1. 1. A memory leak is memory that is being used that is no longer accessible because the pointer to the memory has been set to null and/or there is no pointer that still points to the memory. A dangling pointer is a pointer to memory that has been freed and is no longer allocated. There are roughly opposites of each other. Dangling pointers is a pointer with no defined allocated memory and a memory leak is memory with no pointer.
   2. Dynamically allocating memory without freeing and setting all pointers to null.
   3. Memory leaks are handled by freeing memory before setting the pointer to null.

int main(void){

int \*ptr = malloc(sizeof(int));

ptr = NULL;

}

int main(void){

int \*ptr = malloc(sizeof(int));

free(ptr);

ptr = NULL;

}

Make sure to free the dynamically allocated memory before setting the pointer to null.

* 1. Dangling pointers are handled by setting the pointer that is dangling to null after you free the memory that the pointer was pointing to.

int main(void){

int \*ptr = malloc(sizeof(int));

free(ptr);

printf("%d", \*ptr);

}

int main(void){

int \*ptr = malloc(sizeof(int));

free(ptr);

ptr = NULL;

}

Make sure to set the pointer to the dynamically allocated memory to null after freeing it.

1. Stack memory and heap memory are different in many ways. Stack memory is temporary and is cleared after the program is finished. Stack memory is typically less space than heap memory. The stack is defined before the program is run. Heap memory is allocated while the program is running, also known as dynamically allocated. This causes the possibility of memory leaks because the memory is not automatically deleted when the program finishes, it is up to the programmer. Heap memory can be allocated and deallocated in fragments, while stack cannot. Heap memory is more accessible and more prone to attack.

C Stack

int x = 0;

C Heap

Int \*x = malloc(sizeof(int))

C++ Stack

Int x=0;

C++ Heap

Int \*x = new int [1];

1. 1. No

mu-018056:~ dcjn7z$ make

c++ -std=c++11 -Wall -Werror -ggdb -O0 -c -o ex.o ex.cpp

**ex.cpp:23:20: error: direct base 'A' is inaccessible due to ambiguity:**

**class C -> class B -> class A**

**class C -> class A [-Werror,-Winaccessible-base]**

class C: public B, public A{

**^~~~~~~~**

**ex.cpp:35:4: error: member 'print' found in multiple base classes of different types**

c.print();

**^**

**ex.cpp:17:8: note:** member found by ambiguous name lookup

void print()

**^**

**ex.cpp:8:8: note:** member found by ambiguous name lookup

void print()

**^**

2 errors generated.

make: \*\*\* [ex.o] Error 1

mu-018056:~ dcjn7z$

* 1. The cause of the compiler error here is due to inheritance. The print function is defined in the class A and in the class B, but not in class C. Also, the class B inherits the class A. Therefore, the class B print overrides and is different from the class A print, since the function is defined in both classes. The class C is told to inherit functions from class A and class B. Because the program calls the print function and the class C inherits both the A and B print functions, the compile doesn’t know which print function we are wanting. A solution would be to remove the inheritance of the class A or class B by class C or define a print function for class C explicitly.



mu-018056:~ dcjn7z$ ./main

Changing number to 1

Changing number to 0

Changing number to 1

Changing number to 0

Changing number to 1

* 1. Because the variable number is initialized as static, it is permanently stored while the main program runs. Therefore, every time the program calls the function, it changes and is saved in the main program. It doesn’t get reinitialized at zero or deleted at the end of the function like a normal variable would.