B4- C++

B-PAV-330

Arcade

A modular game platform

v1.1

Arcade

A modular game platform

repository name:	cpp_arcade
repository rights:	ramassage-tek
language:	C++
group size:	2-3

- Your repository must contain the totality of your source files, but no useless files (binary, temp files, obj files,...).
- All the bonus files (including a potential specific Makefile) should be in a directory named *bonus*.
- Error messages have to be written on the error output, and the program should then exit with the 84 error code (0 if there is no error).

In this project, you will have to create a gaming platform. This platform must include at least two games and must be able to run with three different GUIs. Games and GUIs will be in dynamicly loaded libraries. You must have a small set of games to choose from, as well as a small set of graphics libraries.

1 Nibbler, Snake, Blockade

`Nibbler` is a small arcade video game released in 1982. Its concept has mainly spread through the cult game `Snake`. `Nibbler` was itself inspired by another great classic: `Blockage`, also inspired `Tron Light Cycle`, which is almost a clone of it.

The simplicity and the addictiveness of `Snake` make it available on almost every existing platform under various names.

For the oldest of the youngsters, `Snake` means hours of procrastination in high school on old `Nokia 3310` phones. For younger people, `Snake` is a tech2 project referring to a certainly underground pre-historic game, which only interests the old fossils pretending to be in the trend of their time.

It is never too late to discover this classic of video game history: `http://jeux-flash.jeu-gratuit.net/jeux_classiques/snake_250.html`

As you can see, `Snake` is about moving a snake around a map. This snake is represented by sections and it has to eat food, which will make it grow by one section every time. The game is over when the head of the snake hits an edge of the map or one of the sections. The goal of the game is to make the snake as long as possible.

Several versions of `Snake` exist. Some of them include obstacles, others have a score system, or bonuses etc.

2 Pacman

Pacman is an arcade video game from 1980. The goal is to explore a maze in order to eat all the “pacgums” in it while avoiding ghosts.

Some “pacgums” let the player invert roles: Pacman can, during a short period of time, eat ghosts instead of being eaten. Eaten ghosts do not disappear: their eyes must go back to an unaccessible zone in the middle of the screen. They change back to normal ghosts after a short time.

http://jeux-flash.jeu-gratuit.net/jeux_peur/pacman-flash_4989.html

3 Qix

Qix is an arcade video game from 1981. The game presents a space which contains a monster: the Qix. The player can move inside this space and leaves a trail behind him. When the player, after he goes through a border of the screen, comes back to another border, the whole zone which was isolated from the Qix disappears.

The player wins when the remaining space containing the Qix goes under 25% of the original size.

The player loses if the Qix crosses the trail.

The player's trail burns from its origin to the player itself if the player stops. If the fire catches him, he loses.

There are also monsters on the space 'drawn' by the player. If one of these monsters touches the player, he loses.

http://jayisgames.com/games/qix/
https://www.youtube.com/watch?v=nBt4w3qKI6I

4 Centipede

Centipede is an arcade video game from 1981. The game features an area with a lot of empty space and some blocking boxes. The player is at the bottom of the screen and can move in every direction with some limitations: he can only move up and down by a little. However, he can move left and right without restriction. The player can also shoot projectiles towards the top of the screen.

Regularly, Centipedes appear at the top of the screen. They move from left to right or right to left and go down when they encounter a blocking box or the border of the screen border. If one of them touches the player, he loses. A Centipede has a head, a tail and a body in between.

When a shot from the player touches a Centipede, the touched part transforms into a blocking box and the body is split in two, becoming two different smaller Centipedes. The Centipede which is from the tail part encounters the newly formed blocking box and heads back on the next line.

Blocking boxes can be destroyed by several shots from the player if he insists a little.

There can be only one shot at on the screen at any given time.

http://my.ign.com/atari/centipede

5 Solar Fox, Space Fighter

Solar Fox is an arcade video game from 1981. The game takes place in space and the player is in command of a space ship. The playing field is a grid. Some part of the grid contains a powerup that can be picked up with a shot, almost in the same way as in Pacman.

