**Free Store**

**Writer:** Sayak Haldar

**1. Which is used to allocate and deallocate storage for objects during the execution?**

a) Stack

b) Heap

c) Freestore

d) None of the mentioned

**Answer: c**

Explanation: Free store is a pool of memory available for you to allocate and deallocate storage for objects during the execution of your program.

**2. Which operators are used in the free store?**

a) new

b) delete

c) both new & delete

d) none of the mentioned

**Answer: c**

Explanation: new and delete operators is used to allocate and deallocate the memory for the program.

**3. What type of class member is operator new?**

a) static

b) dynamic

c) const

d) smart

**Answer: a**

So, if you are planning to overload operator new, make the function static.

**4. What is the output of this program?**

#include <new>

#include <iostream>

using namespace std;

struct A

{

virtual ~A() { };

void operator delete(void\* p)

{

cout << "A :: operator delete" << endl;

}

};

struct B : A

{

void operator delete(void\* p)

{

cout << "B :: operator delete" << endl;

}

};

int main()

{

A\* ap = new B;

delete ap;

}

a) A::operator delete

b) B::operator delete

c) Both A::operator delete & B::operator delete

d) None of the mentioned

**Answer: b**

Explanation: In this program, We are passing the value to the B, So we are printing B::operator delete.

Because, delete operator will be called the object type which is allocated dynamically.

**5. What is the output of this program?**

#include <iostream>

using namespace std;

struct A

{

virtual ~A()

{

cout << "~A()" << endl;

}

void operator delete[](void\* p, size\_t)

{

cout << "A :: operator delete[]" << endl;

delete [] p;

}

};

struct B : A

{

void operator delete[](void\* p, size\_t)

{

cout << "B :: operator delete[]" << endl;

delete [] p;

}

};

int main()

{

A\* bp = new B[3];

delete[] bp;

};

a) ~A()

b) A :: operator delete[].

c) B :: operator delete[].

d) Warning

View Answer

**Answer: d**

Explanation: In this program, the behavior of the statement delete[] bp is undefined.

Because, it the overloaded version of delete, we pass void pointer. And deleting void pointer invokes undefined behaviour. Hence, warning will be thrown informing the same.

**6.What is the output of this program?**

#include <cstdlib>

#include <iostream>

using namespace std;

class X

{

public:

void\* operator new(size\_t sz) throw (const char\*)

{

void\* p = malloc(sz);

if (p == 0)

throw "malloc() failed";

return p;

}

void operator delete(void\* p)

{

cout << "X :: operator delete(void\*)" << endl;

free(p);

}

};

class Y

{

int filler[100];

public:

void operator delete(void\* p, size\_t sz) throw (const char\*)

{

cout << "Freeing " << sz << " bytes" << endl;

free(p);

};

};

int main()

{

X\* ptr = new X;

delete ptr;

Y\* yptr = new Y;

delete yptr;

}

a) X::operator delete(void\*)

b) Freeing 400 bytes

c) Depends on the compiler

d) Both X::operator delete(void\*) & Depends on the compiler

**Answer: d**

X :: operator delete(void\*)

Freeing 400 bytes

**Explanation:** The memory value allocated for the program depends on compiler ony.

Let’s modify the program a bit, and print the step by step printing:

Now, you will see, it’s highly compiler depended.

#include <cstdlib>

#include <iostream>

using namespace std;

class X

{

public:

void\* operator new(size\_t sz) throw (const char\*)

{

void\* p = malloc(sz);

if (p == 0)

throw "malloc() failed";

cout<<"The memory which was allocated for class X's object is: "<<sz<<endl;

return p;

}

void operator delete(void\* p)

{

cout << "X :: operator delete(void\*)" << endl;

free(p);

}

void operator delete(void \*p,size\_t sz) throw (const char\*)

{

cout<<"X :: Freeing "<<sz<<" bytes"<<endl;

}

};

class Y

{

int filler[100];

public:

//Now, it does not overload the overator new

void operator delete(void \*p)

{

cout<<"Y:: operator delete(void\*)"<<endl;

free(p);

}

void operator delete(void\* p, size\_t sz) throw (const char\*)

{

cout << "Y:: Freeing " << sz << " bytes" << endl;

};

};

int main()

{

X\* ptr = new X;

delete ptr;

Y\* yptr = new Y;

delete yptr;

return 0;

}

**Now, output is drastically changed:**

The memory which was allocated for class X's object is: 1

X :: operator delete(void\*)

Y:: operator delete(void\*)

**7.What is the output of this program?**

#include <iostream>

#include <new>

#include <cstdlib>

using namespace std;

const int bsize = 512;

int \*pa;

bool allocate = true;

void get\_memory()

{

cerr << "free store exhausted" << endl;

delete [] pa;

allocate = false;

}

void eat\_memory(int size)

{

int \*p = new int[size];

if (allocate)

eat\_memory(size);

else

cerr << "free store addr = " << p << endl;

}

int main()

{

set\_new\_handler(get\_memory);

pa = new int[bsize];

cerr << "free store addr = " << pa << endl;

eat\_memory(bsize);

return 0;

}

a) Free store addr

b) Error

c) Segmentation fault

d) None of the mentioned

Answer) We keep allocating memory from heap. At some point, we will not able to allocate more memory as free store will be exhausted.

Now, why during first memory allocation, anything is not printed. Because, the function get\_memory which is written as handler function will only be called in erroneous case.

**new\_handler set\_new\_handler (new\_handler new\_p) throw();**

The new-handler function is a function which is called by the default allocation functions (operator new and operator new[]) when they fail to allocate storage.

The new-handler function may try to make more storage available for a new attempt to allocate the storage. If -and only if- the function succeeds in making more storage available, it may return. Otherwise it shall either throw a bad\_alloc exception (or a derived class) or terminate the program (such as by calling abort or exit).

If the new-handler function returns (i.e., it made more storage available), it may be called repeatedly for as long as the allocation function fails to allocate the requested storage, or until the new-handler function does not return or is replaced.

However, we need deeper understanding. That why the handler function does not print any log. Because, it will try to delete some memory when allocation function fails, right?

**At least one log from it, is expected by me. So, I have to study it.**

**8.What must be an operand of operator delete?**

a) Pointer

b) Array

c) Stack

d) None of the mentioned

**Answer: a**

Explanation: The operand of delete must be a pointer returned by new.

**9. How can object be allocated outside the object lifetime?**

a) int

b) float

c) void\*

d) none of the mentioned

**Answer: c**