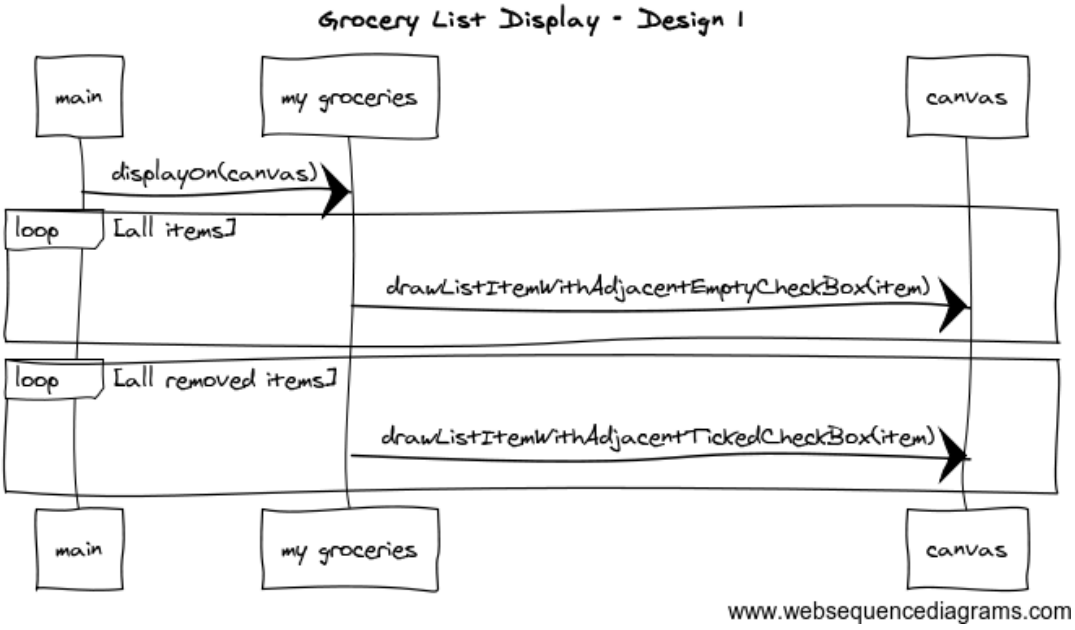


Figure 1: Grocery list display sequence diagram for *Design 1*

a) Figure 1 is a sequence diagram illustrating the object interactions that occur in *Design 1* when an already populated list is displayed on the screen.

Draw *two* sequence diagrams, one for each of the other design alternatives (*Design 2* and *Design 3*), illustrating the object interactions that occur when an *already populated* list is displayed on the screen.

(13 marks)

b) Evaluate all the design alternatives (*Designs 1*, *2* and *3*) in terms of their strengths and weaknesses and justify which option you would choose. In performing your evaluation make reference to relevant design principles that have been covered in the course.

(9 marks)

**Question 4** [Total Marks: 26]

A bank offers multiple account types, each with their own rules for withdrawals, interest rates, and monthly charges.

Specifically, there are these three account types:

1. A small business account that allows a limited overdraft of R 50 000 (allowing a user to have a negative balance of up to R 50 000), has a monthly interest rate of 0.83% on positive balances, and an interest rate of 2% on negative balances. There is a fixed monthly fee of R 100.
2. A savings account that does not allow for an overdraft, has a monthly interest of 0.83%, and charges a fixed monthly fee of R 40.
3. A student account that has no overdraft facility, a monthly interest rate of 0.25%, and applies per-withdrawal fees of R 2 when a withdrawal is made. There is no fixed monthly fee.

Each account must support deposits and withdrawals, withdrawals must be disallowed if the account will be overdrawn for accounts that do not allow for overdrafts, or when the overdraft limit is hit for those that do.

The bank needs to apply month-end processing rules by calculating the interest on the account's balance at month-end and applying it to the account. If the account's balance is positive then the interest is added to the balance. If the account's balance is negative (in the case of business account) then the interest is subtracted from the balance. After this, the monthly bank fees, according to the account type, need to be subtracted from the balance.

Provide an object-oriented solution to the problem, giving *all* the source code for each of the classes that are involved. Bear in mind that the solution must be able to be easily extended to handle new account types. Remember to follow the DRY principle. Use an integral type to represent the account balances in rands. Round values where necessary using `std::round(value)`.