The main difference with Pacman is the shape of the game area, and the way enemies and the player move.

- In Pacman, the game area is a small maze. In Solar Fox, it is completely free of obstacles.
- Enemies in Pacman move inside the maze and chase the player. In Solar Fox, enemies are guns on the border of the game area and shoot in a straight line.
- Last detail, in Solar Fox, the player cannot stop. He can shoot a small laser which can break powerups and intercepts enemy shots.

Solar Fox is a dodge and collect game. Many clones added functionalities and speed, which make Solar Fox a classic arcade game.

https://www.youtube.com/watch?v=eTeC8Za9oSs
https://www.youtube.com/watch?v=ZKDgr2YxysA

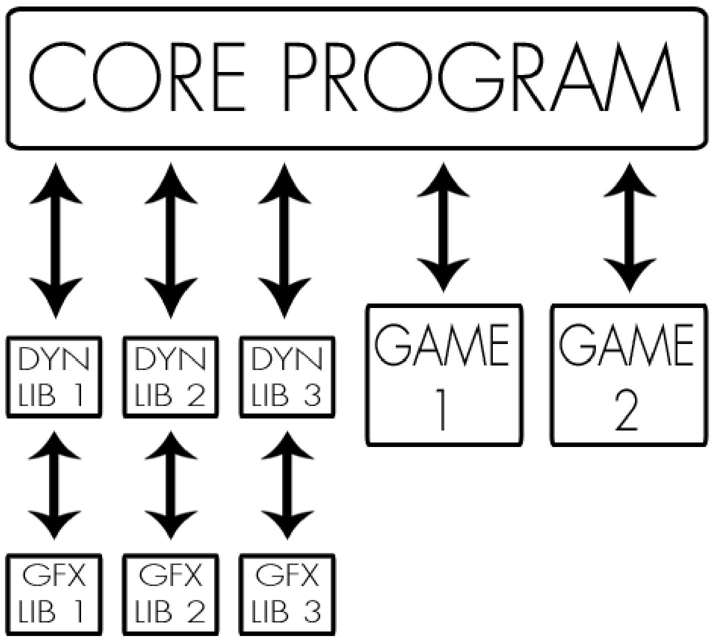
6 The project

Arcade is a program that lets the user choose a game to play and that keeps score. When the program starts, it must display in separated boxes:

- All games available (In the ./games/ directory)
- All graphic libraries available in the ./lib/ directory)
- Scores
- A field for the user to enter a name

6.1 Generalities

For your own culture, it is very important to know several graphic libraries. This is why your final project must provide 3 different graphic libraries. The point of this project being to make you handle dynamic libraries at run-time, the graphic rendering as well as output must be located within a dynamic library. In order to improve your software design, you will also have to put your games in dynamic libraries. The body of your program must interact the same way with all of your libraries.



Each GUI available for the program must be used as a shared library that will be loaded and used dynamically by the main program. It is strictly **FORBIDDEN** to refer to one graphic library or another in your main program. Only your dynamic libraries can do it! This also applies to your games.

Again: It is strictly **FORBIDDEN** to refer to one graphic library or another in your main program.

6.2 Dynamic libraries

You must use your dynamic libraries at run-time. This means that you MUST use the `dlopen`, `dlclose`, `dlsym` and `dlerror` functions to handle your dynamic libraries. Dependency to your libraries will not appear when passing your program as parameter to the command `ldd`.

These dynamic libraries can be seen as plug-ins providing various capabilities for your main program. In NO CASE must graphic libraries influence the game logic. Game libraries must not contain any information about screen rendering or low level events.

You MUST NOT make any difference between one library or another in your main program. Each of your libraries MUST be handled in a generic and uniform manner. The generic aspect is what matters to your grader during the project defense.

Class and material available on the e-learning, as well as the tutorials, are important sources of information to ease you through this part of the project. However, nothing replaces good thinking.

