# C++ Advanced – Exam 1 (18 Nov 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/1315/CPlusPlus-Advanced-Exam-1-18-Nov-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 4 – Terran

*In a certain strategy game, players can make units and give commands to those units, including commands to build other units. Each unit has commands it can execute. Executing some commands happens immediately (e.g. reporting the state of the unit) and other commands take time (e.g. constructing a building or creating a unit from a building). The player can either give a command, or do nothing, and in both cases, time passes, and all units react to time passing.*

You are given code that simulates part of the game by handling the input of commands (and idle time) and sending them to the proper units.

The provided code runs a loop and reads user actions, until the line "end" is entered. On each iteration of the loop, the code sends the actions to the appropriate units, updates all units, and if there are any new units produced by the existing units, it adds them to the list of all units.

There are 3 types of user actions:

* idle – instructs the program to simulate a given amount of time passing. No commands are executed on the units, but they are notified of time passing
* create – creates either a Worker or a Marine (explained later) with an id, and adds them to the list of all units before the next iteration
* command – sends a command to a unit with a specific id. Different units handle different commands in different ways (explained later)

All objects in the game inherit the Unit class, (except the object that handles the create command). A Controllable class is used to access any object that can process a user action (a user action is sent as a vector of strings, each string representing a word in the action). Any object that can create other objects should inherit the Producer class (or the ProducerBase, which inherits Producer).

For this task, there are 4 units – Worker, Marine, Depot and Barracks (the last 2 are actually buildings, but we treat everything as a “unit”), with the Depot class already implemented.

The Marine is the simplest type of unit. It only handles the report command by printing "marine", followed by its id, followed by " reporting". So, if a Marine with an id of 42 is told to report, the string "marine 42 reporting" should be printed.

The Worker unit can produce other units. It handles the following commands:

* build depot – constructs a Depot after 25 units of time, if free, otherwise reports it is busy
* build barracks – constructs a Barracks after 50 units of time, if free, otherwise reports it is busy
* report – prints "worker ", followed by its id, followed by either " busy" or " free"

A Worker cannot change what it is constructing – after a build command it goes into a busy state, which ends when it completes constructing the current unit it was told to construct. For example, if we have the commands:

create worker w

command w build depot

command w build barracks

The worker will start building the depot, but when told on the next time unit to build the barracks, will just say it is busy ("worker w busy"). However, if the commands are:

create worker w

command w build depot

idle 30

command w build barracks

the worker will have time to finish the depot and then will start the barracks, without printing anything.

The Barracks unit has only one command – train marine. This command starts the training of a Marine and one is produced after 15 units of time. The Barracks also has a **queue** with 5 positions – if more train marine commands are given while the Barracks is training a marine, those marines are enqueued. The first enqueued marine will be started immediately on the time unit after the Barracks completes the training of the current marine. If the queue already has 5 items, the train marine command is ignored.

After processing all user actions, the program prints all the units, in the order they were added.

Your task is to study the provided code and implement the Marine, Worker and Barracks class, as well as any other classes/files you decide are necessary for implementing the task.

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

There will be no operations with non-existing ids, no invalid ids and no invalid commands. No unit will receive a command it does not support. There will be no units with the same id.

The tests are setup so that an error of 1 time unit in the time of completion of a construction or training of a marine will not lead to an incorrect result.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| create worker w  create depot  create marine m  command m report  end | marine m reporting  ---  worker w  depot  marine m |
| create worker w  create marine m  command w build depot  idle 10  command m report  command w build barracks b  idle 20  command w report  command w build barracks b  idle 55  command b train marine d  command b train marine e  idle 31  command d report  command e report  command b train marine f  command b train marine g  create worker x  command b train marine h  command b train marine i  command b train marine j  command b train marine k  command b train marine z  idle 500  end | marine m reporting  worker w busy  worker w free  marine d reporting  marine e reporting  ---  worker w  marine m  depot  barracks b  marine d  marine e  worker x  marine f  marine g  marine h  marine i  marine j  marine k |