STARCRAFT

42-WAR

Staff 42 bocal@42.staff.fr

Abstract: This document is the subject of the first C++ pool exam.

Contents

Chapter I

General rules

- Any function implemented in a header (except in the case of templates), and any unprotected header, means 0 to the exercise.

- Every output goes to the standard output, and will be ended by a newline, unless specified otherwise.

- The imposed filenames must be followed to the letter, as well as class names, function names and method names.

- Remember: You are coding in C++ now, not in C anymore. Therefore:

  ○ The following functions are FORBIDDEN, and their use will be punished by a -42, no questions asked: *alloc, *printf and free.

  ○ You are allowed to use basically everything in the standard library. HOWEVER, it would be smart to try and use the C++-ish versions of the functions you are used to in C, instead of just keeping to what you know, this is a new language after all. And NO, you are not allowed to use the STL until you actually are supposed to (that is, until d08). That means no vectors/lists/maps/etc... or anything that requires an include <algorithm> until then.

- Actually, the use of any explicitly forbidden function or mechanic will be punished by a -42, no questions asked.

- Also note that unless otherwise stated, the C++ keywords "using namespace" and "friend" are forbidden. Their use will be punished by a -42, no questions asked.

- Files associated with a class will always be ClassName.hpp and ClassName.cpp, unless specified otherwise.

- Turn-in directories are ex00/, ex01/, ..., exn/.

- You must read the examples thoroughly. They can contain requirements that are not obvious in the exercise's description. If something seems ambiguous, you don't understand C++ enough.

- Since you are allowed to use the `C++` tools you learned about since the beginning of the pool, you are not allowed to use any external library. And before you ask, that also means no `C++11` and derivates, nor `Boost` or anything your awesomely skilled friend told you `C++` can't exist without.

- You may be required to turn in an important number of classes. This can seem tedious, unless you're able to script your favorite text editor.

- Read each exercise FULLY before starting it ! Really, do it.

- The compiler to use is `g++`, not `gcc` nor `clang` !

- Your code has to be compiled with the following flags : `-Wall -Wextra -Werror`.

- Each of your includes must be able to be included independently from others. Includes must contains every other includes they are depending on, obviously.

- The subject can be modified up to 4h before the final turn-in time.

- In case you're wondering, no coding style is enforced during the `C++` pool. You can use any style you like, no restrictions. But remember that a code your peer-evaluator can't read is a code she or he can't grade.

- Important stuff now : You will NOT be graded by a program, unless explictly stated in the subject. Therefore, you are afforded a certain amount of freedom in how you choose to do the exercises. However, be mindful of the constraints of each exercise, and DO NOT be lazy, you would miss a LOT of what they have to offer !

- It's not a problem to have some extraneous files in what you turn in, you may choose to separate your code in more files than what's asked of you. Feel free, as long as the day is not graded by a program.

- Even if the subject of an exercise is short, it's worth spending some time on it to be absolutely sure you understand what's expected of you, and that you did it in the best possible way.

- By Odin, by Thor ! Use your brain !!!

Chapter II

Foreword

* Incoming transmission *

At ease, recruit.

I am Gregory Wasteland, head of the Department of Information and Technology Testing for the Terran Empire, and our Emperor Arcturus Mengsk. You have passed the previous screening tests with flying colors. Congratulations. But for now, the real test begins and maybe a few of you will become one of our honored G-TIPE (Ground Tactical Independant Programmer-Engineer). This test will simulate the programming of our field military equipments in real conditions.

In order to register your work for evaluation, You must create a file named `auteur` at the root of your turn-in directory. This file must contain your login followed by a newline. For instance, mine would be:

```
$>cat auteur
gwastela
$>
```

The test consists in 3 simulations with increasing difficulty. The simulations will put you in situations you might encounter on the battlefield. Be focused, because mistakes are not tolerated: the evaluation will stop at the slightest mistake. Such a mistake is the difference between living to see another day and annihilation when you face the countless Queen Of Blades' minions or the shady Protoss.