6.3 Graphic libraries

You must choose three graphic libraries. Choose one from each group.

Group 1:
- NCurses
- NDK++
- aa-lib
- libcaca

Group 2:
- Allegro4/ Allegro5
- Xlib
- GTK++
- SFML
- SDL1/ SDL2

Group 3:
- OpenGL
- Vulkan
- Qt
- LibLapin

If you want to use a graphics library that is not in this list, contact your local team to get authorization. You can also ask on the school network so that your schoolmates benefit from the answer.

Some of these libraries cannot be installed on the standard dumps of the school. You should then consider that you will have to manually install these libraries, without privileged rights. You will acquire more than culture at this point.

6.4 Games: Nibbler

Rules are simple and must be respected.

- The measuring unit is the "cell". The size of a cell is up to you, but it MUST be reasonable and CAN vary with the graphics libraries you are using. A 1px cell is too small while a 1000px is too large.
- The game area is a finite amount of cells. The edges of the area can't be passed through.
- The snake starts with a size of 4 cells in the middle of the game area.
- The snake moves forwards automatically at a constant speed. Each section of its tail follows the exact same path as the head.
- The snake can turn right or left when the corresponding key is pressed.
- The goal of this game is to feed your snake so that it can grow. The game area MUST NEVER have less that one element of food.
- A food element fills only one cell.
- When the head of the snake goes over a cell with food, the food disappears, and a one cell-long section is added at the tail of the snake. The new section appears in the first free tile next to the last cell of the tail. If there is no free cell, the game is over. If a new section is added, then a new food element appears.

Once you are done with games and the 3 graphics libraries, you can look for a bonus by extending the game rules. Here are some examples:

- Bonus food appears for a short period of time
- The head section looks different from the other sections
- Movement speed increases during the game
- The game area has obstacles
- The size of the snake increases randomly when eating
- A speed boost when pressing the space bar
- ...

6.5 Games: Pacman

Rules are simple and must be respected.

- The measuring unit is the "cell". The size of a cell is up to you, but it MUST be reasonable and CAN vary with the graphics libraries you are using. A 1px cell is too small while a 1000px one is too large.
- The game area has a specific size. Going through the side of the area makes the player appear on the opposite side. All cells that are not walls may be walked through and contain the "pacgum" powerup.
- In the middle of the map is a small area 5 cells wide and 4 cells high that contains ghosts.
- Ghosts can get out of their box 10 seconds after the game starts.
- Pacman starts the game right under the ghosts.
- Some special, bigger "pacgums" let Pacman eat ghosts. This effect lasts 10 seconds. During this period, ghosts become blue and instead of hunting Pacman, they flee him. Their movement speed is quite slow during this period. There are only 4 "pacgums" of this kind on the map.
- When Pacman eats a ghost, only its eyes remain. These eyes quickly go back to the ghost box where the ghost is healed after a short period of time.
- The player wins when Pacman eats all "pacgums". A new map is loaded right after, or the current one is reloaded and goes faster.
- On screen, Pacman and ghosts must not move cell by cell, but move smoothly.

Once you are done with your project and the 3 dynamic libraries, you can look bonuses by extending the game rules. Here are some examples:

- Food appears for a short period of time at the start position of Pacman. It provides a powerup or huge score bonus.
- Pacman and a few ghosts can jump, just like in `Pacmania`: https://www.youtube.com/watch?v=lX07p897vZU
- The game speed increases
- The game features a camera, like in `Pacmania`.
- ...