Write your classes so they can be used exactly as in the `main` function shown in Listing 5.

```
int main() {
    try {
        std::vector<std::shared_ptr<Account>> accounts;

        // Create some accounts, each account has an owner and an opening balance
        accounts.push_back(std::make_shared<SmallBusinessAccount>("
            UniversityTextBooksSuppliers CC", 3000));
        accounts.push_back(std::make_shared<SavingsAccount>("Jane Nkosi", 15000));
        accounts.push_back(std::make_shared<StudentAccount>("Mary Roberts", 500));

        // Perform some transactions
        // withdraw() throws if account cannot cover the withdrawal
        // withdraw() and deposit() throw if amount is not positive
        accounts[0]->withdraw(5000);
        accounts[1]->deposit(1000);
        accounts[2]->withdraw(50);

        std::cout << "Balances before month-end:" << endl;
        for (auto& acc : accounts)
            { std::cout << acc->owner() << ": " << acc->balance() << endl; }

        // Apply month-end processing rules to each account
        for (auto& acc : accounts)
            { acc->month_end(); }

        std::cout << "Balances after month-end:" << endl;
        for (auto& acc : accounts)
            { std::cout << acc->owner() << ": " << acc->balance() << endl; }

        return 0;

    } catch (const std::invalid_argument& ex) {
        std::cerr << "Error: you can only withdraw or deposit a positive amount.";
        return 1;
    }
    catch (const std::runtime_error& ex) {
        std::cerr << "Error: account cannot be overdrawn.";
        return 2;
    }
}
```

Listing 5: Using the bank account classes

**Please fill in the question numbers on the front page of your script.**

## Appendix: Grocery List Modelling and Display — Design 1

```
class GroceryList
{
public:
   void add(const string& item);
   void remove(const string& item);
   // ...
   void displayOn(Canvas& canvas);

private:
   List _list;
   List _removed_item_list;
};
```

Listing 6: GroceryList header file

```
void GroceryList::add(const string& item)
{
   _list.push_back(item);
}

void GroceryList::remove(const string& item)
{
   // do nothing if the list item cannot be found
   List::iterator found = find(_list.begin(), _list.end(), item);
   if (found != _list.end())
   {
      _removed_item_list.push_back(*found);
      _list.erase(found);
   }
}

void GroceryList::displayOn(Canvas& canvas)
{
   for (const auto& item : _list)
   {
      canvas.drawListItemWithAdjacentEmptyCheckBox(item);
   }

   for (const auto& removed_item : _removed_item_list)
   {
      canvas.drawListItemWithAdjacentTickedCheckBox(removed_item);
   }
}
```

Listing 7: GroceryList implementation file

```
int main()
{
   GroceryList my_groceries;

   my_groceries.add("Tomatoes");
   my_groceries.add("Lettuce");
   my_groceries.add("Onions");
   my_groceries.remove("Lettuce");

   Canvas canvas(12, Arial);
   my_groceries.displayOn(canvas);
}
```

Listing 8: Main program

## Appendix: Grocery List Modelling and Display — Design 2

```
class GroceryList
{
public:
   void add(const string& item);
   void remove(const string& item);
   List itemsOnList() const;
   List itemsRemovedFromList() const;
   // ...

private:
   List _list;
   List _removed_item_list;
};
```

Listing 9: GroceryList header file

```
void GroceryList::add(const string& item)
{
   _list.push_back(item);
}

void GroceryList::remove(const string& item)
{
   // do nothing if the list item cannot be found
   List::iterator found = find(_list.begin(), _list.end(), item);
   if (found != _list.end())
   {
      _removed_item_list.push_back(*found);
      _list.erase(found);
   }
}

List GroceryList::itemsOnList() const
{
   return _list;
}

List GroceryList::itemsRemovedFromList() const
{
   return _removed_item_list;
}
```

Listing 10: GroceryList implementation file

```
class ListDisplayer
{
public:
   ListDisplayer();
   void display(const GroceryList& list) const;

private:
   Canvas _canvas;
};
```