Trust my experience, in real life combat situations, time will be your most dangerous enemy. You will therefore have only 4 hours to complete these simulations. But keep in mind that a G-TIPE veteran can finish these simulations in 45 minutes flat while defending her or his working station.

Each simulation routines will be stored in an independant directory on your turn-in repository. Its name will match the simulation number. For instance the simulation `00` will be stored in the `ex00/` directory, simulation `01` will be stored in the `ex01/` directory, and so on.

Some simulations will use the `C++` routines produced in a previous simulation. You

will therefore have to copy the appropriate files when necessary.

Moreover, no `C++ main` function must be present in any of the files you turn in. We will use our own to evaluate your routines. That's why file names, class names and public identifiers are imposed to you. You have to follow them to the letter to earn your points. Please note that the identifiers you will have to write will always use this naming convention :

```
aFunctionName();
anotherName();
int     anInteger;
getAMemberData();
```

A few more words before I let you start this test. Obviously, the compiler to use is `g++`, along with the following flags: `-Wall -Wextra` and `-Werror`. Remember to protect your header files against multiple inclusion, but who would do such a trivial mistake anyway ?

Any C function (`*alloc, free, *printf*`, etc ...), along with the `C++` keywords `using` and `friend`, are illegal in the entire Terran Empire. Any use of them will be sanctioned by the execution of the `-42` order, which involves sending you a Ghost for immediate termination. Note that your body will remain property of the Empire and might be used for various scientific research, the details of which are not important here. This rule applies obviously in simulation conditions as well. But don't be silly: if a simulation's header mentions "no forbidden functions", it obviously means "no forbidden functions aside from the ones I mentioned here".

One last and very important rule: the `switch` control structure and successive `if` (and/or `else if`) are FORBIDDEN as we lost too many valuable men because of them. This rule WILL be verified during evaluation, and ANY use of them will be sanctioned by elimination, with no possible recourse or excuse. Only branchings smaller than or equal to `IF THEN (ELSE IF THEN ELSE)` are allowed. More reliable `C++` syntactic constructions are available to you. Use them.

Good luck, and be worthy of your Emperor.

* End of transmission *

Chapter III

Exercice 00: The Unit class

	Exercise 00
	Exercice 00: The Unit class
Turn-in directory : *ex*00/	
Files to turn in : `Unit.class.hpp`, `Unit.class.cpp`	
Forbidden functions : `Nothing`	
Remarks : `n/a`	

- In this simulation, we will test the following capabilities:

  - Basic class mechanics
  - Visibility (`private`, `protected` and `public`)
  - Member functions and attributes
  - Non-member functions and attributes (`static`)
  - Overloading of functions and operators
  - Canonical form

- Read this entire simulation before actually start to write code.

- Write a canonical class named `Unit`. I kindly remember you that a class is said to be canonical if it comports: a default constructor, a copy constructor, a destructor (`virtual` if needed) and an affectation operator (`"="`). All these elements are `public` within the `Unit` class.

- Add the following `protected` member attributes to your `Uunit` class :

  - Its index: `int`
  - Its type: `std::string`
  - Its X position: `int`
  - Its Y position: `int`

- The damage it causes: `int`

- Its current health points: `int`

- Its maximum health points: `int`

- The unit's index is different for every unit. The first unit created has the index `0` and each new unit receives the next available index. Even in copy construction, indexes must remain different. But in an assignation, the index is left unchanged. You need a non member attribute (`static`) to handle this behaviour.

- Whenever a new instance is created, the constructor displays: `"[CREATED] Unit #{INDEX}:  {TYPE} in {X}/{Y} with {cHP}/{mHP}HP damaging at {DAM}."`.

- Whenever an instance is destroyed, the destructor displays: `"[DESTRUCTED] Unit #{INDEX}:  {TYPE} in {X}/{Y} with {cHP}/{mHP}HP damaging at {DAM}."`.

- It must be possible to create an instance of the `Unit` class by specifying (in this order): its type, X position, Y position, damage, current HP and maximum HP. Else, the unit is initialized to its default values: `0` for the integers and `"Unit"` for its type.