6.6 Games: Qix

Rules of the game are very basic and MUST be respected. These are the bases:

- The measuring unit is the "cell". The size of a cell is up to you, but it MUST be reasonable and CAN vary with the graphics libraries you are using. A 1px cell is too small while a 1000px one is too large.
- The game area has a specific size. A cell can contain three different values that indicate its type: the cell can be walkable, it can be non walkable, or it can be a border. A border is a special walkable area.
- Non walkable cells are space that were taken out by a player action. Borders are cells that are directly in touch with a walkable case and at least one non walkable cell.
- The walkable area contains a monster: the `Qix`, which is several cells long and moves in a random manner (You have to find a nice way to make him dreadful without having to hunt the player) If the `Qix` touches the player or his trail, the player loses and goes back to the border he came from.
- When the player goes on a walkable cell, he leaves a trail behind him. This trail ignites if he stops. When he touches a border after walking inside the walkable area, the walkable area splits: the area which contains the `Qix` stays and the other becomes non walkable. Note that the player cannot cross the trail.
- Borders contain special monsters, called `Sparks`. Sparks make the player lose by touching him. Sparks can also go on the trail or old borders, even if they are in a non walkable area, if it is of interest in order to hunt the player. A spark cannot turn back.
- To win, the `Qix` must be sealed inside less than 25% of the starting area.

Once you are done with your project and the 3 dynamic libraries, you can look for bonuses by extending the game rules. Here are some examples:

- Powerups may appear in the walkable area. The player may touch the powerup directly or with the trail. The effects of powerups are for you to decide.
- Maps may contain obstacles or scenery.
- ...

6.7 Games: Centipede

Rules of the game are very basic and MUST be respected. These are the bases:

- The measuring unit is the "cell". The size of a cell is up to you, but it MUST be reasonable and CAN vary with the graphics libraries you are using. A 1px cell is too small while a 1000px one is too large.
- The game area is split into two parts: the first is walkable and the other is not. The walkable area is at the bottom of the screen. The walkable part fills the entire width of the screen but only covers 20% of the height.
- Centipedes are coming from the top of the screen. They move from side to side and each obstacle or screen border makes them change direction and go down to next line. A centipede is a snake composed of several parts.
- The player can shoot. When a shot hits a centipede, it is split in half. The part that was hit turns into an obstacle, the head part keeps moving on and the tail part collides with the obstacle, turns back and moves on to the next line.
- If a centipede touches the player, the player loses the game.
- If a centipede touches the bottom of the screen, the player loses score.
- To win, the player must survive 10 centipedes. The map is then reset and the game starts over.
- A `Centipede` map contains randomly generated obstacles.
- An obstacle can be destroyed by a series of 5 shots.
- There can only be one shot on the screen at any given time.

Once you are done with your project and the 3 dynamic libraries, you can look for bonuses by extending the game rules. These are some examples:

- Several centipedes on screen at the same time.
- Several type of centipedes: length, speed, movement pattern.
- The player may play two characters instead of one.
- ...

6.8 Games: Solar Fox

Rules of the game are very basic and MUST be respected. These are the bases:

- The measuring unit is the "cell". The size of a cell is up to you, but it MUST be reasonable and CAN vary with the graphics libraries you are using. A 1px cell is too small while a 1000px one is too large.
- The game area is split in two parts: a central part that is walkable and has a margin of 2 or 3 cells between its limit and the screen border. The other part is non walkable and contains opponents.
- The player can move around the walkable area. He can move in every direction, but cannot turn back directly. The player cannot stop: he always moves forward. There is even a key to go faster.
- The walkable area is filled with powerups that the player must break by shooting to win.
- Opponents appear at the border of the walkable area and shoot.
- Opponents' shots can be destroyed by the player's shots.
- Player's shots only have a two cell range. Their speed is three or four times faster than the player's ship's speed.
- The player loses the game if his spaceship is touched by a shot, by special bad powerups or by hitting the walkable area's borders.
- The shaceship, lasers, opponents, must not have a cell per cell movement. It must be fluid.

Once you are done with your project and the 3 dynamic libraries, you can look for a bonus by extending the game rules. These are some examples:

- Powerups in the walkable area.
- Opponents with different kinds of shots or that move.
- Obstacles, indestructible or not, that appear in the game area.
- ...

