Table of Contents

C++

Code Checklist

Go through entire list before turning in the exam!

- Every method that doesn't change the object should be const
- Objects passed as parameters should be passed as const and/or as reference whenever possible.
- When allocating memory to a collection of objects as part of a loop, make sure to try and catch so that there is no memory leaks even if an exception occurs mid loop

```
} catch (const std::exception&) {
  delete[] elems;
  throw;
```

- When making a function that returns an object's field by reference, make a const and a non const version of that function

```
const T& operator()(int i, int j) const {
  return elems[index(i, j)];
}
T& operator()(int i, int j) {
  return elems[index(i, j)];
}
```

- When returning a reference, make sure it isn't a local variable of the function or it will get destroyed
- Make destructors of parent objects virtual

- Check the validity of arguments and to do it like this unless stated otherwise
```
if(maxFuel < 0)
    throw invalid_argument("Invalid Max Fuel");
```
- A getter method for an object should look like this
```
const Position &GetPosition() const {return position;}
```
- When working with pointers or smart pointers use arrows instead of dots, note that "this" is a pointer.
- When working with unique_ptr use move(ptr) whenever changing ownership
- Make functions that overload 'override' and functions that get overloaded 'virtual'
- Don't use the word 'new' when defining an object directly and not as a pointer
- Don't forget to write template<class T> above a template class and above any template function (not within the class declaration)
- Make sure you iterate over a reference of objects inside the container (unless the object is a shared_ptr)
- If a conversion doesn't make sense for a 1 parameter constructor, it should probably be marked as explicit
- Instead of duplicating code you override, access parent version of the function using Parent::func()
- When using a class within a template class from outside, qualify it as typename. For example: typename Set<T>::Iterator it
- Minimize code duplication

Reference Code

Exceptions

Throw an exception with a what string either with

```
throw runtime_error("Cannot divide by zero")
```

or with

```
class DivideByZero: public exception{
    const char * what() const noexcept override {
        return "Cannot divide by zero";
    }
};
throw DivideByZero();
```

Unique PTR

```
unique_ptr<int> a (new int(10));
unique_ptr<int> b (move(a));
```

Clone

```
Parent* clone() const override {

        return new Child(*this);

}
```

Basic Operator Usage

2/13/23, 10:34 PM c:\...\snippets\Operators.h

```
class Num{
    int val;
public:
    //When non explicit, is usable as a conversion from int to Num
    Num(int param):
        val(param) {}
    //Big three - Copy Constructor/Assignment Operator/Destructor
    Num(const Num& other) = default;
    Num& operator=(const Num& other) = default;
    ~Num() = default;
    //Increment/assignment +=, -=, *=, /=
    //++, -- are the same but without parameters
    Num& operator+=(int param){
        val += param;
        return *this;
    }
    //Can be used for any binary operator (=,-,*,/, %,=,!=,<,>,<=,>=)
    //Use when desired symmetrical effect for implicit conversion
    friend Num operator+(const Num& param1, const Num& param2){
        return param1.val + param2.val; //Implicit conversion
    }
    //Same as previous but non symmetrical
    Num operator+(const Num& param){
        return this->val + param.val;
    }
    //Postfix unary operators (++, --)
    //Unary minus is the same but without parameters
    Num operator++(int){
        Num temp = *this;
        val++;
        return temp;
    }
    //Function operator num()
    bool operator()(){
        return val != 0;
    }
    //Printing operator needs to be friend
    friend ostream& operator<<(ostream& os, const Num& num){
        os << num.val;
        return os;
    }
    //Conversion from Num to float
    operator float() const{
        return (float)val;
    }
};
```

Basic Container Template

2/13/23, 10:59 PM c:\...\snippets\Containers.h

```
template<class T>
class Stack{
public:
    class Iterator;
    Stack();
    void push(const T& param);
    void pop();
    int& front() const;
    const Iterator& begin() const { return *first;}
    const Iterator& end() const {return *last;}
private:
    shared_ptr<Iterator> first, last;
};
template<class T>
class Stack<T>::Iterator{
    friend class Stack;
    shared_ptr<T> val;
    shared_ptr<Iterator> next;
    Iterator(shared_ptr<T> val, shared_ptr<Iterator> next): val(val), next(next) {}
public:
    int& operator*(){ return *val; }
    const int& operator*() const{ return *val; }
    Stack::Iterator& operator++(){
        if (next == nullptr)
            throw exception();
        *this = *next;
        return *this;
    }
    bool operator!=(const Iterator& it) const{ return !(next == it.next); }
};
template<class T>
Stack<T>::Stack(): first(new Iterator(nullptr, nullptr)), last(first) {}
template<class T>
void Stack<T>::push(const T& param){
    shared_ptr<T> val = shared_ptr<T> (new T(param));
    first = shared_ptr<Iterator>(new Iterator(val, first));
}
template<class T>
void Stack<T>::pop(){ ++(*first); }
template<class T>
int& Stack<T>::front() const{ return *(*first); }
```