Listing 11: ListDisplayer header file

```
ListDisplayer::ListDisplayer(): _canvas(12, Arial)
{}

void ListDisplayer::display(const GroceryList& list) const
{
   List items_list = list.itemsOnList();
   for (const auto& item : items_list)
   {
      _canvas.drawListItemWithAdjacentEmptyCheckBox(item);
   }

   List removed_items_list = list.itemsRemovedFromList();
   for (const auto& removed_item : removed_items_list)
   {
      _canvas.drawListItemWithAdjacentTickedCheckBox(removed_item);
   }
}
```

Listing 12: ListDisplayer implementation file

```
int main()
{
   GroceryList my_groceries;

   my_groceries.add("Tomatoes");
   my_groceries.add("Lettuce");
   my_groceries.add("Onions");
   my_groceries.remove("Lettuce");

   ListDisplayer displayer;
   displayer.display(my_groceries);
}
```

Listing 13: Main program

## Appendix: Grocery List Modelling and Display — Design 3

```
class GroceryList
{
public:
   void add(const string& item);
   void remove(const string& item);
   void display(ListDisplayer& displayer) const;
   // ...

private:
   List _list;
   List _removed_item_list;
};
```

Listing 14: GroceryList header file

```
void GroceryList::add(const string& item)
{
   _list.push_back(item);
}

void GroceryList::remove(const string& item)
{
   // do nothing if the list item cannot be found
   List::iterator found = find(_list.begin(), _list.end(), item);
   if (found != _list.end())
   {
      _removed_item_list.push_back(*found);
      _list.erase(found);
   }
}

void GroceryList::display(ListDisplayer& displayer) const
{
   displayer.displayMe(_list, _removed_item_list);
}
```

Listing 15: GroceryList implementation file

```
class ListDisplayer
{
public:
   ListDisplayer();
   void displayMe(const List& list, const List& removed_item_list) const;

private:
   Canvas _canvas;
};
```

Listing 16: ListDisplayer header file

```
ListDisplayer::ListDisplayer(): _canvas(12, Arial)
{}

void ListDisplayer::displayMe(const List& list, const List& removed_item_list)
    const
{
   for (const auto& item : list)
   {
      _canvas.drawListItemWithAdjacentEmptyCheckBox(item);
   }

   for (const auto& removed_item : removed_item_list)
   {
      _canvas.drawListItemWithAdjacentTickedCheckBox(removed_item);
   }
}
```

Listing 17: ListDisplayer implementation file

```
int main()
{
   GroceryList my_groceries;

   my_groceries.add("Tomatoes");
   my_groceries.add("Lettuce");
   my_groceries.add("Onions");
   my_groceries.remove("Lettuce");

   ListDisplayer displayer;
   my_groceries.display(displayer);
}
```

Listing 18: Main program

<string> class

Assume the following declarations:

```
string s, s1, s2;
char c;  char* cs;
string::size_type i, start, len, start1, len1, start2, len2, pos, newSize;
int num;
```

## Methods and operators

*Constructors and destructors*

| | |
|---|---|
| `string s;` | Creates a string variable. |
| `string s(s1);` | Creates s; initial value from s1. |
| `string s(cs);` | Creates s; initial value from cs. |

*Altering*

| | |
|---|---|
| `s1 = s2;` | Assigns *s2* to *s1*. |
| `s1 = cs;` | Assigns C-string *cs* to *s1*. |
| `s1 = c;` | Assigns char *c* to *s1*. |
| `s[i] = c;` | Sets ith character. Subscripts from zero. |
| `s.at(i) = c;` | As subscription, but throws `out_of_range` if *i* isn't in string. |
| `s.append(s2);` | Concatenates s2 on end of s. Same as s += s2; |
| `s.append(cs);` | Concatenates cs on end of s. Same as s += cs; |
| `s.assign(s2, start, len);` | Assigns s2[start..start+len-1] to s. |
| `s.clear();` | Removes all characters from s |
| `s.insert(start,s1);` | Inserts s1 into s starting at position *start*. |
| `s.erase(start,len);` | Deletes a substring from s. The substring starts at position *start* and is *len* characters in length. |

*Access*

| | |
|---|---|
| `cs = s.c_str();` | Returns the equivalent c-string. |
| `s1 = s.substr(start, len);` | s[start..start+len-1]. |
| `c = s[i];` | ith character. Subscripts start at zero. |
| `c = s.at(i);` | As subscription, but throws `out_of_range` if *i* isn't in string. |

