C++

C++ is C language with addition of

- classes (for object-oriented programming)
- templates (having types as parameters)

Differences in basic syntax

- `#include <filename>` instead of
  `#include <filename.h>`
- Have new set of libraries and include-files, used by default
- include-files for C may be used by prefixing their names with "c"
- `cin` and `cout` replace `scanf` and `printf`
  Syntax:
  `cin >> <variable name>` where `variable name` may be a variable of type `int`, `char` or `string`
  `cout << <variable-or-string 1> << <variable-or-string 2 > ...`
  where `variable-or-string` is either a string constant, which is printed directly, or a variable of type `int`, `char` or `string`, in which case its current value is printed.

`cin` and `cout` may be extended to cover user-defined datatypes

Namespaces

C++ has a mechanism for limiting scope of identifiers (names for variables, objects and classes)

A namespace is a collection of declarations of such identifiers

If `ns` is a namespace and `ident` is an identifier, you refer to `ident` in the program by `ns.ident`

Example: if integer `count` is part of namespace `ns`, would write
`ns.count++`

In addition have command `using namespace <ns>`, which makes all identifiers of namespace `ns` available in the program

Above example could also be written as

```
using namespace ns;
count++;
```

I/O-library defines namespace `std` for I/O-operations, which should be used by all programs (with `using namespace std`)

Classes

C++ has classes similar to Java

Class definition has two parts:

- Listing types of fields and member functions
- Definition of member functions

keywords `public, private` and `protected` have same meaning as in Java

Operations `new` and `delete` create and delete objects of classes

`new` creates new object, calls constructor function and returns pointer to object

`delete` calls destructor function and deallocates object afterwards

no automatic garbage collection ⇒ programmer must call `delete` (or `free`) to free memory

Abstract classes

C++ implements class hierarchy in two ways:

- Subclasses may override member functions
- Have `abstract classes`: class may not provide implementation of all member functions via keyword `virtual`

Have multiple inheritance:
one class may implement member functions for several abstract classes

Exceptions

work in same way as in Java
raised by several library functions in C++, eg `new` and `delete`
⇒ important to catch them.

Templates in C++

Operations on data types like lists and stacks often independent of type of item stored in lists and stacks (eg `append, reverse, push, pop`)
Implemented by using classes as parameters
Need two additional syntactic constructions:

- class templates: used in class definitions whenever operation works for object of any class
  syntactically defined as `template <class T>`
- instantiation: for objects of class defined using templates, need to provide a concrete class for each class template.
  syntactically defined as `templateClass<concreteClass>`

Containers

Library based around concept of container
container is a template class designed to store objects
Examples:

- Vectors: one-dimensional array with dynamic extensions
- Lists: doubly-linked lists
- deque: double-ended queues;
- queue: queues;
- stack: stacks;
- map: associative arrays;
- set: sets;
- bitset: set of booleans.

Operations on containers

called Algorithms: manipulate objects in containers.
Have several standard algorithms for

- searching
- finding and replacing objects
- traversing objects in a sequence

Have additional algorithms for sets, like union and intersection.

Iterators

Iterators are objects pointing to object in a container together with capability to iterate through the objects in the container
Iterators are typically arguments for algorithms
There are the following classes of iterators:

- Input Iterators: permit single pass through container for reading data;
- Output Iterators: permit single pass through container for assigning values;
- Forward Iterators: permit multiple-pass algorithms for both reading data and assigning values;
- Bidirectional Iterators: as forward iterators, but in addition movement in both directions is allowed;
- Random Access Iterators: allow random access to objects in containers

Each container defines set of available iterators

Examples of iterators for vectors

Vectors provide random access iterators like

- begin: iterator pointing to beginning of vector;
- end: iterator pointing to end of vector;
- []: assignment operator;

Memory allocation in C++

During execution, memory is allocated at the following stages:

- At program start time for global variables
- For each local variable when the block where it is defined is entered
- At the execution of an explicit request like `new`, `malloc`

Memory is freed at the following stages:

- For global variables, when the program exits;
- For a local variable, when the block where it is defined is left;
- when a `free`-or `delete`-command is executed.

Only memory allocated by an explicit request (`malloc` or `new`) may be freed explicitly, by calling `free` or `delete` respectively.
Constructors and destructors of object also called when memory for object is allocated or released via `new` and `delete`, but not when `malloc` or `free` are called.

Object creation and destruction in C++

Objects are created and the constructor functions called at the following stages:

- At program start for globally defined objects
- For each local object when the block where it is defined is entered
- At the execution of a `new`-command

Objects are destroyed and the destructor functions called at the following stages:

- For each local object when the block where it is defined is left
- when a `delete`-command is executed

Destructor function for globally defined object not called at program end

Only objects created with `new` should be destroyed with `delete`