Week 1

Exercise 2

../2/conversion.h

```cpp
#ifndef _CONVERSIONT
#define _CONVERSIONT

template <typename outputT, typename inputT>  // Two types
outputT as(inputT inputVar)                   // Return outT, input inT
{
  return static_cast<outputT>(inputVar);      // Cast inT to outT
};

#endif
```

../2/main.ih

```cpp
#define ERR(msg) printf("%s : %d", (msg), __LINE__)

#include "conversion.h"

#include <iostream>     // For testing/printing

using namespace std;
```

../2/main.cc

```cpp
#include "main.ih"

int main(int argc, char const **argv)
{
  int chVal = 'X';
  cout << as<char>(chVal) << '\n';
}
```

Exercise 3

../3/rawCapacity.h

```cpp
#ifndef _RAWCAPACITYT
#define _RAWCAPACITYT

#include <cstddef>                // For size_t

template <typename typeT>         // One var type
typeT* rawCapacity(size_t noVars) // Return pointer to specified type
{
  return new typeT[noVars];       // P to array of noVars var type
};

#endif
```

../3/main.ih

```cpp
#define ERR(msg) printf("%s : %d", (msg), __LINE__)

#include "rawCapacity.h"

#include <string>   // For the example used
#include <iostream> // For printing (testing)

using namespace std;
```

../3/main.cc

```cpp
#include "main.ih"

int main(int argc, char const **argv)
{
  string *pStringArray = rawCapacity<string>(10); // Initialise 10 strings
  pStringArray[1] = "hello";                      // Place "hello" at [1]
  cout << pStringArray[1];                        // Print it (for checking)
  delete[] pStringArray;                          // Free memory
}
```

Exercise 4

Note: I have not yet figured out how to make the template use the correct overloaded function (or even compile with an overloaded function in place). One solution would be to create a namespace for them, but that does not seem like the intended method.

../4/forwarder.h

```
#ifndef _FORWARDER
#define _FORWARDER

template <typename funT, typename ...anyT>        // Function and par package
void forwarder(funT inputFun, anyT&& ...anyVars)  // Needs forwarding
{
  inputFun(anyVars...);
};

#endif
```

../4/main.ih

```
#define ERR(msg) printf("%s : %d", (msg), __LINE__)

using namespace std;

#include "forwarder.h"

#include <iostream>

void fun(int first, int second);
void incrementer(int &one, int &two, int &three);
```

../4/main.cc

```
#include "main.ih"

int main(int argc, char const **argv)
{
  forwarder(fun, 1, 3);             // Calls fun() correctly
  int x = 0;
  forwarder(incrementer, x, x, x);
  cout << x << '\n';                // Prints '3'
}
```

../4/fun.cc

```
#include "main.ih"

void fun(int first, int second)
{
  cout << "fun(" << first << ", " << second << ")\n";
}
// Just an example function
```

../4/incrementer.cc

```
void incrementer(int &one, int &two, int &three)
{
  ++one;
  ++two;
  ++three;
}
// Also just an example function
```

Exercise 5

../5/operator/operator.h

```
#ifndef INCLUDED_OPERATOR_
#define INCLUDED_OPERATOR_

#include <string>

class Operator: public string
{
  public:
    Operator() = default;
};

#endif
// Just a barebones class that is basically a string
```

../5/operator/operator.ih

```
#include "operator.h"

using namespace std;
```

../5/smooth.h

```
#ifndef INCLUDED_SMOOTHT_
#define INCLUDED_SMOOTHT_

template<typename rT>
Operator operator+(Operator const &leftSide, rT const &rightSide)
{
  Operator smoothOp(leftSide);  // Returns a new variable
  smoothOp += rightSide;        // constructed from left and right
  return smoothOp;              // Left has to be Operator, right any
}

#endif
```

../5/main.ih

```
#define ERR(msg) printf("%s : %d", (msg), __LINE__)

using namespace std;

#include <iostream>

#include "operator/operator.h"
#include "smooth.h"
```

../5/main.cc

```
#include "main.ih"

int main(int argc, char const **argv)
{
  Operator one{ "yes, " };              // Added this to test whether
  Operator two;                         // joining actually works

  Operator three{ one + two };
  Operator four{ one + 42 };
  Operator five{ one + "hello world" };

  cout << five;
}
```

Exercise 6

../6/storage/storage.h

```
#ifndef INCLUDED_STORAGE_
#define INCLUDED_STORAGE_

#include <vector>

class Storage
{
  std::vector<size_t> d_data;

  public:
    Storage() = default;
    Storage(std::initializer_list<size_t> const &list);

    template <typename inputT>
    size_t operator[](inputT const inputVar) const;
};

#include "indexOp.h"  // Where does this go?

#endif
```

../6/storage/storage.ih

```
#include "storage.h"

using namespace std;
```

../6/storage/c_storageInitList.cc

```
#include "storage.ih"

Storage::Storage(std::initializer_list<size_t> const &list)
: d_data(list.begin(), list.end())
{
}
// Just populates d_data using an initialiser list
```

../6/storage/indexOp.h

```
#ifndef INCLUDED_INDEXOPT_
#define INCLUDED_INDEXOPT_

template <typename inputT>
size_t Storage::operator[](inputT const inputVar) const
{
  return d_data.at(static_cast<size_t>(inputVar));
}

#endif
// This is only a 'safe' index operator. Does a non-safe one (i.e. one that
// allows for insertion) also have to be created?
```

../6/main.cc

```
#include "main.ih"

#include <iostream>

int main(int argc, char const **argv)
{
  Storage storage = {1, 2, 3, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14};

  cout << storage[Icmp::ID]          << '\n'
       << storage[TcpUdp::PROTOCOL] << '\n'
       << storage[12]                << '\n'
       //<< storage['a']               << '\n' // Does compile, but out of range
       << storage[12.5]              << '\n';
}
// Note: the vector could be expanded so that 'a' would run (now it throws)
// an out of bounds error, but it seemed a bit excessive for now.
```

../6/main.ih

```
#define ERR(msg) printf("%s : %d", (msg), __LINE__)

#include "storage/storage.h"
#include "enums.h"

using namespace std;
```

../6/enums.h

```
#ifndef INCLUDED_ENUMS_
#define INCLUDED_ENUMS_

enum class TcpUdp
{
  SECONDS      = 1,
  MU_SECONDS,
  PROTOCOL,
  SRC,
  DST,
  SPORT,
  DPORT,
  SENTPACKETS,
  SENTBYTES,
  RECVDPACKETS,
  RECVDBYTES,
  nFields
};

enum class Icmp
{
  SECONDS      = 1,
  MU_SECONDS,
  SRC,
  DST,
  ID,
  SENTPACKETS,
  SENTBYTES,
  RECVDPACKETS,
  RECVDBYTES,
  nFields
};

#endif
```