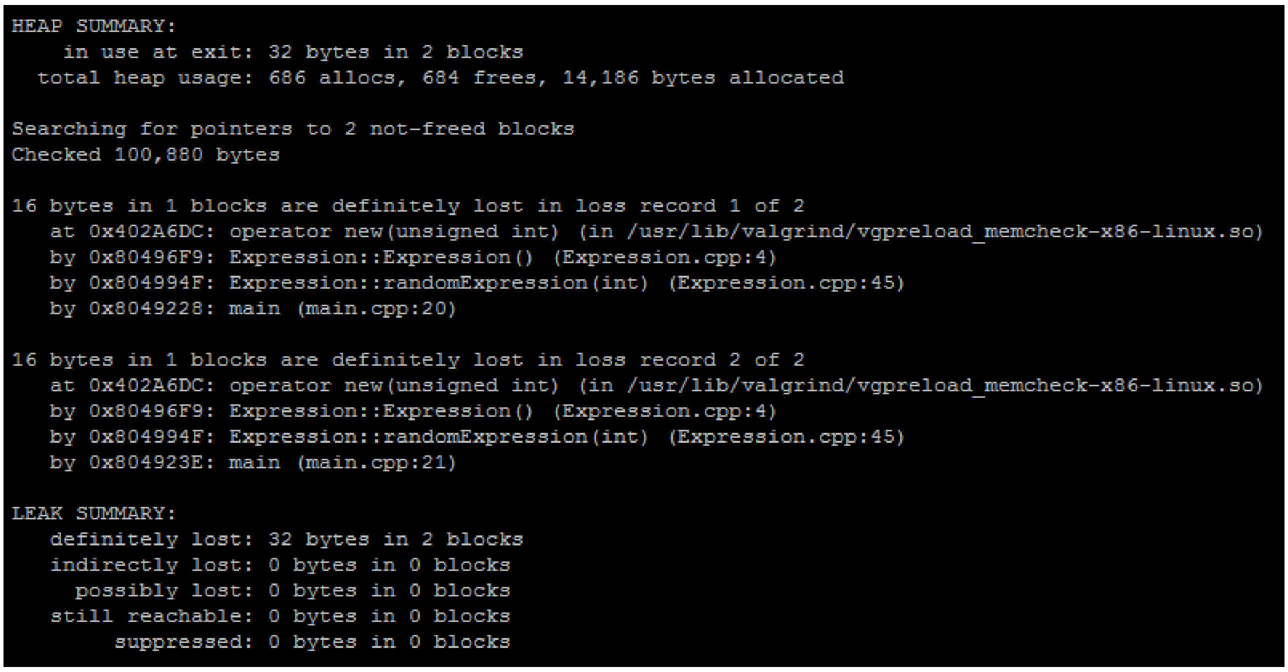
Below is the output of my code before the fix



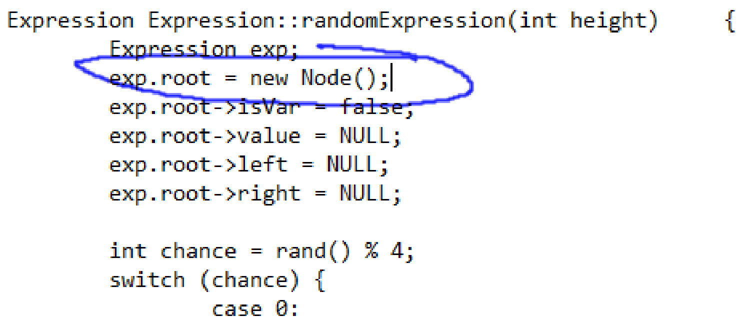
```
HEAP SUMMARY:
    in use at exit: 32 bytes in 2 blocks
  total heap usage: 686 allocs, 684 frees, 14,186 bytes allocated

Searching for pointers to 2 not-freed blocks
Checked 100,880 bytes

16 bytes in 1 blocks are definitely lost in loss record 1 of 2
   at 0x402A6DC: operator new(unsigned int) (in /usr/lib/valgrind/vgpreload_memcheck-x86-linux.so)
   by 0x80496F9: Expression::Expression() (Expression.cpp:4)
   by 0x804994F: Expression::randomExpression(int) (Expression.cpp:45)
   by 0x8049228: main (main.cpp:20)

16 bytes in 1 blocks are definitely lost in loss record 2 of 2
   at 0x402A6DC: operator new(unsigned int) (in /usr/lib/valgrind/vgpreload_memcheck-x86-linux.so)
   by 0x80496F9: Expression::Expression() (Expression.cpp:4)
   by 0x804994F: Expression::randomExpression(int) (Expression.cpp:45)
   by 0x804923E: main (main.cpp:21)

LEAK SUMMARY:
   definitely lost: 32 bytes in 2 blocks
   indirectly lost: 0 bytes in 0 blocks
     possibly lost: 0 bytes in 0 blocks
   still reachable: 0 bytes in 0 blocks
        suppressed: 0 bytes in 0 blocks
```

Why Valgrind was useful

Firstly, I compiled and checked my code using the commands:

```
c++ -std=c++11 -w -g -o Expression main.cpp Expression.cpp
/usr/bin/valgrind -v --leak-check=full ./Expression
```

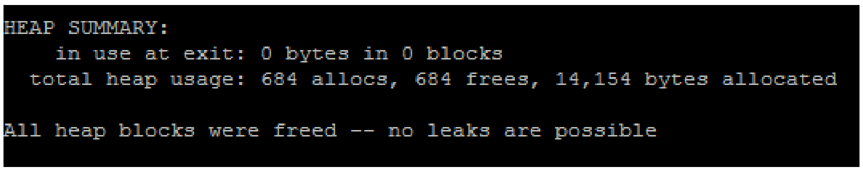
This was extremely helpful because I the `-g` and `--leak-check=full` flags allowed me to see line numbers of where my errors started at. Below is the area of my code that was suspect according to valgrind.



```
Expression Expression::randomExpression(int height)     {
        Expression exp;
        exp.root = new Node();
        exp.root->isVar = false;
        exp.root->value = NULL;
        exp.root->left = NULL;
        exp.root->right = NULL;

        int chance = rand() % 4;
        switch (chance) {
                case 0:
```

I took the liberty of circling the troublesome line. The problem was that the line `Expression exp;` is calling the ctor in which I have the line `root = new Node();`. I then proceed to reassign the root to a different new node as seen above. This caused a memory leak as the first node would be lost and never deallocated. Valgrind led me right to the area where my problem was.

Below is the output of my code after the fix



```
HEAP SUMMARY:
    in use at exit: 0 bytes in 0 blocks
  total heap usage: 684 allocs, 684 frees, 14,154 bytes allocated

All heap blocks were freed -- no leaks are possible
```