Other operators/template stuff

```
template<typename Condition>
func(Condition condition){} //Template can be used for a function object,
keyword typename and class are interchangeable her
```

```
template<int N>
class NSizedObject {} //Makes an object with a parameter that's defined at
compilation time
```

```
typedef PassengerCar<2> PersonalCar; //Makes a template function with a
certain parameter into a non template function
```

```
template<class T>
const T* Set<T>::Iterator::operator->() const {
  assert(index >= 0 && index < set->getSize());
  return &(set->data[index]);
}
```

the only change from operator*()

```
const int& Array::operator[](int index) const {
  assert(index >= 0 && index < size);
  return data[index];
}
```

```
int& Array::operator[](int index) {
  assert(index >= 0 && index < size);
  return data[index];
}
```

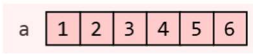
STL interfaces

array<T,size>

fixed-size array

#include <array>

```
std::array<int,6> a {1,2,3,4,5,6};
cout << a.size();     // 6
cout << a[2];         // 3
a[0] = 7;             // 1st element ⇒ 7
```



contiguous memory; random access; fast linear traversal

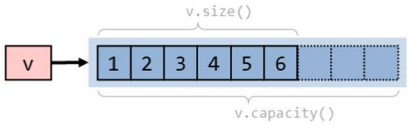
vector<T>

dynamic array

C++'s "default" container

#include <vector>

```
std::vector<int> v {1,2,3,4,5,6};
v.reserve(9);
cout << v.capacity();      // 9
cout << v.size();          // 6
v.push_back(7);            // appends '7'
v.insert(v.begin(), 0);    // prepends '0'
v.pop_back();              // removes last
v.erase(v.begin()+2);      // removes 3rd
v.resize(20, 0);           // size ⇒ 20
```



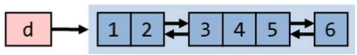
contiguous memory; random access;
fast linear traversal; fast insertion/deletion at the ends

deque<T>

double-ended queue

#include <deque>

```
std::deque<int> d {1,2,3,4,5,6};
// same operations as vector
// plus fast growth/deletion at front
d.push_front(-1);   // prepends '-1'
d.pop_front();      // removes 1st
```



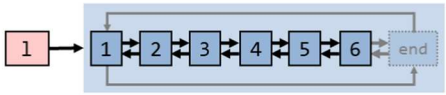
fast insertion/deletion at both ends

list<T>

doubly-linked list

#include <list>

```
std::list<int> l {1,5,6};
std::list<int> k {2,3,4};
// O(1) splice of k into l:
l.splice(l.begin()+1, std::move(k))
// some special member function algorithms:
l.reverse();
l.sort();
```



fast splicing; many operations without copy/move of elements

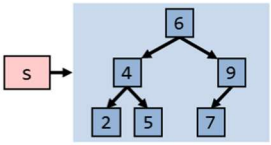
set<Key>

unique, ordered keys

multiset<K>

(non-unique) ordered keys

```
std::set<int> s;
s.insert(7); …
s.insert(5);
auto i = s.find(7);  // → iterator
if(i != s.end())     // found?
  cout << *i;        // 7
if(s.contains(7)) {…}   C++20
```



usually implemented
as balanced binary tree
(red-black tree)

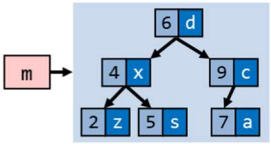
map<Key,Value>

unique key → value-pairs; ordered by keys

multimap<K,V>

(non-unique) key → value-pairs, ordered by keys

```
std::map<int,char> m;
m.insert({7,'a'});  …
m[4] = 'x';          // insert 4 → x
auto i = s.find(7);  // → iterator
if(i != s.end())     // found?
  cout << i->first   // 7
       << i->second; // a
if(s.contains(7)) {…}   C++20
```



