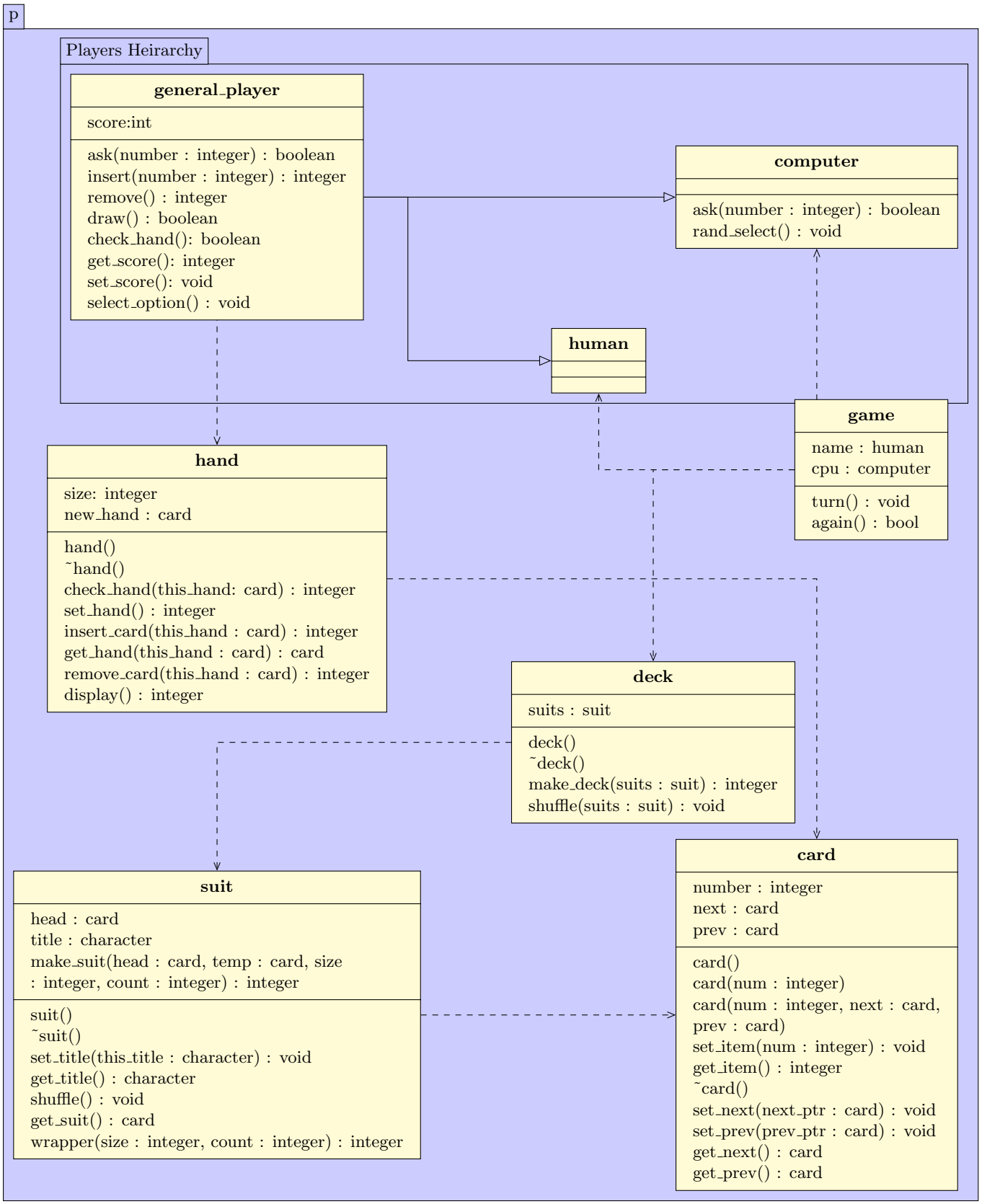
p
Players Heirarchy
general_player
score:int
ask(number : integer) : boolean
insert(number : integer) : integer
remove() : integer
draw() : boolean
check_hand(): boolean
get_score(): integer
set_score(): void
select_option() : void
computer
ask(number : integer) : boolean
rand_select() : void
human
game
name : human
cpu : computer
turn() : void
again() : bool
hand
size: integer
new_hand : card
hand()
˜hand()
check_hand(this_hand: card) : integer
set_hand() : integer
insert_card(this_hand : card) : integer
get_hand(this_hand : card) : card
remove_card(this_hand : card) : integer
display() : integer
deck
suits : suit
deck()
˜deck()
make_deck(suits : suit) : integer
shuffle(suits : suit) : void
suit
head : card
title : character
make_suit(head : card, temp : card, size
: integer, count : integer) : integer
suit()
˜suit()
set_title(this_title : character) : void
get_title() : character
shuffle() : void
get_suit() : card
wrapper(size : integer, count : integer) : integer
card
number : integer
next : card
prev : card
card()
card(num : integer)
card(num : integer, next : card,
prev : card)
set_item(num : integer) : void
get_item() : integer
˜card()
set_next(next_ptr : card) : void
set_prev(prev_ptr : card) : void
get_next() : card
get_prev() : card

