# C++ Advanced – Exam 2 (2 Dec 2018)

Write C++ code for solving the tasks on the following pages.

Code should compile under the C++03 or the C++11 standard.

Submit your solutions here: https://judge.softuni.bg/Contests/1334/CPlusPlus-Advanced-Exam-2-2-Dec-2018

Any code files that are part of the task are provided under the folder **Skeleton**.

Please follow the exact instructions on uploading the solutions for each task.

# Task 3 – UnitPtr

This task is a variant of **Task 3 – Group** in **C++ Advanced – Exam 1 (18 Nov 2018)**, **however**, instead of using raw pointers, the **skeleton code uses a UnitPtr** class which **automatically manages memory by deleting memory when no UnitPtr points to it**. As a skeleton you are given the solution to that task, but with the usage of the **UnitPtr** class, instead of raw **Unit\***.

That means **UnitPtr** implements shared ownership of a resource – once a **UnitPtr** is created for a block of memory, indicated by a **pointer**, it keeps track of that **UnitPtr** and all other **UnitPtr** objects that point to the same memory (**UnitPtr** copies and **UnitPtr** objects initialized with the same **pointer**), and when all of them are deallocated, it **delete**s the **pointer**.

You are provided with the declarations for **UnitPtr** in the **UnitPtr.h** file.

Your task is to create a **.cpp** file and implement the **UnitPtr** in it, in such a way that the code for the task works correctly and there are **no memory leaks or undefined behavior**.

NOTE: The provided code for **Task 3 – Group** contains the solution for that task – you do NOT need to worry about solving that task, just focus on the **UnitPtr** class – if you implement it so that it correctly accesses and manages memory (as described above), the rest of the code will work correctly.

### Hints

The tests in the Judge system are the same as those for **Task 3 – Group**, you can use them to test out this task.

**Below is the Description for Task 3 – Group.**

**You do NOT need to solve that task (the skeleton contains the solution for it), but it will help you understand what the code does and under what restrictions it operates.**

*In a certain strategy game, players can control groups of up to 12 units (each unit has an id). To ease control, players can create* ***numbered groups*** *(from 0 to 9 inclusive) of units, called* ***control groups****. Groups are created by first* ***selecting units****, and then* ***adding the selected units to a group*** *(Note: the selection itself is a group). This can be done in* ***two ways*** *– either a numbered group is* ***set to the value of the current selection****, or the current selection is* ***appended to a numbered control group*** *(i.e. in the way does not reset units already in the group).*

*If a group already has 12 units and a new unit is added, the* ***oldest unit added to the group is replaced with the newest unit****. For this program, when* ***printing a group, the units must be sorted in increasing order*** *of their id.*

*Adding a Unit to a Group that already contains a Unit with that id is a no-op; - i.e. it should have no effect on the Group.*

*You are given the code of a program that tests executing a series of such grouping operations multiple times, to test how well the grouping code handles high amounts of input from the player. A test can be executed anywhere between 1 and 1000 times (yep, those Korean pro-gamers are fast).*

*Operations are one of the following:*

* *unit – creates a Unit with an id, that can be selected and added to groups*
* *select – resets the* ***current selection*** *Group to contain a single Unit.* *Units already contained in the group are ignored (they remain in the target group in their original position)*
* *shift-select – adds one or more units to the* ***current selection*** *Group*
* *ctrl**– sets a control Group with the* ***current selection******Group***
* *shift**– appends the* ***current selection*** *Group**to a* ***control*** *Group.* *Units already contained in the group are ignored (they remain in the target group in their original position)*

*Keep in mind the current selection is a Group, so the same rules for adding Units apply – if there are already 12 Units in the current selection, the oldest Unit is replaced with the newest Unit.*

*Also keep in mind that* ***printing*** *a Group on the console requires printing the Units* ***sorted in ascending order of their*** *id, but that does not mean that they must be stored in the group in that order (as that would complicate replacing the “oldest” with the “newest” unit when the max number of units are reached).*

*The provided code handles parsing these operations but uses a Group class to execute them – your task is to study and implement the Group class and implement its print operation according to the rules above. The Group class is declared in the Group.h header, but its implementation is missing.*

*You should submit a single .zip file for this task, containing ONLY the files you created.*

*The Judge system has a copy of the other files and will compile them, along with your file, in the same directory.*

### Restrictions

Each test will be executed between 1 and 500 times. Make sure you handle memory correctly.

About half the tests will be executed only 1 time.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 1  unit a  unit b  unit c  unit d  select a  select b  unit e  ctrl 1  shift-select a c d .  ctrl 2  shift 3  select e  ctrl 4  shift 3  end | 0:  1: b  2: a b c d  3: a b c d e  4: e  5:  6:  7:  8:  9: |
| 1  unit a  unit b  unit c  unit d  select a  select b  unit e  unit f  unit g  unit h  unit i  unit j  unit k  unit l  unit m  ctrl 1  shift-select c .  shift-select a .  shift-select d .  ctrl 2  shift 3  select e  ctrl 4  shift 3  shift 3  shift-select g h i j f k e .  ctrl 8  select a  shift-select c a d m l b .  shift 8  ctrl 9  end | 0:  1: b  2: a b c d  3: a b c d e  4: e  5:  6:  7:  8: a b c d f g h i j k l m  9: a b c d l m |