- Write the following `public` accessors for the attributes :

  - `int getIndex( void ) const;`
  - `std:string const & getType( void ) const;`
  - `int getX( void ) const;`
  - `int getY( void ) const;`
  - `int getDam( void ) const;`
  - `int getCHP( void ) const;`
  - `int getMHP( void ) const;`

- Overload the `"<<"` operator with the following output: `"Unit #{INDEX}:  {TYPE} in {X}/{Y} with {cHP}/{mHP}HP damaging at {DAM}."`.

- Add the member function `"void die( void );"` that displays `"Unit #{INDEX} ({TYPE}) died."`. If a unit that dies still has HP, its HP are set to `0`. A dead unit can't do any action or receive additional damage. It's like, you know, dead.

- The only attribute of the class that is not read-only is the current HP. Overload the `"+="` and `"-="` operators to allow changing this attribute's value. Current HP can't go over the maximum HP: `IF cHP > mHP THEN cHP = mHP`. Also, current HP can't go below 0: `IF cHP < 0 THEN cHP = 0`. If a unit's HP drop to 0, call the `die` member function: the unit is dead.

- Add the member function `"void move( int x, int y );"`. This member function displays `"Unit #{INDEX} ({TYPE}) is moving from {Xf}/{Yf} to {Xt}/{Yt}."`, then moves the unit to the specified location.

- Add the member function `"void attack( Unit & target ) const;"`. This member function substracts the damage of the attacker from the target's current HP and displays: `"Unit #{ATT_INDEX} ({ATT_TYPE}) damaged unit #{TAR_INDEX} ({TAR_TYPE})."`.

The following code,

```cpp
int     main( void ) {

    Unit    u0;
    Unit    u1("Marine",    // type
            13, 45,         // posX, posY
            6,              // dam
            40, 40);        // cHP, mHP
    Unit    u2( u1 );

    u0 = u1;

    std::cout << "Index: "      << u0.getIndex() << std::endl;
    std::cout << "Type: "       << u0.getType()  << std::endl;
    std::cout << "X: "          << u0.getX()     << std::endl;
    std::cout << "Y: "          << u0.getY()     << std::endl;
    std::cout << "Damage: "     << u0.getDam()   << std::endl;
    std::cout << "Current HP: " << u0.getCHP()   << std::endl;
    std::cout << "Maximum HP: " << u0.getMHP()   << std::endl;

    std::cout << u0 << std::endl;
    std::cout << u1 << std::endl;
    std::cout << u2 << std::endl;

    u2.die();

    u0.move( 12, 45 );
    u1.move( 14, 45 );

    while ( u0.getCHP() > 0 && u1.getCHP() > 0 ) {
        u0.attack( u1 );
        u1.attack( u0 );
    }

    return 0;
}
```

Must produce the following output :

```
$>g++ -Wall -Wextra -Werror Unit.class.cpp main.cpp
$>./a.out | cat -e
[CREATED] Unit #0: Unit in 0/0 with 0/0HP damaging at 0.$
[CREATED] Unit #1: Marine in 13/45 with 40/40HP damaging at 6.$
[CREATED] Unit #2: Marine in 13/45 with 40/40HP damaging at 6.$
Index: 0$
Type: Marine$
X: 13$
Y: 45$
Damage: 6$
Current HP: 40$
Maximum HP: 40$
Unit #0: Marine in 13/45 with 40/40HP damaging at 6.$
Unit #1: Marine in 13/45 with 40/40HP damaging at 6.$
Unit #2: Marine in 13/45 with 40/40HP damaging at 6.$
Unit #2 (Marine) died.$
Unit #0 (Marine) is moving from 13/45 to 12/45.$
Unit #1 (Marine) is moving from 13/45 to 14/45.$
Unit #0 (Marine) damaged unit #1 (Marine).$
Unit #1 (Marine) damaged unit #0 (Marine).$
Unit #0 (Marine) damaged unit #1 (Marine).$
Unit #1 (Marine) damaged unit #0 (Marine).$
Unit #0 (Marine) damaged unit #1 (Marine).$
```

```
Unit #1 (Marine) damaged unit #0 (Marine).$
Unit #0 (Marine) damaged unit #1 (Marine).$
Unit #1 (Marine) damaged unit #0 (Marine).$
Unit #0 (Marine) damaged unit #1 (Marine).$
Unit #1 (Marine) damaged unit #0 (Marine).$
Unit #0 (Marine) damaged unit #1 (Marine).$
Unit #1 (Marine) damaged unit #0 (Marine).$
Unit #0 (Marine) damaged unit #1 (Marine).$
Unit #1 (Marine) died.$
[DESTRUCTED] Unit #2: Marine in 13/45 with 0/40HP damaging at 6.$
[DESTRUCTED] Unit #1: Marine in 14/45 with 0/40HP damaging at 6.$
[DESTRUCTED] Unit #0: Marine in 12/45 with 4/40HP damaging at 6.$
$>
```

