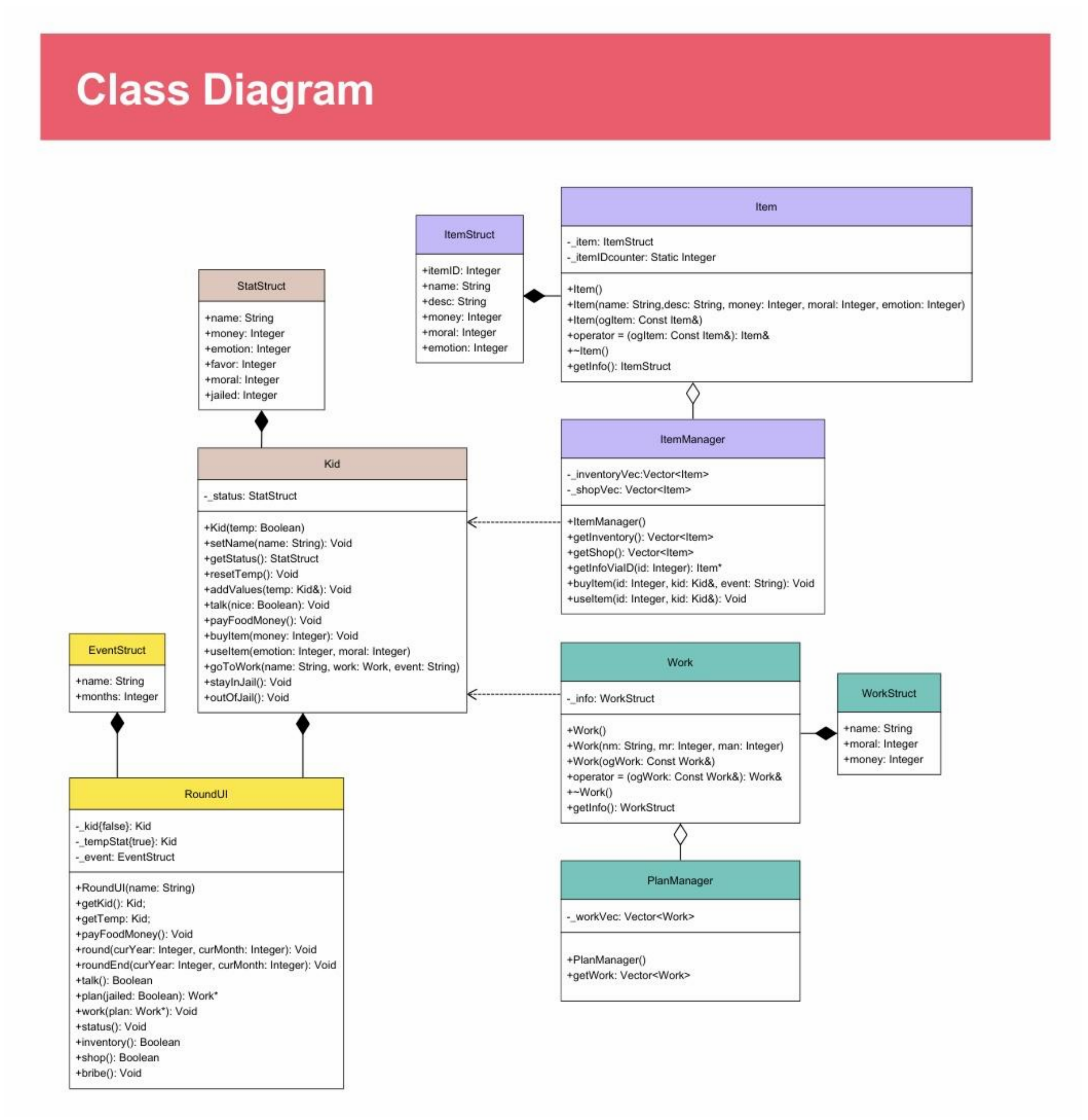

Above is the UML class diagram.
I'll introduce all my source code starting from MainGame.cpp, down to RoundUI.h & RoundUI.cpp, and then all the little ones.

. MainGame.cpp

Contains the main() function for the game.

This is a simple game of raising a child.
In main(), after letting the player decide the kid's name and their own referral, the game starts.
The game lasts for 2 years (24 months, each month a round) before the ending.

There are 9 endings in total, determined based on the kid's money, moral, and favor values.
endCoding() decides the code for calling each ending.
ending() prints different endings based on the code from endCode().

. RoundUI.h & RoundUI.cpp

Defines class RoundUI.
It is the class capable of setting up each round.
It contains 3 private members: _kid, _tempStat, and _event.
_kid and _tempStat are responsible for storing the kid's total values and monthly temporary values, while _event is for recording the current event.