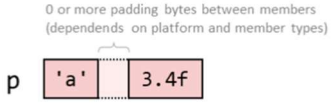
usually implemented
as balanced binary tree
(red-black tree)

pair<A,B>

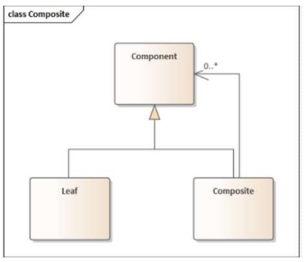
#include <utility>

contains *two* values
of same or different type

```
std::pair<char,float> p;
p.first  = 'a';
p.second = 3.4f;
char  x = std::get<0>(p);
float y = std::get<1>(p);
```

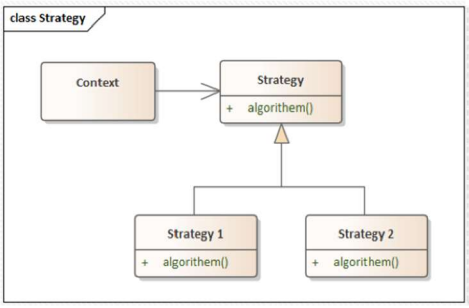


Design Patterns



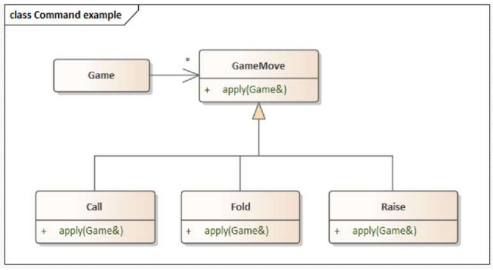
- Composite – Defining a hierarchy using an object that contains a collection of its parent object

- Strategy – Defining an object's behavior using an external class, the behavior object should be passed as a parameter in the constructor




```
class Behaviour {
  virtual void makeMove(Game& game) = 0;
};

class AggressiveBehaviour : public Behaviour {
public:
  void makeMove(Game game&) override;
};

class Player {
 Behaviour *m_behaviour;

public:
  void setBehaviour(Behaviour *behaviour) { m_behaviour = behaviour; }
  void makeMove(Game& game) { m_behaviour->makeMove(game); }
};
```

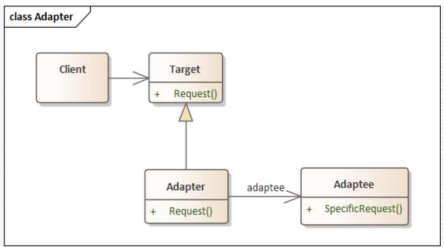
- Command – Defining an action as an external class




```
class Game {
  vector<GameMove*> moves;
public:
  void makeMove(GameMove* move) {
    cout << "Applied: " << move;
    moves.push_back(move);
    move->apply(*this);
    nextPlayer();
  }

  void undo() {
    GameMove* m = moves.pop_back();
    if (m != nullptr) {
      m->undo();
    }
  }
};
```

```
class GameMove {
public:
  virtual void apply(Game&) const = 0;
  virtual void undo(Game&) const = 0;
};

class Fold : public GameMove {
public:
  void apply(Game& game) const override;
  void undo(Game&) const override;
};

class Call : public GameMove {
public:
  void apply(Game& game) const override;
  void undo(Game&) const override;
};

class Raise : public GameMove {
  int amount;
public:
  void apply(Game& game) const override;
  void undo(Game&) const override;
};
```

- Adapter – Adapting object1 to object2 by inheriting object2 and containing object1

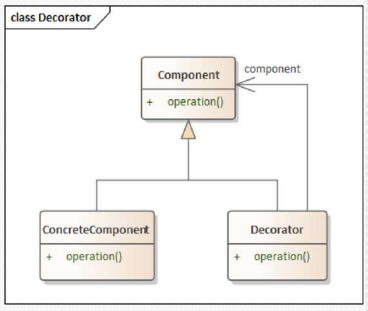



```
namespace Poker {
  class PlayerAdapter : public GuiLib::Agent {
    Player* m_player;
  public:
    PlayerAdapter(Player* player) : m_player(player) { }
    string getName() const override {
      return m_player->getName();
    }
    GuiLib::Cards getCards() const override {
      return convertHandToCards(m_player->getHand());
    }
  };
}
```

Explicit namespaces can improve readability

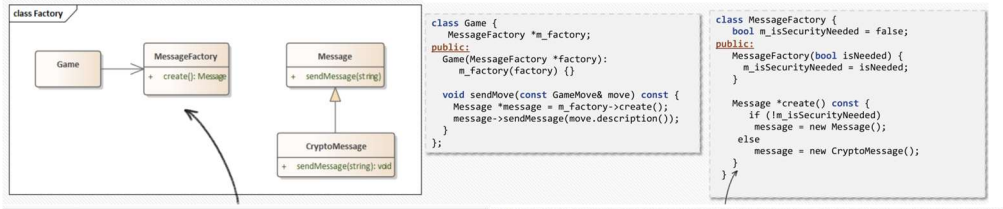
Plain old data conversion (not an adapter)