6.9 Usage

The results of your turn in directory compilation MUST BE a program, 3 graphics dynamic libraries and at least two games (also dynamic libraries).
The program MUST be named "arcade" and the dynamic libraries MUST be named according to the graphics or game library they are exposing. Only the executable name is imposed. However the libraries SHOULD have a name similar to "lib_arcade_XXX.so", where "XXX" is the name of the graphics library used or the name of the game.

The "arcade" executable must take the graphics library to use as a startup argument. It must nevertheless be possible to change the graphics library at runtime.

Example :

```
>./arcade ./lib/lib_arcade_opengl.so
```

You MUST handle the following cases:

- If the number of arguments passed to your program is different from 1, your program MUST display a usage and exit properly.
- If the dynamic library does not exist or is not compatible, your program MUST display a relevant error message and exit properly.

When your program is running, these keys (or any other mapping) MUST behave in the following way:

- '2' : Previous graphics library.
- '3' : Next graphics library.
- '4' : Previous game.
- '5' : Next game.
- '8' : Restart the game.
- '9' : Go back to the menu.
- Escape: Exit.

7 Mandatory Protocol

You MUST implement an IO system with STDIN and STDOUT in your game library. This system will let a program play a game, for example: the automatic correction program. Two pipes will be linked to your program.

In order to do so, all you game libraries will feature the **void Play(void)** function that will start the game in a specific manner. The automatic correction system will call this function and play a little.

In this mode, your game will have to be synchronous: your game must only advance when it is asked to by the automatic player, with a specific command.

The IO system you will have to use is a binary one. It means you will exchange structures and not text on STDIN and STDOUT.

A little example with a GET_MAP command and an 8x8 map:

Terminal

```
~/B-PAV-330> ls
main.cpp libarcade_starcraft.so
~/B-PAV-330> cat main.cpp
// The famous header
extern "C" void Play(void);

int main(void)
{
    Play();
    return (0);
}
~/B-PAV-330> gcc main.cpp -L./ -larcade_starcraft -o starcraft
~/B-PAV-330> /bin/echo -en "\01\0" | ./starcraft | hexdump -C
00000000 01 00 08 00 08 00 01 00 01 00 01 00 01 00 01 00 |................|
00000010 01 00 01 00 01 00 01 00 00 00 00 00 00 00 06 00 |................|
00000020 00 00 00 00 01 00 01 00 00 00 00 00 00 00 01 00 |................|
00000030 00 00 00 00 01 00 01 00 00 00 00 00 00 00 00 00 |................|
*
00000070 00 00 00 00 01 00 01 00 01 00 01 00 01 00 01 00 |................|
00000080 01 00 01 00 01 00                               |......|
00000086
```

```
// Kind of Advanced Language Assistant Laboratory 2006-2042
// Epitech 1999-2042
// Jason Brillante brilla_a brilla_b
// Have you played Atari today?
//
// WELCOME TO THE ARCADE

#ifndef                 __ARCADE_PROTOCOL_HPP__
# define                __ARCADE_PROTOCOL_HPP__
# include               <stdint.h>

namespace               arcade
{
  enum class            CommandType : uint16_t
  {
      WHERE_AM_I        = 0,            // RETURN A WHERE AM I STRUCTURE
      GET_MAP           = 1,            // RETURN A GETMAP STRUCTURE
      GO_UP             = 2,            // MOVE THE CHARACTER UP
      GO_DOWN           = 3,            // MOVE THE CHARACTER DOWN
      GO_LEFT           = 4,            // MOVE THE CHARACTER LEFT
      GO_RIGHT          = 5,            // MOVE THE CHARACTER RIGHT
      GO_FORWARD        = 6,            // USELESS
      SHOOT             = 7,            // SHOT (FOR SOLAR FOX AND CENTIPEDE)
      ILLEGAL           = 8,            // THE INSTRUCTION WAS ILLEGAL
      PLAY              = 9             // PLAY A ROUND
    };

  enum class            TileType : uint16_t
    {
      EMPTY             = 0,            // TILE WHERE THE CHARACTER CAN GO
      BLOCK             = 1,            // TILE WHERE THE CHARACTER CANNOT GO
      OBSTACLE          = 2,            // FOR CENTIPEDE
      EVIL_DUDE         = 3,            // EVIL DUDE
      EVIL_SHOOT        = 4,            // EVIL SHOOT
      MY_SHOOT          = 5,            // YOUR OWN SHOT
      POWERUP           = 6,            // POWERUP
      OTHER             = 7             // ANYTHING THAT WILL BE IGNORED BY THE KOALINETTE
    };

  /// The format is width, height, and width * height * sizeof(TileType) quantity of TileType
  struct                GetMap
  {
    CommandType         type;
    uint16_t            width;
    uint16_t            height;
    TileType            tile[0];        // SPECIFY AN ADDITIONAL SIZE WHILE ALLOCATING
  } __attribute__((packed));

  /// The format is length, length * Position quantity of TileType
  struct                Position
  {
    uint16_t            x;
    uint16_t            y;
  } __attribute__((packed));

  struct                WhereAmI
  {
    CommandType         type;
    uint16_t            lenght;
    Position            position[0];    // SPECIFY AN ADDITIONAL SIZE WHILE ALLOCATING
  } __attribute__((packed));
}

#endif  //              __ARCADE_PROTOCOL_HPP__
```