*Size*

| | |
|---|---|
| `i = s.length();` | Returns the length of the string. |
| `i = s.size();` | Same as s.length() |
| `i = s.capacity();` | Number of characters s can contain without reallocation. |
| `b = s.empty();` | True if empty, else false. |
| `i = s.resize(newSize, padChar);` | Changes size to *newSize*, padding with *padChar* if necessary. |

*Searching*

*All searches return `string::npos` on failure. The `pos` argument specifies the starting position for the search, which proceeds towards the end of the string (for "first" searches) or towards the beginning of the string (for "last" searches); if `pos` is not specified then the whole string is searched by default.*

| | |
|---|---|
| `i = s.find(c, pos);` | Position of leftmost occurrence of char *c*. |
| `i = s.find(s1, pos);` | Position of leftmost occurrence of *s1*. |
| `i = s.rfind(s1, pos);` | As find, but right to left. |
| `i = s.find_first_of(s1, pos);` | Position of first char in s which is in s1 set of chars. |
| `i = s.find_first_not_of(s1, pos);` | Position of first char of s not in s1 set of chars. |
| `i = s.find_last_of(s1, pos);` | Position of last char of s in s1 set of chars. |
| `i = s.find_last_not_of(s1, pos);` | Position of last char of s not in s1 set of chars. |

*Comparison*

| | |
|---|---|
| `i = s.compare(s1);` | <0 if s<s1, 0 if s==s1, or >0 if s>s1. |
| `i = s.compare(start1, len1, s1, start2, len2);` | Compares s[start1..start1+len1-1] to s1[start2..start2+len2-1]. Returns value as above. |
| `b = s1 == s2` also `> < >= <= !=` | The comparison operators work as expected. |

*Input / Output*

| | |
|---|---|
| `cin >> s;` | >> overloaded for string input. |
| `getline(cin, s);` | Next line (without newline) into s. |
| `cout << s;` | << overloaded for string output. |

*Conversions*

| | |
|---|---|
| `num = stoi(s);` | Converts string s to integer num |
| `s = to_string(num);` | Converts integer num (or any numeric type) to string s |

## Concatenation

The + operator is overloaded to concatentate two strings.

```
s = s1 + s2;
```

Similarly the += operator is overloaded to append strings.

```
s += s2;
s += cs;
s += c;
```

ELECTRICAL AND INFORMATION ENGINEERING

University of the Witwatersrand, Johannesburg

Software Development II

# Reference Sheets

## 1 The `vector` container

**Declarations**

Assume that `T` is some type (eg, `int`).

```
T e;
vector<T> v, v1;
vector<T>::iterator iter, iter2, beg, end;
(use vector<T>::const_iterator or vector<T>::reverse_iterator if appropriate)
int i, n, size;
bool b;
```

**Constructors and destructors**

| | |
|---|---|
| `vector<T> v;` | Creates an empty vector of `T`s. |
| `vector<T> v(n);` | Creates vector of `n` default values. |
| `vector<T> v(n, e);` | Creates vector of `n` copies of `e`. |
| `vector<T> v(beg, end);` | Creates vector with elements copied from range `[beg, end)`. |
| `vector<int> v{1,2,3};` | Creates vector with elements 1, 2 and 3. |
| `v.~vector<T>();` | Destroys all elements and frees memory. |

**Size**

| | |
|---|---|
| `i = v.size();` | Number of elements. |
| `i = v.capacity();` | Max number of elements before reallocation. |
| `i = v.max_size();` | Implementation max number of elements. |
| `b = v.empty();` | True if empty (same as `v.size() == 0`). |
| `v.reserve(size);` | Increases capacity to `size` before reallocation. |

**Altering**

| | |
|---|---|
| `v = v1;` | Assigns `v1` to `v`. |
| `v[i] = e;` | Sets i-th element. Subscripts from zero. |
| `v.at(i) = e;` | As subscript, but may throw `out_of_range`. |
| `v.push_back(e);` | Adds `e` to end of `v`; expands `v` if necessary. |
| `v.pop_back();` | Removes last element of `v`. |
| `v.clear();` | Removes all elements. |
| `v.assign(n, e);` | Replaces existing elements with `n` copies of `e`. |
| `v.assign(beg, end);` | Replaces existing elements with copies from range `[beg, end)`. |
| `iter2 = v.insert(iter, e);` | Inserts a copy of `e` at `iter` position, and returns its position. |
| `v.insert(iter, n, e);` | Inserts `n` copies of `e` starting at `iter` position. |
| `v.insert(iter, beg, end);` | Inserts all elements in range `[beg, end)` starting at `iter` position. |
| `iter2 = v.erase(iter);` | Removes element at `iter` and returns position of next element. |
| `iter2 = v.erase(beg, end);` | Removes range `[beg, end)` and returns position of next element. |