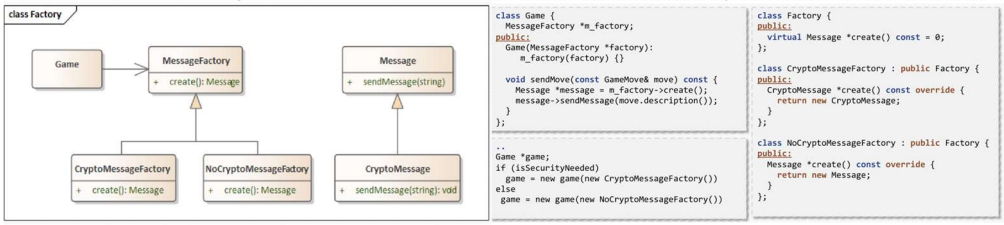
- Decorator – Defining a hierarchy of (usually visual) elements nested in one another. Unlike Composite, only contains 1 instance of parent object.



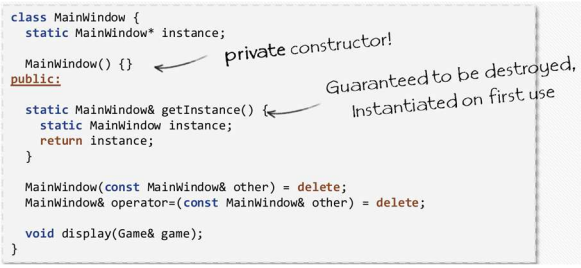

```
class GuiWindow {
public:
  virtual void display(Game& game);
};

class AnimatingWindow : public GuiWindow {
  GuiWindow* window;
public:
  void display(Game& game) override {
    window->display(game);
    animate(window, game);
  }
};

class StatisticsWindow : public GuiWindow {
  GuiWindow* window;
public:
  void display(Game& game) override {
    window->display(game);
    addStatistics(window, game);
  }
};
```

- Factory – Delegating the action of creating objects to a separate “factory” object. Normal factory is done using a Boolean or enum and an abstract factory is done using inheritance (Implementation is similar to strategy)





- Singleton – A class for an object of which only one copy can ever be created and can be accessed from everywhere



Python

Functions

Function	What it does	Example	Return
Basic			
print	prints all parameters separated by spaces	print(“one”, “two”, 3) print(“one”, “two”, 3, sep=’;’)	prints “one two 3” prints “one;two;3”
type	Returns the type of variable	Type(2.5)	<class ‘float’>
Numbers			
+,-,*,%	Standard operators, Same as C	2*3	6
**	To the power of	2 ** 3	8
/	Real division	05-Feb	2.5
//	C division	5 // 2	2
Strings			
+, append	Concatenate strings	s = ‘wonder’+”ful”	wonderful
[] substring	Returns a substring. Can use start, end and spacing as parameters	s[:6] s[-3:] s[1::2]	wonder ful odru
len	Returns string length	len(s)	9
startswith	Returns true if string starts with parameter	s.startswith(‘wo’)	TRUE
strip	Gets rid of spaces in start and end	‘ word ‘.strip()	word
Islower, isupper	Returns true if lowercase/uppercase	s.Islower()	TRUE
lower,upper	Converts to lowercase/uppercase	s.upper()	WONDERFUL
Isdigit,isalpha	Returns true if string of numbers/letters	s.isalpha()	TRUE
find	Returns index of first instance of parameter	s.find(‘de’)	3
replace	Replaces all instances of one substring with another	s.replace(‘w, ‘W’)	Wonderful
split	Splits a string into a list of strings based on a separator of choice, default is any whitespace (including \n).	‘a,b,c\n’.split(‘,’) ‘a b c\n’.split()	[‘a’, ‘b’, ‘c\n’] [‘a’, ‘b’, ‘c’]
isspace	Returns true if string of spaces/tabs/newlines	s.isspace()	FALSE
format	Insert variables into the middle of a string	"Person {} got {}".format(4,100)	Person 4 got 100

