CS 162 Worksheet 7

Read and trace the following code, and answer the following questions.

```
#include <iostream>
#include <string>

using namespace std;

class Garage
{
private:
  int num_car;  //num of cars
  string* cars; //an array of car names
public:
  Garage();
  Garage(int);

  //Big three
  ~Garage();
  Garage(const Garage&);
  Garage& operator=(const Garage&);

  //other member functions
  Garage fun1();
  void fun2(Garage);
  Garage& fun3(Garage&);
};

Garage::Garage() {
  cout << "Garage()" << endl;
  this->num_car = 0;
  this->cars = NULL;
}

Garage::Garage(int n) {
  cout << "Garage(int)" << endl;
  this->num_car = n;
  this->cars = new string [n];
}

//implementation of the big three
Garage::~Garage() {
  cout << "~Garage()" << endl;
  if (this->cars != NULL){
    delete [] this->cars;
    this->cars = NULL;
  }
}

Garage::Garage(const Garage& g){
  cout << "CC" << endl;
  this->num_car = g.num_car;
  this->cars = new string [this->num_car];
  for (int i = 0; i < this->num_car; ++i)
  {
    this->cars[i] = g.cars[i];
  }
}

Garage& Garage::operator=(const Garage& g){
  cout << "AOO" << endl;
  if (this == &g)
    return *this;

  if (this->cars != NULL)
    delete [] this->cars;

  this->num_car = g.num_car;
  this->cars = new string [this->num_car];
  for (int i = 0; i < this->num_car; ++i)
  {
    this->cars[i] = g.cars[i];
  }

  return *this;
}

Garage Garage::fun1() {
  cout << "1" << endl;
  Garage temp;
  //...some code
  return temp;
}

void Garage::fun2(Garage g) {
  cout << "2" << endl;
  //...some code
}

Garage& Garage::fun3 (Garage& g) {
  cout << "3" << endl;
  //...some code
  return g;
}


int main()
{
  Garage g1;
  Garage g2(5);
  Garage g3 = g2;
  g1 = g2;

  g1 = g2.fun1();

  g3.fun2(g1);

  g1.fun3(g2);

  return 0;
}
```

1. For line 12 and 13, which one is the default constructor, and which one is the non-default constructor?

2. Label the big three from line 16-18. Options: Copy constructor, Assignment Operator Overload, Destructor

3. Explain the difference between shallow copy and deep copy.

4. Which parts within the big three functions implement the deep copy?

5. What is the purpose of line 59-60?

6. What is the purpose of line 62-63?

7. What is the output of line 96-99?

8. What is the output of line 101?

9. What is the output of line 103?

10. What is the output of line 105?

11. What is the output after line 108?

12. When would each of the big three functions be called?

13. From your answers above, what is the most efficient way to pass an object into a function?

For each memory-related error reported by valgrind below, explain what caused the error and how to fix it:

1. **main.cpp:**

```cpp
#include <string>

using namespace std;

string* create_array(int size) {
        return new string[size];
}

int main() {
        int num_strings = 10;
        string* arr = create_array(num_strings);

        for (int i = 0; i < num_strings; i++) {
                arr[i] = "Hello, world!";
        }
}
```

Valgrind output:

```
guyera@flip3:1$ g++ -g main.cpp
guyera@flip3:1$ valgrind --leak-check=full ./a.out
==1281051== Memcheck, a memory error detector
==1281051== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==1281051== Using Valgrind-3.19.0 and LibVEX; rerun with -h for copyright info
==1281051== Command: ./a.out
==1281051==
==1281051==
==1281051== HEAP SUMMARY:
==1281051==     in use at exit: 328 bytes in 1 blocks
==1281051==   total heap usage: 2 allocs, 1 frees, 73,032 bytes allocated
==1281051==
==1281051== 328 bytes in 1 blocks are definitely lost in loss record 1 of 1
==1281051==    at 0x484622F: operator new[](unsigned long) (vg_replace_malloc.c:640)
==1281051==    by 0x401186: create_array[abi:cxx11](int) (main.cpp:6)
==1281051==    by 0x4011DC: main (main.cpp:11)
==1281051==
==1281051== LEAK SUMMARY:
==1281051==    definitely lost: 328 bytes in 1 blocks
==1281051==    indirectly lost: 0 bytes in 0 blocks
==1281051==      possibly lost: 0 bytes in 0 blocks
==1281051==    still reachable: 0 bytes in 0 blocks
==1281051==         suppressed: 0 bytes in 0 blocks
==1281051==
==1281051== For lists of detected and suppressed errors, rerun with: -s
==1281051== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

2.

main.cpp:

```
 1 #include <string>
 2
 3 using namespace std;
 4
 5 string* create_array(int size) {
 6         return new string[size];
 7 }
 8
 9 int main() {
10         int num_strings = 10;
11         string* arr = create_array(num_strings);
12
13         for (int i = 0; i < num_strings; i++) {
14                 arr[i] = "Hello, world!";
15         }
16
17         string *copy = arr;
18
19         delete [] arr;
20         delete [] copy;
21 }
```

Regular output:

```
guyera@flip3:2$ g++ main.cpp
guyera@flip3:2$ ./a.out
Segmentation fault (core dumped)
```

Valgrind output (this produces many errors, so here's just the first error and the heap summary):

```
guyera@flip3:2$ g++ -g main.cpp
guyera@flip3:2$ valgrind --leak-check=full ./a.out
==1288540== Memcheck, a memory error detector
==1288540== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==1288540== Using Valgrind-3.19.0 and LibVEX; rerun with -h for copyright info
==1288540== Command: ./a.out
==1288540==
==1288540== Invalid read of size 8
==1288540==    at 0x4012A9: main (main.cpp:20)
==1288540==  Address 0x4dacc80 is 0 bytes inside a block of size 328 free'd
==1288540==    at 0x48488CF: operator delete[](void*, unsigned long) (vg_replace_malloc.c:1116)
==1288540==    by 0x401299: main (main.cpp:19)
==1288540==  Block was alloc'd at
==1288540==    at 0x484622F: operator new[](unsigned long) (vg_replace_malloc.c:640)
==1288540==    by 0x4011A6: create_array[abi:cxx11](int) (main.cpp:6)
==1288540==    by 0x4011FD: main (main.cpp:11)
==1288540== HEAP SUMMARY:
==1288540==     in use at exit: 0 bytes in 0 blocks
==1288540==   total heap usage: 2 allocs, 3 frees, 73,032 bytes allocated
==1288540==
==1288540== All heap blocks were freed -- no leaks are possible
==1288540==
==1288540== For lists of detected and suppressed errors, rerun with: -s
==1288540== ERROR SUMMARY: 13 errors from 4 contexts (suppressed: 0 from 0)
```

3.

main.cpp:

```
 1 #include <iostream>
 2
 3 using namespace std;
 4
 5 int create_int() {
 6         int x = 0;
 7         return x;
 8 }
 9
10 int main() {
11         int x;
12         if (x == 0) {
13                 cout << "X is 0!" << endl;
14         }
15 }
```

Valgrind output:

```
guyera@flip3:3$ g++ -g main.cpp
guyera@flip3:3$ valgrind --leak-check=full ./a.out
==1292483== Memcheck, a memory error detector
==1292483== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==1292483== Using Valgrind-3.19.0 and LibVEX; rerun with -h for copyright info
==1292483== Command: ./a.out
==1292483==
==1292483== Conditional jump or move depends on uninitialised value(s)
==1292483==    at 0x401192: main (main.cpp:12)
==1292483==
X is 0!
==1292483==
==1292483== HEAP SUMMARY:
==1292483==     in use at exit: 0 bytes in 0 blocks
==1292483==   total heap usage: 2 allocs, 2 frees, 73,728 bytes allocated
==1292483==
==1292483== All heap blocks were freed -- no leaks are possible
==1292483==
==1292483== Use --track-origins=yes to see where uninitialised values come from
==1292483== For lists of detected and suppressed errors, rerun with: -s
==1292483== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```