9

Chapter IV

Exercice 01: The Marine class

	Exercise 01
	Exercice 01: The Marine class
Turn-in directory : *ex01/*	
Files to turn in : `Unit.class.hpp`, `Unit.class.cpp`, `Marine.class.hpp`, `Marine.class.cpp`	
Forbidden functions : `Nothing`	
Remarks : `n/a`	

- In this simulation, we will test the following capabilities:

    ○ Everything from the first simulation

    ○ Basic inheritance

    ○ Basic polymorphism by sub-typing (virtual member functions)

- Read this entire simulation before actually starting to write code.

- Our Marines are our colonies' first line of defense. The formidable armors they fight in allow them to resist extreme combat situations. These proud soldiers carry Gauss C-14 'Impaler' rifles, allowing them to unleash an impressive individual firepower.

- Copy your `Unit.class.hpp` and `Unit.class.cpp` files from the previous simulation's directory into the current one's.

- Edit the `Unit` class to declare the destructor, the member function `die` and the member function `move` as virtual member functions.

- The Marines' armor embeds its own information system represented by the `Marine` class.

- The `Marine` class inherits publicly from the `Unit` class.

- The `Marine` class is canonical. All canonical elements within this class are `public`.

- The construction of a marine requires its X and Y positions. As a unit, you will need to initialise its type, damage and health points.

- The type of a marine is `"Marine"`. This famous unit has 40HP and inflicts 6 points of damage.

- Upon creation, a Marine says: `"Unit #{INDEX}:  You want a piece of me, boy ?"`.

- Override the `die` virtual member function to display `"Unit #{INDEX}:  Aaaargh..."` on call after the regular output.

- Override the `move` virtual member function to display `"Unit #{INDEX}:  Rock'n'roll !!!"` on call after the regular output.

- Add the member function `"void stimpack( void );"`. This member function doubles the marine's damage and decreases his current HP by `10`. He answers: `"Unit #{INDEX}:  Ho yeah..."`. The stimpack can't decrease the Marine's HP below `1` HP.

The following code,

```cpp
int     main( void ) {

    Marine  m0;
    Marine  m1( 23, 56 );
    Marine  m2( m1 );

    m0 = m1;

    std::cout << m0 << std::endl;
    std::cout << m1 << std::endl;
    std::cout << m2 << std::endl;

    m2.die();

    m0.move( 22, 56 );
    m1.move( 24, 56 );

    m1.stimpack();

    while ( m0.getCHP() > 0 && m1.getCHP() > 0 ) {
        m0.attack( m1 );
        m1.attack( m0 );
    }

    return 0;
}
```