**Access**

| | |
|---|---|
| `e = v[i];` | i-th element. No range checking. |
| `e = v.at(i);` | As subscript, but may throw `out_of_range`. |
| `e = v.front();` | First element. No range checking. |
| `e = v.back();` | Last element. No range checking. |

**Iterators**

| | |
|---|---|
| `beg = v.begin();` | Returns iterator to first element. |
| `end = v.end();` | Returns iterator to *after* last element. |
| `beg = v.rbegin();` | Reverse iterator to first (in reverse order) element. |
| `end = v.rend();` | Reverse iterator to after last (in reverse order) element. |
| `auto diff = v.end() - v.begin();` | Compute position difference/number of elements. |
| `iter++;` | Advance to next element. |
| `iter += 2;` | Advance 2 elements. |
| `iter--;` | Go back to the previous element. |
| `iter -= 3;` | Go back 3 elements. |
| `if (iter < iter2) { };` | Check if `iter` precedes `iter2` |
| `if (iter > iter2) { };` | Check if `iter` succeeds `iter2` |
| `if (iter == iter2) { };` | Check if `iter` equals `iter2` |
| `if (iter != iter2) { };` | Check if `iter` is not equal to `iter2` |

# 2 The doctest Framework

```
TEST_CASE("This is a test case")
{
    CHECK(expression);          // assertion passes if expression is true
    CHECK_FALSE(expression);    // assertion passes if expression is false
    CHECK_THROWS(expression);   // assertion passes if an exception of
        any type is thrown by expression
    CHECK_NOTHROW(expression);  // assertion passes if no exception is
        thrown by expression
    CHECK_THROWS_AS(expression, exception_type); // assertion passes if
        expression throws an exception of exception_type
    CHECK(doctest::Approx(left) == right);  // assertion passes if left
        is approximately equal to right (floating point comparison)
}
```

**Listing 1:** doctest framework: syntax and assertions

# 3 The STL Algorithms

## 3.1 Numerical Algorithms

The following tables summarize commonly used algorithms from <numeric>. The parameter names used in the signatures are explained below. The algorithms given here do not support parallelization.

| Function Parameters | |
|---|---|
| first, last | Iterators denoting a half-open input range [first,last). |
| first1, last1, first2 | Two-range variants: [first1,last1) with a second range starting at first2. |
| result | Iterator to the beginning of the destination/output range. |
| val, init | Initial value or starting value used by the algorithm. |
| binary_op | Binary operation to combine values (see below). |

| Common Binary Operation Function Objects (in <functional>) | |
|---|---|
| std::plus<> | Addition (a + b). |
| std::minus<> | Subtraction (a - b). |
| std::multiplies<> | Multiplication (a * b). |
| std::divides<> | Division (a / b). |
| std::modulus<> | Modulo (a % b). |
| std::logical_and<> | Logical AND (a && b). |
| std::logical_or<> | Logical OR (a  b). |

| Numerical Algorithms | |
|---|---|
| `accumulate(first, last, init)` | Returns the sum of the elements in `[first,last)` added to the value `init`: $\sum a_i + i$ |
| `accumulate(first, last, init, binary_op)` | Accumulates using `binary_op` instead of `+`. |
| `inner_product(first1, last1, first2, init)` | Returns the sum of element-wise products over `[first1,last1)` and the range starting at `first2`, added to `init`: $\sum a_i \cdot b_i + i$ |
| `inner_product(first1, last1, first2, init, binary_op1, binary_op2)` | Uses `binary_op2` for element-wise combine (replaces multiplying the pairs) and `binary_op1` to accumulate (replaces summing the results). |
| `iota(first, last, val)` | Fills the range `[first,last)` with sequentially increasing values starting at `val`, then `val+1`, `val+2`, … |
| `partial_sum(first, last, result)` | Writes the running (prefix) sums of `[first,last)` to `result`: for inputs $\{a_0, a_1, \ldots\}$, outputs $\{a_0, a_0{+}a_1, a_0{+}a_1{+}a_2, \ldots\}$. |
| `partial_sum(first, last, result, binary_op)` | Uses `binary_op` instead of `+` to form prefix results. |
| `adjacent_difference(first, last, result)` | Writes the first element unchanged, then the pairwise differences: $\{a_0, a_1{-}a_0, a_2{-}a_1, \ldots\}$. |
| `adjacent_difference(first, last, result, binary_op)` | Uses `binary_op` to combine adjacent elements instead of subtraction. |

