CONFIG += c++11

1 Slide 3.3

```
#include <iostream>
#include <vector>
using namespace std;

int global;

int main(int argc, char** argv)
{
  int a[1];
  int* b = new int[1];
  const char* c = "hello world";

  std::cout << " argument = " << argv << "stack = " << a << " heap = " << b
            <<" global = " << &global << " constant = " << (void*)c
            << " code = " << (void*)&main << std::endl;

  return 0;
}
```

2 Slide 3.4

```
delete[] b;
delete[] a; // ?
delete[] c; // ?
```

3 Slide 3.6

```
#include <iostream>

int main()
{
  int a[10];
  int* a_ptr = a;
  int* b = new int[10];
  int* c = (int*)malloc(10 * sizeof(int));
}
```

3.1 Initialise the array

```
int main()
{
  int a[10] = {10, -2, 4, 49, 4, -2, 3, 8, 3, 9 };
  for(int i = 0; i <10; ++i)
```

```
  {
    std::cout << "a["<< i << "]=" << a[i] << std::endl;
  }
}
```

[language=C++]

3.2 play with accessing memory content

```
  std::cout << a[0] << "␣" << a[1] << std::endl;
  std::cout << a[-5] << std::endl;
  std::cout << a[-500] << std::endl; // then increase
  a[-5] = 10;
```

3.3 add an element to the middle

```
  for(int i= 9; i > 4; --i)
  {
    a[i] = a[i-1];
  }
  a[4] = 7;
  for(int i = 0; i <10; ++i)
  {
    std::cout << "a["<< i << "]=" << a[i] << std::endl;
  }
```

3.4 do it with a dynamic array

```
int main()
{
  int* a = new int[10] {10, -2, 4, 49, 4, -2, 3, 8, 3, 9 };
  for(int i = 0; i <10; ++i)
  {
    std::cout << "a["<< i << "]=" << a[i] << std::endl;
  }
  {
    int* new_a = new int[11];
    for(int i = 0; i < 4; ++i)
    {
      new_a[i] = a[i];
    }
    for(int i= 10; i > 4; --i)
    {
      new_a[i] = a[i-1];
    }
    new_a[4] = 7;
    delete a;
    a = new_a;
  }
  for(int i = 0; i <10; ++i)
  {
    std::cout << "a["<< i << "]=" << a[i] << std::endl;
```

```
  }

}
```

[language=C++]

3.5 make a function

```
  int* insert(int* arr, int index, int _value, int arr_size)
  {
  ...
  }

  a = insert(a, 4, 7);
```

3.6 make a template function

```
  template<typename _T_>
  _T_* insert(_T_* arr, int index, _T_ _value), int arr_size)
  {
  ...
  }

  a = insert(a, 4, 7);
```

3.7 make a template struct

```
  template<typename _T_>
  struct my_array
  {
    my_array(_T_* _init, int _init_size) : data(_init), arr_size(_init_size) {}
    void insert(int index, _T_ _value)
    {
    ...
    }
    _T_* data;
    int arr_size;
  };
  my_array<int> a (new int[10] {...}, 10);
```

4 Slide 3.11

```
#include <vector>

vector<int> v; //  initialy empty
int         x;

while (cin >> x)
    v.push_back(x); // insert at the end capacity will be increased if needed

for (size_t i = 0; i < v.size(); ++i) // loop with usual indexation
    v[i] = v[i] + 1;
```

```
for (auto it = v.begin(); it != v.end(); ++it) // loop with iterator -
    *it = *it + 1;                             // works like pointer

for (auto x : v) // C++11, range loop, elements cannot be changed
    cout << x << '\n';

for (auto& x: v) // C++11, range loop, elements can be changed
    x = 2 * x + 1;
```

5 Slide 3.12

```
std::vector<int> v = {1,2,3,4}; //  initialy empty
```

Demonstrate

front, back, [], at, pop_back, insert, erase,
clear, empty, size, v1=v2, v1==v2, v1<v2, v1.swap(v2), swap(v1,v2)

For [] vs at, show that [] does not assert while at does.