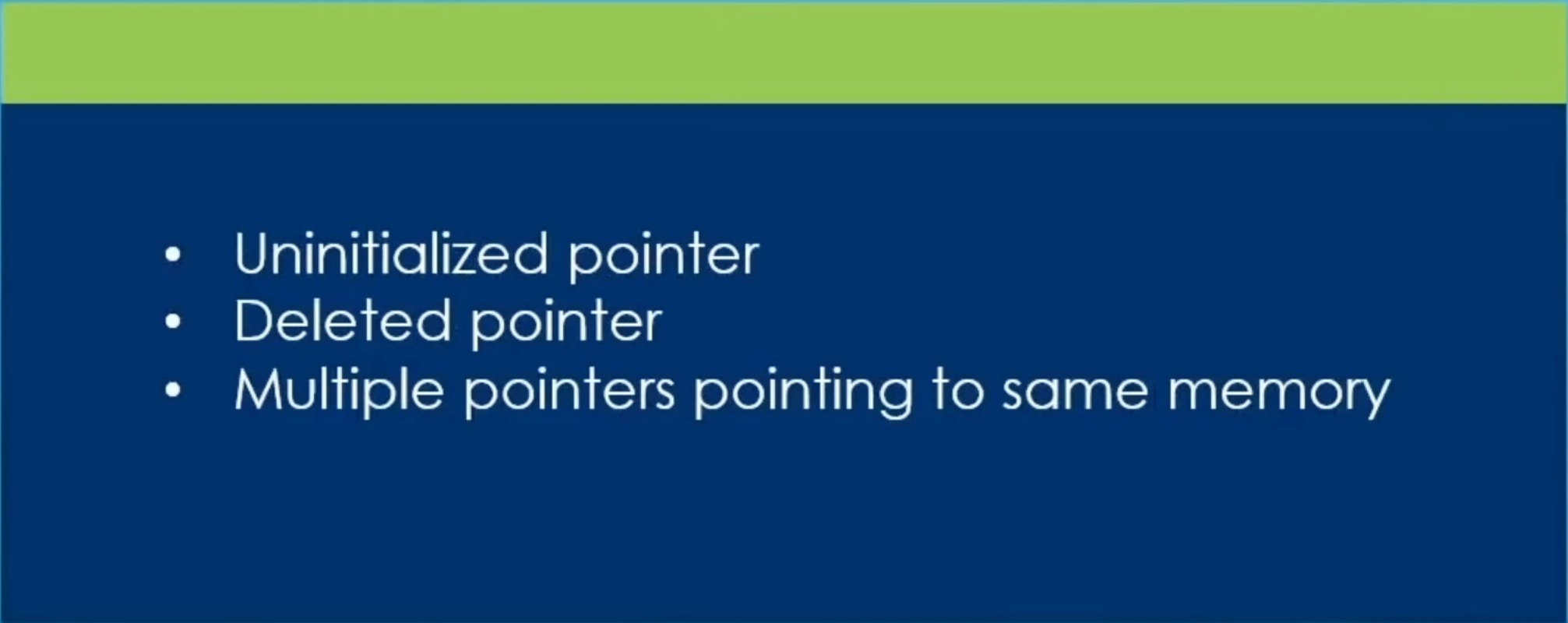



Dangling Pointers

A pointer that doesn't point to valid memory address. Trying to dereference and use a dangling pointer will result in undefined behavior

- 
- Uninitialized pointer
 - Deleted pointer
 - Multiple pointers pointing to same memory
- 

Non initialized pointer

```
//Case1 : Uninitialized pointer  
int * p_number; // Dangling uninitialized pointer  
  
std::cout << std::endl;  
std::cout << "Case 1 : Uninitialized pointer : ." << std::endl;  
std::cout << "p_number : " << p_number << std::endl;  
//std::cout << "*p_number : " << *p_number << std::endl; //CRASH!
```


Deleted pointer

```
//Case 2 : deleted pointer
std::cout << std::endl;
std::cout << "Case 2 : Deleted pointer" << std::endl;
int * p_number1 {new int{67}};

std::cout << "*p_number1 (before delete) : " << *p_number1 << std::endl;

delete p_number1;

std::cout << "*p_number1(after delete) : " << *p_number1 << std::endl;
```

Multiple pointers pointing to same address

```
//Case 3 : Multiple pointers pointing to same address
std::cout << std::endl;
std::cout << "Case 3 : Multiple pointers pointing to same address : " << std::endl;

int *p_number3 {new int{83}};
int *p_number4 {p_number3};

std::cout << "p_number3 - " << p_number3 << " - " << *p_number3 << std::endl;
std::cout << "p_number4 - " << p_number4 << " - " << *p_number4 << std::endl;

//Deleting p_number3
delete p_number3;

//p_number4 points to deleted memory. Dereferencing it will lead to
//undefined behaviour : Crash/ garbage or whatever
std::cout<< "p_number4(after deleting p_number3) - " << p_number4 << " - " << *p_number4 << std::endl;
```

Solutions

- Initialize your pointers
- Reset pointers after delete
- For multiple pointers to same address, make sure the owner pointer is very clear

Solution 1

```
//Solution1 : Initialize your pointers immediately upon declaration
std::cout << std::endl;
std::cout << "Solution 1 : " << std::endl;
int *p_number5{};
int *p_number6{new int{56}};

//Check for nullptr before use
if(p_number6!= nullptr){
    std::cout << "*p_number6 : " << *p_number6 << std::endl;
}
```


Solution 2

```
//Solution 2 :  
//Right after you call delete on a pointer, remember to reset  
//the pointer to nullptr to make it CLEAR it doesn't point anywhere  
  
std::cout << std::endl;  
std::cout << "Solution 2 : " << std::endl;  
int *p_number7{new int{82}};  
  
//Use the pointer however you want  
std::cout << "p_number7 - " << p_number7 << " - " << *p_number7 << std::endl;  
  
delete p_number7;  
p_number7 = nullptr; // Reset the pointer  
  
//Check for nullptr before use  
if(p_number7!= nullptr){  
    std::cout << "*p_number7 : " << *p_number7 << std::endl;  
}
```

Solution 3

```
//Solution 3
//For multiple pointers pointing to the same address , Make sure there is
//one clear pointer (master pointer) that owns the memory ( responsible for releasing when
// necessary) , other pointers should only be able to dereference when the master pointer is valid

int * p_number8 {new int{382}}; // Let's say p_number8 is the master pointer
int * p_number9 {p_number8};

//Dereference the pointers and use them
std::cout << "p_number8 - " << p_number8 << " - " << *p_number8 << std::endl;

if(!(p_number8 == nullptr)){ // Only use slave pointers when master pointer is valid
    std::cout<< "p_number9 - " << p_number9 << " - " << *p_number9 << std::endl;
}

delete p_number8; // Master releases the memory
p_number8 = nullptr;

if(!(p_number8 == nullptr)){ // Only use slave pointers when master pointer is valid
    std::cout<< "p_number9 - " << p_number9 << " - " << *p_number9 << std::endl;
}else{
    std::cerr << "WARNING : Trying to use an invalid pointer" << std::endl;
}
```