Introduction to Systems Programming
234214
ex1-dry

Ori Zohar – 205960750

Guy Kehat – 208520510

1. **Slice:**

```
class BadInput : public std::exception{};
template <class T>
std::vector<T> slice(std::vector<T> vec, int start, int step, int stop) {
    //These three ifs have been seperated for readability.
    if(start<0 || start>= vec.size() ) {
        throw BadInput();
    }
    if(stop<0 || stop > vec.size() ) {
        throw BadInput();
    }
    if(step<=0) {
        throw BadInput();
    }
    std::vector<T> new_vec;
    //If start is bigger than stop we will return an empty vector.
    if(start>=stop)
    {
        return new_vec;
    }
    //If we have reached here the input is valid, and we can safely insert
the relevant objects.
    for(int i=start ; i<stop ; i+=step)
    {
        new_vec.push_back(vec[i]);
    }
    return new_vec;
}
```

2. **A new version for class A:**

```
class A {
public:
    std::vector<std::shared_ptr<int>> values;
    void add(int x) {
        int* x_ptr = new int(x);
        std::shared_ptr<int> x_shared_ptr(x_ptr);
        values.push_back(x_shared_ptr);
    }
};
```

The reason for the memory leak was that in function add there were dynamically Allocated memory that was not freed. So a way of fixing it is by using shared pointers, Those will be responsible for freeing the memory when the function ends.

***Please note you can copy the text from those images.**