Lists [],append,+,+=, len work similar to strings Tuples are identical to lists except they are declared with () and are immutable			
remove	Removes first instance of parameter	l.remove(58)	[‘wow’, [4, 1, 1, 3]]
pop	Removes value at parameter’s index and returns it	l.pop(0)	‘wow’ l = [58, [4, 1, 1, 3]]
copy	Copies the list (non basic types copied by reference)	a = l.copy()	
deepcopy	Copies the list completely (not by reference)	From copy import deepcopy a = l.deepcopy()	
==	Returns true if same value	a==L	TRUE
is	Returns true if identical reference	a is L	FALSE
min, max	Returns minimum/maximum value	max([1,2,3])	3
Count	Returns number of times value appears	[1, 1, 2,3].count(1)	2
index	Returns index of first appearance of parameter	[1,2,3,4].index(2)	1
sort	Sorts list in ascending order,	[5, 3, 8, -1, 6].sort()	[-1, 3, 5, 6, 8]
reverse	Reverses list order	[1,4,3,2].reverse()	[2,3,4,1]
in	Returns true if exists in list	4 in [1,2,3,4]	TRUE
Dictionaries			
Get value	Get value tied to key	d[1]	one
Set value	Adds key,value pair	d[4] = ‘four’	
For loops			
for element in list:	Itterates on every element in the list	for element in [‘a’,’b’,2]: print(element)	a b 2
for char in string:	Iterates on every char in the string	for char in ‘wow’: print(char)	w o w
for num in range:	Iterates on every number in a given range with a given step	for num in range(8, 3,-2): print(num)	8 6 4

for i,e in enumerate:	Iterates on numbers and elements simultaneously	for I,e in enumerate([7,8,3]): print(I,e)	0 7 1 8 2 3
for k,v in d.items()	iterates on key,value pairs in a dictionary	for k,v in {1: "one", 2: "two}: print(k, v)	1 one 2 two
for e1,e2 in zip:	Iterates on 2 lists simultaneously	for e1,e2 in zip([1,2,3],[4,5,6]): print(e1,e2)	1 4 2 5 3 6
List Comprehension	Create a list using itteration in one line	[elem**2 for elem in [1,2,3,'w'] if type(elem) is int]	[1,4,9]
Dictionary Comprehension	Create a dictionary using itteration in one line	{elem: elem**2 for elem in [1,2,3,'w'] if type(elem) is int}	{1: 1, 2: 4, 3: 9}

Files	
with open(file_name, mode) as f:	Opens file and automatically close it at the end of indent ‘w’ – write, ‘r’ – read, ‘a’ - append
f.read()	Returns a string containing the entire file
f.readlines()	Returns a list where every element is a string containing a line from the file. Every line except the last one ends with \n which can be removed using line.strip(‘\n’)
f.write(string)	Writes a string to the file
Json	
dict = json.load(f)	Read JSON file as dictionary
json.dump(dict, f)	Dumps a dictionary to a JSON file
os	
os.listdir(path)	Returns list of files in given directory
os.path.join(path1, path2)	Joins 2 path strings in a way suitable to the os
os.path.dirname(path), os.path.basename(path)	Returns name of path directory/name of path file
os.mkdir(path)	Makes a folder with the path
os.path.isfile(path), os.path.isdir(path)	Returns true if path is file/directory

Reference Code

2/18/23, 4:02 PM c:\...\hello_world\Reference.py

```
class Dog:
    def __init__(self, name): #constructor
        if type(name) != str:
            raise ValueError("Invalid name") #throw
        self.name = name
    def bark(self):
        print(self.name, 'says Woof')

class Chiwawa(Dog): #inheritance
    def bark(self):
         print(self.name, 'says Weef')

try:
    for dog in [Dog("Woofer"), Chiwawa("Chi")]:
        dog.bark()
    dog2 = Dog(1)
except ValueError as err: #catch
    print(err)
finally:
    print("This is the end")

data = { #Nested dictionary comprehension
    'capacity': bag.capacity,
    'items': [{'name': item.name, 'weight': item.weight} for item in bag.items]
}

with open('test.txt', 'r') as f:
    for l in f:
        l = l.split()
        do something with l

with open('test.txt', 'w') as f:
    for e in list:
        out = do something with e
        f.write(out+'\n')

with open('test.json', 'r') as f:
    dict = json.load(f)

with open('test.json', 'w') as f:
    json.dump(dict, f)
```

Checklist

- Make sure to remove the \n at the end of a line of text when reading from a file. Alternatively if you use parameter-less .split() it isn't an issue
- When writing to a file make sure to add \n to separate lines
- Don't forget to make your functions return the thing they are supposed to return.
- Note that +, += operators on lists don't work exactly like append. += works like append when adding number or a string to the list, but works like extend when adding a list. So if you want to add a list into a list as a list and not add its elements you should use append.