Exercise I

- Pseudo-code:
  - read-in integer value N from user
  - generate integer array v of size N
  - write and call a function init(…) that initializes v[i] to be i (for i = 0..N-1)
  - generate integer array v2 of size N
  - write and call a function cp(…) that copies content of v into v2
  - output content of v2 to console to test code
- How to proceed?
  - generate a small ex1.cpp file which reads in N and outputs N, compile it, run it
  - go step by step through the pseudo-code: add one new element, compile, run, test
  - good to have: small test.cpp file to quickly test unknown C++ features where needed

Exercise II

- Rewrite code from exercise I using a class
  - move “v” and “N” into class “Container” and define member functions for this code:

```
int main() {
    int N;
    cout << "Enter integer number: ";
    cin >> N;

    Container c(N);  // generate array of size N and set v[i] = i
    Container c2(c); // generate new array and copy array from c
    c2.display();    // output content of array to command line
    c2.writefile();  // output content of array to file "out.txt"

    return 0;
    // make sure arrays in c and c2 are destroyed in destructor!
}
```

goal: same behavior & output as exercise I

new

  - as for Ex1: proceed step-by-step…
  - new function “writefile”: write content of “v” into file “out.txt”
    - follow tutorial: http://www.cplusplus.com/doc/tutorial/files/ (test it in test.cpp)
      - hint: any command with “cout” will also work for a variable of type ofstream

Exercise III

- Understand inheritance with a test code
  - write a class A and a class B which inherits from A and a class C inheriting from B
  - define a function f in A, B and C with different console outputs
  - use the following main function to explore all possible scenarios:

```
int main() {
    // define instances for 3 classes
    A a;
    B b;
    C c;
    // use base pointers to A and B for testing
    A* pa = &a; pa->f();
    pa = &b;    pa->f();
    pa = &c;    pa->f();
    B* pb = &b; pb->f();
    pb = &c;    pb->f();

    return 0;
}
```

  - add the “virtual” keyword to f of class B and predict the outcome
  - repeat with making only f of class A virtual or only f of class C or setting all to virtual

Exercise IV

- Apply inheritance to choose particles at runtime
  - write base class BaseParticle for a 2D particle with position (double) x, y and velocities vx, vy. Member functions:
    - init(): position particle at x = 0, y = 1
    - setvelocity(): pure virtual function
    - move(double dt): update position as x = x + dt*vx and y = y + dt*vy
    - display(): write x and y to console output
  - define 2 derived classes of BaseParticle with (vx,vy) = (-y,x) and (vx,vy) = (x,y) respectively. Define BaseParticle pointer “p” at runtime and make this code work:

```
// do N = 10 steps with timestep dt = 0.1
p->init();
for (int i = 0; i < 10; ++i) {
    p->setvelocity();
    p->move(0.1);
}
p->display();
```

  - write a function “run(int N, double dt)” in BaseParticle to replace code above with:

```
p->run(10, 0.1);
p->display();
```

Exercise V

old code (from solution of exercise 2)

```
Container c(N);       // generate array
Container c2(c);      // copy array
c2.display();         // output content
```

new code

```
Container<int> c(N);
Container<int> c2(c);
c2.display();
// same with double
Container<double> cd(N);
Container<double> cd2(cd);
cd2.display();
// convert
convert(c,cd);
cd.display();
convert(cd2,c2);
c2.display();
```

- Templatize a given code
  - download Ex5-container.cpp from the course webpage
  - the main function assumes you can write Container<T> for any type T
  - make the code compile and work, by changing the Container class accordingly
  - add a template convert function, such that the convert-calls work (uncomment them)
    - convert(src,trg) overwrites elements of trg with elements of src (even if they have different data types) (only needs to work if they both have same size)
- Feel free to practice templates first in a separate test code

Exercise VI

- Learn use of vector and algorithms
  - fill a vector v with values {1,2,3,1,2,3,3,4,5,4,5,6,7}
  - write a function “display(vector<int>& v)” which displays content of v using “for_each”
  - find all unique integers in v and fill a vector vu with those unique numbers
  - generate an integer vector vc of the same size as vu
  - for each vu[i], count how often it occurs in v and store the result in vc[i]
  - use display to show vu and vc. Expected result:
    vu = {1,2,3,4,5,6,7}, vc = {2,2,3,2,2,1,1}
- Reminders:
  - explore http://en.cppreference.com/w/cpp/algorithm to find suited functions
  - if you use examples with C++-11 features, compile with “g++ -std=c++11 file.cpp”