**Example:** Using a standard function object with `accumulate`.

```
#include <numeric>
#include <functional>
#include <vector>

std::vector<int> v{2, 3, 4};
// Product of elements using std::multiplies
auto product = std::accumulate(v.begin(), v.end(), 1, std::multiplies<>{});
// product is 24
```

## 3.2 General Algorithms

The following tables provide information on some of the algorithms which are available in <algorithm>. Parameters which are used in the function signatures are explained below. All of this information has been adapted from: http://www.cplusplus.com/reference/algorithm/.

| Function Parameters | |
|---|---|
| first and last | Represent a pair of iterators which specify a range. The range specified is [first,last), which contains all the elements between first and last, including the element pointed to by first but not the element pointed to by last. |
| result | Represents an iterator pointing to the start of the output range. |
| val, old_value, new_value | Represent elements which are of the same type as those contained in the range. |
| pred | Represents a function which accepts an element in the range as its only argument. The function returns either true or false indicating whether the element fulfills the condition that is checked. The function shall not modify its argument. pred can either be a function pointer or a function object. |

| Non-Modifying Sequence Operations | |
|---|---|
| `all_of(first, last, pred)` | Returns `true` if `pred` returns `true` for all the elements in the specified range or if the range is empty, and `false` otherwise. |
| `any_of(first, last, pred)` | Returns `true` if `pred` returns `true` for any of the elements in the specified range, and `false` otherwise. |
| `none_of(first, last, pred)` | Returns `true` if `pred` returns false for all the elements in the specified range or if the range is empty, and `false` otherwise. |
| `for_each(first, last, fn)` | Applies function `fn` to each of the elements in the specified range. `fn` accepts an element in the range as its argument. Its return value, if any, is ignored. `fn` can either be a function pointer or a function object. |
| `find(first, last, val)` | Returns an iterator to the first element in the specified range that compares equal to `val`. If no such element is found, the function returns `last`. The function uses `operator==` to compare the individual elements to `val`. |
| `find_if(first, last, pred)` | Returns an iterator to the first element in the specified range for which `pred` returns `true`. If no such element is found, the function returns `last`. |
| `find_first_of(first1, last1, first2, last2)` | Returns an iterator to the first element in the range `[first1,last1)` that matches any of the elements in `[first2,last2)`. If no such element is found, the function returns `last1`. The elements in `[first1,last1)` are sequentially compared to each of the values in `[first2,last2)` using `operator==`. |
| `count(first, last, val)` | Returns the number of elements in the specified range that compare equal to `val`. The function uses `operator==` to compare the individual elements to `val`. |
| `equal(first1, last1, first2)` | Compares the elements in the range `[first1,last1)` with those in the range beginning at `first2`, and returns `true` if all of the elements in both ranges match, and `false` otherwise. The elements are compared using `operator==`. |
| `search_n(first, last, count, val)` | Searches the specified range for a sequence of successive `count` elements, each comparing equal to `val`. The function returns an iterator to the first of such elements, or `last` if no such sequence is found. |
| `binary_search(first, last, val)` | Returns `true` if any element in the specified range is equivalent to `val`, and `false` otherwise. The elements are compared using `operator<`. Two elements, `a` and `b` are considered equivalent if `(!(a<b) && !(b<a))`. The elements in the range *shall already be sorted* according to this same criterion (operator<). The function optimizes the number of comparisons performed by comparing non-consecutive elements of the sorted range, which is especially efficient for random-access iterators. |
| `min_element(first, last)` | Returns an iterator pointing to the element with the smallest value in the specified range. The comparisons are performed using `operator<`. An element is the smallest if no other element compares less than it. If more than one element fulfills this condition, the iterator returned points to the first of such elements. |
| `max_element(first, last)` | Returns an iterator pointing to the element with the largest value in the specified range. The comparisons are performed using `operator<`. An element is the largest if no other element does not compare less than it. If more than one element fulfills this condition, the iterator returned points to the first of such elements. |

