**Purpose:**

* To store the data about video games purchased by a user.
* Also, to maintain a list of the user’s favorite games that supports addition and removal of items.

**Requirements:**

* The database must store all the following items about the game
  + Name
  + Platform (e.g., Xbox-360, PS3, DS, or Wii)
  + Video game category
  + Publisher
  + Rating
  + Purchase date (MM/DD/YYYY)
  + Purchase price (in dollars)
* All the data must be validated before input:
  + After removing leading and trailing blank characters (<SPACE> or <TAB>), a video game name or a publisher name must not be empty.
  + Platform must be one of the following: Xbox-360, PS3, DS, or Wii.
  + Video game category must be one of the following: Action, Education, Fighting, Music/Party, Puzzle/Cards, Roleplaying, Shooter, Simulation, Sports, Strategy, or Other.
  + Rating must be one of the following: EC, E, E10+, T, M, AO, RP, PG, PG13, R, or NC17.
  + Purchase date must be in the MM/DD/YYYY format and be a valid date. The month must be 2-digit long (01 through 12). The day must be 2-digit long (01 through 31). The year must be 4-digit long and has a value of 1970 or greater. (Note: it may be a bit tricky to figure out that 02/29/2010 is not a valid date.)
  + Purchase price can be an integer or an integer followed by a period then followed by one or two digits. It must not contain commas or have more than 2 digits after the decimal point.
  + The program should be able to handle invalid input such as accidental key presses
* The favorites list must support addition and removal of items

**Classes:**

Game.cpp-

* Purpose: To store the game data for each game
* Member data: Game information
* Member methods: getters, setters and print function to print the data.

**Global data/functions:**

* No global variables
* The main database.cpp file contains all the functions that are outside the class
* This includes:
  + int main(char, char\*\*)
  + void print(vector<Game> &games, vector<int> &favs, int ch)
  + void add(vector<Game> &games)
  + void add(vector<Game> &games, vector<int> &favs)
  + void rem(vector<Game> &games, vector<int> &favs)

**High level architecture:**

* The main function creates a vector games to contain Game objects.
* A vector favs is also created to contain the indices of favorites in games
* User is then prompted for a choice.
* If the user choses to add a game
  + add(vector<Game> &games) is called
  + The function creates a new game object temp
  + The function then accepts and validates all the game data
  + The setter methods of temp are then used to set the data of temp
  + temp is then pushed back to games
* If the uses choses to add a game to the favorites list
  + add(vector<Game> &games, vector<int> &favs)is called
  + The function searches through games to check if the game is in the database.
  + If the game is found, the function then searches through favs to check that the game has not already been added
  + If the game is in games but not in favs, the game’s index is pushed back to favs
* If the user choses to remove a game from the favorites list
  + rem(vector<Game> &games, vector<int> &favs)is called
  + The function searches through favs to check if the game is in the database
  + If the game is found, it is removed
  + If it is not found, an error message is printed
* If the user choses to print all the games
  + print(vector<Game> &games, vector<int> &favs, int ch) is called
  + If the database is empty, an error message is printed.
  + Otherwise, the function traverses games, calling each object’s print() method
* If the user choses to print the favorites list
  + print(vector<Game> &games, vector<int> &favs, int ch) is called
  + If the list is empty, an error message is printed.
  + The function traverses the favs, calling the print() method of the objects in games that the elements in favs point to.
* The program then repeats until the user choses to quit.

**User Interface:**

* The program uses a text-based interface.
* The program is menu-driven, so the user is repeatedly prompted for a choice until he chooses to quit.
* The program reads input from the keyboard and directs output to the monitor.

**Test Cases:**

* The program has been tested for a variety of test cases
* Input validation is tested by entering invalid inputs
* This covers all the corner cases such as badly formatted input and input that is out of bounds
* Specific examples include:
  + Badly formatted inputs(invalid characters etc)
  + Invalid dates(eg. 01/32/1992, 02/29/2100)
  + Invalid prices (.42, 65.433, -8)
  + Invalid categories/platforms/ratings

**Function Documentation:**

**Game:**

* All the setters accept data of the type to be encoded and return a 0 for success or a -1 for failure.
* The getName() function accepts no data but returns the objects name as a string
* The print function neither returns nor accepts data. It prints all the data of the calling object.

**Database:**

* int main(char, char\*\*)
  + Accepts arguments from the command line
  + Returns an int to the operating system to indicate success or failure
* void print(vector<Game> &games, vector<int> &favs, int ch)
  + Accepts the addresses of the games and favorites vectors
  + Prints the data
  + Does not return data
* void add(vector<Game> &games)
  + Overloaded function
  + Accepts address of the games vector
  + Accepts user input to create objects
* void add(vector<Game> &games, vector<int> &favs)
  + Overloaded function
  + Accepts the addresses of the games and favorites vectors
  + Adds a game to the favorites list
  + Doesn’t return data
* void rem(vector<Game> &games, vector<int> &favs)
  + Accepts the addresses of the games and favorites vectors
  + Removes games from the favorites list