Must output the following:

```
$>g++ -Wall -Wextra -Werror Unit.class.cpp Marine.class.cpp main.cpp
$>./a.out | cat -e
[CREATED] Unit #0: Marine in 0/0 with 40/40HP damaging at 6.$
Unit #0: You want a piece of me, boy ?$
[CREATED] Unit #1: Marine in 23/56 with 40/40HP damaging at 6.$
Unit #1: You want a piece of me, boy ?$
[CREATED] Unit #2: Unit in 0/0 with 0/0HP damaging at 0.$
Unit #2: You want a piece of me, boy ?$
Unit #0: Marine in 23/56 with 40/40HP damaging at 6.$
Unit #1: Marine in 23/56 with 40/40HP damaging at 6.$
Unit #2: Marine in 23/56 with 40/40HP damaging at 6.$
Unit #2 (Marine) died.$
Unit #2: Aaaargh...$
Unit #0 (Marine) is moving from 23/56 to 22/56.$
Unit #0: Rock'n'roll !!!$
Unit #1 (Marine) is moving from 23/56 to 24/56.$
Unit #1: Rock'n'roll !!!$
Unit #1: Ho yeah...$
Unit #0 (Marine) damaged unit #1 (Marine).$
Unit #1 (Marine) damaged unit #0 (Marine).$
Unit #0 (Marine) damaged unit #1 (Marine).$
Unit #1 (Marine) damaged unit #0 (Marine).$
Unit #0 (Marine) damaged unit #1 (Marine).$
Unit #1 (Marine) damaged unit #0 (Marine).$
Unit #0 (Marine) damaged unit #1 (Marine).$
Unit #1 (Marine) damaged unit #0 (Marine).$
Unit #0 (Marine) died.$
Unit #0: Aaaargh...$
[DESTRUCTED] Unit #2: Marine in 23/56 with 0/40HP damaging at 6.$
[DESTRUCTED] Unit #1: Marine in 24/56 with 6/40HP damaging at 12.$
[DESTRUCTED] Unit #0: Marine in 22/56 with 0/40HP damaging at 6.$
$>
```

Chapter V

Exercise 02: The SCV class

	Exercise 02
	Exercise 02: The SCV class
Turn-in directory : *ex02/*	
Files to turn in : `Unit.class.hpp`, `Unit.class.cpp`, `Marine.class.hpp`, `Marine.class.cpp`, `Scv.class.hpp`, `Scv.class.cpp`, `IBuilder.interface.hpp`	
Forbidden functions : `Nothing`	
Remarks : `n/a`	

V.1 Warm-ups

- In this simulation, we will test your following capabilities:

  - Everything from the second simulation
  - Interfaces and pure virtual member functions
  - Simple multiple inheritance
  - Pointers to members
  - Insertion of code in a pre-existing environnement

- Read this entire simulation before actually start to write code.

- Copy your `Unit.class.hpp`, `Unit.class.cpp`, `Marine.class.hpp` and `Marine.class.cpp` files from the previous simulation's directory into the current one's.

- Designed for repairs on space platforms of Tarsonis, the SCV has proven its usefulness and reliability in space. Fast and efficient, its use is widely spread in campaigns involving the construction of forward bases.

- SCVs embed complex building and gathering routines that interact with other pieces of code that you are provided with for this simulation. See the `misc/` directory.

    - `Ressource.class.hpp` and `Ressource.class.o`
    - `Building.class.hpp` and `Building.class.o`
    - `Barracks.class.hpp` and `Barracks.class.o`
    - `Bunker.class.hpp` and `Bunker.class.o`
    - `CommandCenter.class.hpp` and `CommandCenter.class.o`
    - `SupplyDepot.class.hpp` and `SupplyDepot.class.o`

- Header files are classes declarations. Object files are compiled implementations of said classes.

- `Ressource.class.hpp` declares the `Ressource`, `Mineral` and `Gas` classes. `Mineral` and `Gas` classes inherit from the `Ressource` class.

- The five other header files declare respectively the `Building`, `Barracks`, `Bunker`, `CommandCenter` and `SupplyDepot` classes. Note that `Barracks`, `Bunker`, `CommandCenter` and `SupplyDepot` inherit `Building`.

- If you want a better idea of these classes, read their declaration.

- Copy these 12 files into the simulation's directory or create links to them. Please remember that these files are not part of the files you have to turn in. None of them must be present in your turn-in repository.

