Control Structures

Determines what the program does next. C++ statement syntax is recursive, each item can be used within another statement.

Summary

- `;` Null statement, performs no action
- `{ ... }` Compound statement (block)
- `data declaration`
- `if (condition) statement`
- `if` (condition) statement else statement‘
- `while (condition) statement`
- `do statement while (condition);`
- `for (int; condition; increment) statement`
- `for (type variable : container) statement`
- `switch (value) { statements }`
- `case value: statement`
- `default: statement`
- `label: statement`
- `goto label;`
- `continue;`
- `break;`
- `return;`
- `return expression;`

Exception handling

Provides a mechanism for dealing with runtime errors and other special events. Most exceptions are runtime errors, arithmetic / overflow. Exceptions handle these exceptions at runtime, respoinding to them immdeiatley.

All exceptions are derived from the base class `std::exception`. The `what()` method can used to retrieve a C-string describing the excption.

Logic-Error Exceptions

- `std::logic_error`: erros in library / operator functions not caught by the complier
- `std::runtime_error`: common runtime errors
- `std::bad_cast`: reports invalid use of `dynamic_cast` expression
- `std::bad_typeid`: reports the use of the `type_id` operator on an object that has a `void` type
- `std::domain_error`: voilation of a precondition assumed bya function
- `std::invalid_argument`: invalid argument, normally caught by the complier
- `std::length_error`: attempt to create an object larger than the physical size supported

- `std::out_of_range`: attempt tot use an arguemtn outside the allowable range

Runtime-Error Exceptions

- `std::bad_alloc`: faiure to allocate requested memroy
- `std::overflow_error` arithmetic overflow of a floating point number
- `std::range_error` reports results that fall outside the allowable range
- `std::underflow_error` floating point numbers that are too tiny to be stored in the supported range

Syntax

```
try { statements; }
[ catch (excption_class object) { statements;} ] ...
[ finally { statements } ]
```

Example

```
#include <stdexcept>
#include <iostream>

using namespace std;

...

try {
    // some stuff
} catch (exception e) {
    cout << e.what() << endl;
}
```