Think twice, read this paragraph several times. We are asking you to implement a way for the koalinette to play your games turn by turn. Each call to Play() asks your program to play a turn, and send some binary data.
In order to do that, and for the koalinette to understand your program, you MUST implement the tiny protocol described in the .h file. Each structure represents a piece of information the koalinette may ask for. The **tab[0]** at the end of the structures is a zero length array. Google is your friend.
This part is really not hard once you understand how to write and read structures, and you do know how to do that from various C projects you had to do. It is even easier with C++.

8 Documents

More than any other kind of program, a program that can be extended like the Arcade project MUST have a documentation.

In order to clarify the way your program and interface work, you will write a documentation. Here is the required information:

- You must provide a class diagram of your program, featuring at least the links between classes and their public member functions.
- Some explanation in a manual that goes with your diagram and describes how procedures are linked in your program.
- Also explain how to create a dynamic library for graphics or for a game that is compatible with your system.

9 Share your interface

We often have to collaborate with other teams. This collaboration may occur on the same project, but also sometimes on shared formats.

In Arcade, there are three parts that can be shared: the score file format, the interface for graphics libraries and the interface for game libraries.

We ask that you collaborate with one other team. Establish a "standard" that contains a description of your score file, graphics libraries interface and game libraries interface. You will have to both implement libraries and programs that use this "standard".
This is NOT an option. You MUST design these all 3 parts together and respect them completely. We WILL try to copy a game from a group and test it with the launcher of the other group, and vice versa.

Be sure to come together, at the same time, to your defense.

10 Rules

You are more or less free to implement your program however you want to. However there are certain rules:

- The only functions of the `libc` that are authorized are those that encapsulate system calls, and that don't have a C++ equivalent.
- Each value passed by copy instead of reference or by pointer must be justified. Otherwise, you'll lose points.
- Each non `const` value passed as parameter must be justified. Otherwise, you'll lose points.
- Each member function or method that does not modify the current instance and which is not `const` must be justified. Otherwise, you'll lose points.
- There is no `C++` norm. However if a code is reckoned to be unreadable or dirty, this code will be penalized. Be serious please!
- Keep an eye on the project instructions. They can change with time!
- We are interested in the quality of our materials. Please, if you find any spelling mistake, grammatical error etc. please contact us at lab.koala@epitech.net so that we can do a proper correction.

11 Turn-in

You must turn in you project in the repository provided by `Epitech`. Thw repository name is `cpp_arcade`.

Your repository will be cloned at the exact hour of the end of the project, intranet `Epitech` being the reference.

Only the code from your repository will be graded during the oral defense.

Good luck!