- For the sake of clarity, the 4 files from the previous simulation must be turned in. Please refer to the header of this simulation if any doubt persists.

- You should now have 16 files before starting this simulation: the 4 from the previous simulation and the 12 that are provided for this simulation.

V.2   The `IBuilder` interface

- First, let's focus on the construction capabilities of the SCV. Create a new file named `IBuilder.interface.hpp`. This file will contain an interface `IBuilder` that units with construction capabilities must implement. Therefore, the `Scv` class we'll focus on later will implement this interface.

- The interface defines an `enum building_e` the definition domain of which is the available buildings: `COMMAND_CENTER`, `BARRACK`, `SUPPLY_DEPOT` and `BUNKER`. The `COMMAND_CENTER` value must be 1. This `enum` definition is `public`.

- Like any interface, the destructor must be `virtual`. Remember to define it empty for obvious `C++` semantics.

- Declare a `public` pure virtual member function "Building * createBuilding(building_e building, int x, int y);".

- To each building corresponds a protected pure method:
"Building * _build{NameOfTheBuilding}(int x, int y);". Obviously the placeholder {NameOfTheBuilding} must be replaced by the name of the building.

V.3   The `Scv` class

V.3.1   The basics

- Create a new file name `Scv.class.hpp` to declare a `Scv` class that publicly inherits from the `Unit` class and publicly implements the `Ibuilder` interface.

- The `Scv` class is canonical. All its canonical elements are `public`.

- The type of a SCV is `SCV T-280`, it has `60`HP and inflicts `5` points of damage.

- Its X and Y positions are specified at construction.

- Upon construction, a SCV will display: "Unit #{INDEX}: SCV good to go, sir.".

- When it dies, a SCV will display: "Unit #{INDEX}: * noise of an exploding SCV *.".

- Upon a move order, a SCV will display: "Unit #{INDEX}: Affirmative.".

V.3.2   The gathering routines

- One major role of the SCV on the battlefield is to gather ressources. These ressources will allow the construction of new units and buildings. There are two kind of ressources : Mineral and Vespene gas. Please refer to the provided file `Ressources.class.hpp`.

- Gathering ressources is a two-phase action: first, go to the ressource and actually gather it; second, return the gathered ressources to a Command Center.

- The `Ressource` class provides a `gather` member function that returns up to 8 units of the ressource upon call according to the remaining units of the ressource.

- The `CommandCenter` class provides an `acceptRessources` member function that take the gathered resources from a returning SCV.

- First part of the action: go to the ressource and actually gather it. Add the "void gatherRessources(Ressource & ressource);" member function to your `Scv` class. A SCV can gather a ressource if and only if they have the same coordinates. Else, the SCV will move **silently** to the ressource coordinates. Then if and only if the ressource is not depleted and the SCV does not already carry ressource units,

the SCV gathers the ressource by calling the `Ressource`'s `gather` member function then displays: `"Unit #{INDEX} ({TYPE}) gathered {N} {RESSOURCE}."`. The placeholder `{N}` stands for the amount of ressource unit gathered. The placeholder `{RESSOURCE}` stands for either "minerals", either "gas" according to the ressource'type.

- Second part of the action: return the gathered ressources to a Command Center. Add the `"void returnRessources( CommandCenter & cc );"` member function to your `Scv` class. A SCV can return ressource units to a Command Center if and only if they have the same coordinates. Else, the SCV will move **silently** to the Command Center coordinates. The SCV then returns the ressource units by calling the `CommandCenter`'s `acceptRessources` member function then displays: `"Unit #{INDEX} ({TYPE}) returned {N} {RESSOURCE}."`. Returning ressources empties the SCV.

- Please note that's it up to you to define how an `Scv` instance stores the amount and type of the gathered ressource.