Debugging and Design

gdb and continuous design modification was required when working through this assignment. gdb helped in debugging memory leaks and semantically checking the program for behavioral correctness. Running the program and running the stack backtrace helped in finding the memory leaks and segmentation faults. Further stepping through main to find the semantic errors in the program was considerably more difficult in order to find the break points to set. A particular area where gdb was used was when the players exchanged cards and checked for a book. In the first attempt this produced segmentation faults for passing a null pointer and freeing. gdb was used to step into the exchange cards function to find the segmentation fault. In particular the commands used were the step and next functions. The code breaking was found with using the debugger in the graphical code interface using the tui command. The memory leaks in the program were checked using valgrind, since gdb has no functionality to do so.
The design aspect of the code followed by adding functions to the header files and changing the design for a more efficient and less code approach. Initially the design was set up so that the hand class was an array of pointers to the deck and that the deck class when shuffled would give the address of the first card to the player and a flag was set in the card class to check when the card was in use. The original draw function worked using the sending an address for the hand to use and point to. When the hand was discarded the flag was updated and the pointer was set to NULL, this approach works well, however counting the pairs in the array for a book was not an accomplishable task with the pointer array, or not something that had an immediate and simple solution. The most easiest solution was designing a hash maps to find all the pairs of cards in an array. Since the linked structure was already present the hash map was the easiest structure to implement to finish this assignment in a timely manner. The collision resolution technique used for the hash map was chaining the linked list and the hash function used the modulo of the table size. Another design aspect difficulty was shuffling the deck, a standard solution was used to solve this problem, in particular the fisher-yates shuffle, with the doubly linked list, this is a fairly standard solution.

1

How well did the data structures perform for the assigned application?
The data strucutres performed the operations of the assigned application well with regards to efficieny and object oriented design. The design of the classes were by the objects that are instrinsic to the assignment for example, the hand, deck and cards are all objects that required in the game. In this program design the classes each had their own responsibilities, for example the card class was strcutured as a node representing a linear linked list. The suit class designs a linear linked structure based on the nodes of the card. The hand class reduces the complexity of the operations of counting and checking for books using the hash table. With regard to the player classes from a top-down design point of view, when the general player has a turn during game play, they check thier hands for a "book" which is a suit (4 of the same cards). If they do have a book the 4 cards are placed onto a discarded **deleted**. The player then asks the computer for a card if the computer does have a card the card is **removed** from the computers hand and **inserted** into the players hand the computer must give up all of the same numbered cards. If the players actions (any action) returns a true value then the players turn continues. The player continues to ask, check for a book in the hand until either returns a false value. If a false value is found, then the computer takes over, which is a **circular** playing structure. The deck class **has a** relationship with the suit class (encapsulation). Each suit class contains a value which is selected from a character array type of suit which is a diamond, spades, clubs, hearts. The suit array has a pointer to doubly linked list which contains the 10 cards from each suit. The Jack, King and Queen are omitted from the suits.

2

Would a different data structure work better? If so, which one and why.
A data structure that would better accomplish this task with fewer lines of code and more efficiently at compile time would be 4 by 13 matrix representing the cards and a hash table representing the hand. This allows for fast access and processing power of the array and run time efficieny is increased. The other data

form continued on next page...

structure that works well in the context of the problem is a stack to represent the deck which is a first in first out structure.

3

What was efficient about this program design?
The efficiency of the design lies in the object oriented design of the program. The responsibilities are characterized as encapsulation or inheritance. Each class working with data structures do implement the remove, insert and other data structures related functions. The inheritance design is also very useful when code needs to be reused. The heirarchical relationships were between the deck and game class and between the player classes. The relationships served to simplify the code and reduce the complexity in design which contributed towards the efficiency of the design.

4

What was not efficient about this program design?
The inefficiency of the design is prevalent in the relationship between hand and the player class in that there are many wrapper functions present so that one class is doing the job of another class. This includes getting cards from the player, removing from the hand and other function specified in the header file while this design is a working solution it is not the best one that comes to mind.

5

What would you do differently?
The relationship between the hand and the player class should be modified to reflect a more object oriented design and also extend the design to reflect a bigger deck. The deck used in the game is set to a limited size.