Memory Model of CBMC (1/3)

```c
// cbmc memory-model1.c -trace -trace-show-code -pointer-check
#include<stdio.h>

int main() {
    char *ptr;
    char *m1 = (char *)malloc(12);
    char *m2 = (char *)malloc(24);

    m1[10] = 'x';
    m2[20] = 'y';

    assert(m1+9 != m2); // NOT violated (allocated memory do not overlap)
    assert(m1 != NULL); // NOT violated (memory failure is not modelled)

    // malloc allocates memory filled w/ non-deterministic values
    assert(m1[7] != 'z');  // Violated.
    assert(m2[17] != 'w'); // Violated
}
```

```
[main.assertion.3] line 17 assertion (signed int)m1[(signed long int)7] != 122: FAILURE
[main.assertion.4] line 18 assertion (signed int)m2[(signed long int)17] != 119: FAILURE
```

Memory Model of CBMC (2/3)

```c
// cbmc memory-model2.c -trace -trace-show-code -pointer-check
#include<stdio.h>
char gv='g';
int main() {
    char *ptr;
    char *m1 = (char *)malloc(12);
    char lv ='l';

    m1[10] = 'x';

    // A non-deterministic pointer can point to any memory
    // including dynamic alloc. memory, global/local variables
    assert(ptr != (m1+10));  // Violated
    assert((ptr+1) != &gv);  // Violated
    assert((ptr+2) != &lv);  // Violated
    assert(*(ptr+5) != m1[10]);  // Violated. Also it fails -pointer-check
}
```

```
[main.assertion.1] line 13 assertion ptr != m1 + 10: FAILURE
[main.assertion.2] line 14 assertion ptr + 1 != &gv: FAILURE
[main.assertion.3] line 15 assertion ptr + 2 != &lv: FAILURE
[main.pointer_dereference.8] line 16 dereference failure: pointer uninitialized in ptr[5]: FAILURE
[main.assertion.4] line 16 assertion ptr[5] != m1[10]: FAILURE
```

Memory Model of CBMC (3/3)

```c
// cbmc memory-model3.c -trace -trace-show-code -pointer-check
#include<stdio.h>
int main() {
    char *m1 = (char *)malloc(12);
    char *m2 = (char *)malloc(24);
    int a;

    m1[10] = 'x';
    m2[20] = 'y';

    assert((m1+30)!=(m2+3));   // Not violated.
    assert((m1+a)!= m2);       // Not violated
    assert(*(m1+30)!=*(m2+3)); // Violated. Also, it fails --pointer-check.

    // undefined func does NOT update memory pointed by a pointer parameter
    undef_func1(m1);
    assert(m1[10] == 'x'); // Not violated

    free(m1);
    assert(m1[7] != 'z');   //violated. Also, it fails --pointer-check.
}
```

```
[main.assertion.3] line 14 assertion m1[30] != m2[3]: FAILURE
[main.pointer_dereference.19] line 14 dereference failure: pointer outside dynamic object bounds in m1[30]: FAILURE
[main.pointer_dereference.38] line 21 dereference failure: deallocated dynamic object in m1[7]: FAILURE
[main.assertion.5] line 21 assertion m1[7] != 122: FAILURE
```