The functions are all public.
The self-explanatory ones will be skipped for the whole document.

The constructor takes the kid's name to generate _kid.
round() sets up the UI and runs each round. It also randomly decides if any event should start.
roundEnd() follows suit, printing out the changes in _tempStat. It stands for the change in values that happened after the kid finished all their plans for the month. During the function, the values in _tempStat will be merged into _kid, and _tempStat will be reset.

There are many different commands for the player in the UI printed by round().

1. talk(): Allows the player to talk with the kid, either nicely or harshly. Calls a function in class Kid to affect the favor value.
2. plan(): To plan the month for the kid. When the plan is finished, run work()
3. work(): The player cannot directly access this function. Runs a for-loop, calls a function in class Kid to make the kid follow each planned work in the time slots. It calls _tempStat and not _kid, for every changed value should be stored in _tempStat before roundEnd(). Events may affect the work salary.
4. status(): Show _kid's stats.
5. inventory(): Show the current inventory. If there are items, the player can check their stats and let the kid use items. When the kid uses items, it calls useItem() function in class ItemManager (which calls useItem() in class Kid for value manipulating)
6. shop(): Show the shop. Events may affect the prices. When buying items, it calls a function buyItem() in class ItemManager (which calls buyItem() in class Kid for value manipulating) If the gambling event is going on, there will be a gambling option.
7. bribe(): Only appears on the UI when the kid is in jail. If the player agrees to pay a certain amount of money (which varies based on how long the kid is jailed), the kid can be immediately released.

. AsciiSprite.h

Stores the kid's sprites.

. Status.h & Status.cpp

Defines class Kid.
It is the class for storing the kid's different variables.
The only variable in class Kid - _status, of type StatStruct - is private. The variables are packed up into StatStruct, including name, money, emotion, favor, moral, and jailed. jailed is an integer instead of a boolean, because it also tracks the remaining months the kid should be jailed for.

The functions are all public.

1. addValues() is only called in roundEnd() in RoundUI.cpp, and is for adding up the temporary values in _tempStat to the main values in _kid.
2. talk() takes a boolean to check if the player is talking nicely or not. Then, a random number between 1 to 5 is chosen. The final affected favor value is decided based on the two factors.
3. buyItem() and useItem() in class Kid are simply used for value manipulating. For the item system, check out Item.h & Item.cpp.
4. goToWork() function takes the kid's name, the assigned work, and the name of the event going on. It prints out the kid's changed values during the work. Do note that the "work" here doesn't always count as working. Resting and staying in jail are stored as works too. For more information, check Work.h & Work.cpp.
   Works with negative moral values are bad, thus risky. The lower the moral value is, the riskier the work is. If the kid got caught, they will be marked as jailed, and will be sent to jail starting from the following month.
5. stayInJail() and outOfJail() simply manage jailed. outOfJail() subtracts the money for bribing, as it is only called when the player chooses to bribe. It is not called if the kid is released after being jailed for enough months.

Some formats are defined below, they are for printing the values when running RoundUI::status() (statResult) and RoundUI::roundEnd() (monthResult).

. Item.h & Item.cpp

Defines classes Item and ItemManager.

Class Item is for storing item data.
There are two variables, both private - _itemIDcounter and _item.
_itemIDcounter is only used for managing itemID in _item, automatically adding 1 whenever a new item is added.
_item is of type ItemStruct. In ItemStruct, the defined variables include itemID, name, desc (description), money, moral, and emotion.

The functions are all public.
There are no other functions rather than the basics.

Class ItemManager is for managing inventory and the shop.
Its two vector variables _inventoryVec and _shopVec are private.

The functions are all public.

1. getInfoViaID() takes the id of the item, and returns the corresponding pointer if it exists in the shop.
2. buyItem() takes the id of the item, a reference of the kid, and the current event's name. If the item exists, it adds the item to _inventoryVec, and calls buyItem() in class Kid for price paying. The event's name is used when calling Kid::buyItem, for it may affect the price.

Note that _shopVec shows a menu instead of a storage.
Buying an item does not result in it being removed from
_shopVec.

3. useItem() takes the id of the item and a reference of the kid. To change the values of the kid accordingly, Kid::useItem is called. Then, the item is removed from _inventoryVec.

. Work.h & Work.cpp

Defines classes Work and PlanManager.

Class Work is for storing work data.
There is only one variable - _info, which is private.
_info is of type WorkStruct. In WorkStruct, the defined variables include name, moral and money.

The functions are all public.
There are no other functions rather than the basics.

Class PlanManager is for managing the list of possible works.
Its vector variable _workVec is private.

The functions are all public.
There are no other functions rather than the basics.
This class serves no use other than recording the list of available works. Do note that resting and staying in jail count as works and are both added into _workVec.