| Modifying Sequence Operations | |
|---|---|
| `copy(first, last, result)` | Copies the elements in the range [`first`,`last`) into the range beginning at `result`. The function returns an iterator to the end of the destination range (which points to the element following the `last` element copied). The ranges shall not overlap in such a way that `result` points to an element in the range [`first`,`last`). |
| `transform(first, last, result, op)` | Applies the function `op` to each of the elements in the specified range and stores the value returned by `op` in the range that begins at `result`. `op` can either be a function pointer or a function object. The `transform` function allows for the destination range to be the same as the input range to make transformations *in place*. `transform` returns an iterator pointing to the element that follows the `last` element written in the `result` sequence. |
| `replace(first, last, old_value, new_value)` | Assigns `new_value` to all the elements in the specified range that compare equal to `old_value`. The function uses `operator==` to compare the individual elements to `old_value`. No value is returned. |
| `replace_if(first, last, pred, new_value)` | Assigns `new_value` to all the elements in the specified range for which `pred` returns `true`. No value is returned. |
| `fill(first, last, val)` | Assigns `val` to all the elements in the specified range. No value is returned. |
| `remove(first, last, val)` | Transforms the specified range into a range with all the elements that compare equal to `val` removed, and returns an iterator to the new end of that range. The function does not alter the size of the container containing the range of elements. The removal is done by replacing the elements that compare equal to `val` by the next element that does not, and signalling the new size of the shortened range by returning an iterator to the element that should be considered its new past-the-end element. The relative order of the elements not removed is preserved, while the elements between the returned iterator and `last` are left in a valid but unspecified state. The function uses `operator==` to compare the individual elements to `val`. |
| `remove_if(first, last, pred)` | Transforms the specified range into a range with all the elements for which `pred` returns `true` removed, and returns an iterator to the new end of that range. The function does not alter the size of the container containing the range of elements. The removal is done by replacing the elements for which `pred` returns `true` by the next element that does not, and signalling the new size of the shortened range by returning an iterator to the element that should be considered its new past-the-end element. The relative order of the elements not removed is preserved, while the elements between the returned iterator and `last` are left in a valid but unspecified state. |

| Modifying Sequence Operations | |
|---|---|
| `unique(first, last)` | Removes all but the first element from every consecutive group of equivalent elements in the specified range. The function does not alter the size of the container containing the range of elements. The removal is done by replacing the duplicate elements by the next element that is not a duplicate, and signalling the new size of the shortened range by returning an iterator to the element that should be considered its new past-the-end element. The relative order of the elements not removed is preserved, while the elements between the returned iterator and `last` are left in a valid but unspecified state. The function uses `operator==` to compare the pairs of elements. |
| `reverse(first, last)` | Reverses the order of the elements in the specified range. There is no return value. |
| `sort(first, last)` | Sorts the elements in the specified range into ascending order. The elements are compared using `operator<`. There is no return value. |

# <locale> Members

The <locale> header file includes functions for character classification. These are listed below.

| | |
|---|---|
| `bool isalnum(char c)` | Returns true if the character tested is alphanumeric; false if it is not. |
| `bool isalpha(char c)` | Returns true if the character tested is alphabetic; false if it is not. |
| `bool iscntrl(char c)` | Returns true if the character tested is a control character; false if it is not. |
| `bool isdigit(char c)` | Returns true if the character tested is a numeric; false if it is not. |
| `bool isgraph(char c)` | Returns true if the character tested is alphanumeric or a punctuation character; false if it is not. |
| `bool isupper(char c)` | Returns true if the character tested is uppercase; false if it is not. |
| `bool islower(char c)` | Returns true if the character tested is lowercase; false if it is not. |
| `bool isprint(char c)` | Returns true if the character tested is a printable; false if it is not. |
| `bool ispunct(char c)` | Returns true if the character tested is a punctuation character; false if it is not. |
| `bool isspace(char c)` | Returns true if the character tested is a whitespace; false if it is not. |
| `bool isxdigit(char c)` | Returns true if the character tested is a character used to represent a hexadecimal number; false if it is not. |
| `char tolower(char c)` | Returns the character converted to lower case. |
| `char toupper(char c)` | Returns the character converted to upper case. |