V.3.3   The building routines

- The `Scv` class implements the `IBuilder` interface and therefore must implement the previously declared pure virtual member functions:

  - `"Building * createBuilding( building_e building, int x, int y );"`
  - `"Building * _buildCommandCenter( int x, int y );"`
  - `"Building * _buildBarrack( int x, int y );"`
  - `"Building * _buildSupplyDepot( int x, int y );"`
  - `"Building * _buildBunker( int x, int y );"`

- Virtual member functions such as `createBuilding` are called "factory methods" because this member function will create different buildings according to its parameters and return them as a pointer to their base class. Hence the "factory". To create these buildings, `createBuilding` calls the according `"_build{NameOfTheBuilding}"` member function that only creates one type of building.

- In the implementation of the four specialised building creation member functions, the SCV will display the following messages:

  - _buildCommandCenter: `"Unit #{INDEX}:  Command center finished, sir."`.
  - _buildBarrack: `"Unit #{INDEX}:  Barrack finished, sir."`.
  - _buildSupplyDepot: `"Unit #{INDEX}:  Supply depot finished, sir."`.
  - _buildBunker: `"Unit #{INDEX}:  Bunker finished, sir."`.

- Please remember that only branchings smaller than or equal to `IF THEN (ELSE IF THEN ELSE)` are allowed. Instead, you must use pointers to member functions in the `createBuilding` member function. A smart `enum building_e` value to function table should do the trick.

- As a matter of security, if the `createBuilding` member function is called with a
  `enum building_e` value that does not make sense, the SCV will display: `"Unit`
  `#{INDEX}:  No such building, sir."`.

The following code,

```cpp
int              main( void ) {

     Ressource * r0 =        new Mineral( 10, 10, 15 );
     Ressource * r1 =        new Mineral( 11, 9, 5000 );
     Ressource * r2 =        new Gas( 14, 11, 5000 );

     CommandCenter        cc( 12, 13 );
     Building *              ba =        NULL;
     Building *              sd =        NULL;
     Building *              bu =        NULL;

     Scv *       s0 =        cc.createScv();
     Scv *       s1 =        cc.createScv();
     Scv *       s2 =        cc.createScv();

     Marine *       m0 =        NULL;
     Marine *       m1 =        NULL;

     while( !(r0->isDepleted()) ) {
                s0->gatherRessources( *r0 );
                s1->gatherRessources( *r1 );
                s2->gatherRessources( *r2 );
                s0->returnRessources( cc );
                s1->returnRessources( cc );
                s2->returnRessources( cc );
        }

     ba = s0->createBuilding( IBuilder::BARRACKS, 8, 17 );
     m0 = static_cast<Barracks *>( ba )->createMarine();
     m1 = static_cast<Barracks *>( ba )->createMarine();

     sd = s1->createBuilding( IBuilder::SUPPLY_DEPOT, 16, 11 );
     bu = s2->createBuilding( IBuilder::BUNKER, 16, 19 );

     delete bu;
     delete sd;
     delete ba;

     delete m0;
     delete m1;

     delete s0;
     delete s1;
     delete s2;

     delete r0;
     delete r1;
     delete r2;

     return 0;
}
```

Must output the following:

```
$>g++ -Wall -Wextra -Werror *.o Unit.class.cpp Marine.class.hpp Scv.class.cpp main.cpp
$>./a.out | cat -e
[CREATED] Ressource #0: Mineral deposit in 10/10 with 15 unit
remaining.$
[CREATED] Ressource #1: Mineral deposit in 11/9 with 5000 unit
remaining.$
[CREATED] Ressource #2: Vespen gas deposit in 14/11 with 5000 unit
remaining.$
[CREATED] Building #0: Command Center in 12/13 with 1500/1500HP.$
```

```
[CREATED] Unit #0: SCV T-280 in 12/13 with 60/60HP damaging at 5.$
Unit #0: SCV good to go, sir.$
[CREATED] Unit #1: SCV T-280 in 12/13 with 60/60HP damaging at 5.$
Unit #1: SCV good to go, sir.$
[CREATED] Unit #2: SCV T-280 in 12/13 with 60/60HP damaging at 5.$
Unit #2: SCV good to go, sir.$
Unit #0 (SCV T-280) is moving from 12/13 to 10/10.$
Unit #0 (SCV T-280) gathered 8 minerals.$
Unit #1 (SCV T-280) is moving from 12/13 to 11/9.$
Unit #1 (SCV T-280) gathered 8 minerals.$
Unit #2 (SCV T-280) is moving from 12/13 to 14/11.$
Unit #2 (SCV T-280) gathered 8 gas.$
Unit #0 (SCV T-280) is moving from 10/10 to 12/13.$
Unit #0 (SCV T-280) returned 8 minerals.$
Unit #1 (SCV T-280) is moving from 11/9 to 12/13.$
Unit #1 (SCV T-280) returned 8 minerals.$
Unit #2 (SCV T-280) is moving from 14/11 to 12/13.$
Unit #2 (SCV T-280) returned 8 gas.$
Unit #0 (SCV T-280) is moving from 12/13 to 10/10.$
Unit #0 (SCV T-280) gathered 7 minerals.$
Unit #1 (SCV T-280) is moving from 12/13 to 11/9.$
Unit #1 (SCV T-280) gathered 8 minerals.$
Unit #2 (SCV T-280) is moving from 12/13 to 14/11.$
Unit #2 (SCV T-280) gathered 8 gas.$
Unit #0 (SCV T-280) is moving from 10/10 to 12/13.$
Unit #0 (SCV T-280) returned 7 minerals.$
Unit #1 (SCV T-280) is moving from 11/9 to 12/13.$
Unit #1 (SCV T-280) returned 8 minerals.$
Unit #2 (SCV T-280) is moving from 14/11 to 12/13.$
Unit #2 (SCV T-280) returned 8 gas.$
Unit #0 (SCV T-280) is moving from 12/13 to 8/17.$
Unit #0: Affirmative.$
Unit #0: Barrack finished, sir.$
[CREATED] Building #1: Barracks in 8/17 with 1000/1000HP.$
[CREATED] Unit #3: Marine in 8/17 with 40/40HP damaging at 6.$
Unit #3: You want a piece of me, boy ?$
[CREATED] Unit #4: Marine in 8/17 with 40/40HP damaging at 6.$
Unit #4: You want a piece of me, boy ?$
Unit #1 (SCV T-280) is moving from 12/13 to 16/11.$
Unit #1: Affirmative.$
Unit #1: Supply depot finished, sir.$
[CREATED] Building #2: SupplyDepot in 16/11 with 500/500HP.$
Unit #2 (SCV T-280) is moving from 12/13 to 16/19.$
Unit #2: Affirmative.$
Unit #2: Bunker finished, sir.$
[CREATED] Building #3: Bunker in 16/19 with 350/350HP.$
[DESTRUCTED] Building #3: Bunker in 16/19 with 350/350HP.$
[DESTRUCTED] Building #2: SupplyDepot in 16/11 with 500/500HP.$
[DESTRUCTED] Building #1: Barracks in 8/17 with 1000/1000HP.$
[DESTRUCTED] Unit #3: Marine in 8/17 with 40/40HP damaging at 6.$
[DESTRUCTED] Unit #4: Marine in 8/17 with 40/40HP damaging at 6.$
[DESTRUCTED] Unit #0: SCV T-280 in 8/17 with 60/60HP damaging at 5.$
[DESTRUCTED] Unit #1: SCV T-280 in 16/11 with 60/60HP damaging at 5.$
[DESTRUCTED] Unit #2: SCV T-280 in 16/19 with 60/60HP damaging at 5.$
[DESTRUCTED] Ressource #0: Mineral deposit in 10/10 with 0 unit
remaining.$
[DESTRUCTED] Ressource #1: Mineral deposit in 11/9 with 4984 unit
remaining.$
[DESTRUCTED] Ressource #2: Vespen gas deposit in 14/11 with 4984 unit
remaining.$
[DESTRUCTED] Building #0: Command Center in 12/13 with 1500/1500HP.$
$>
```