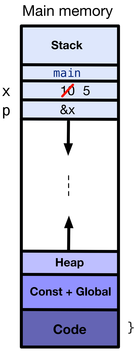
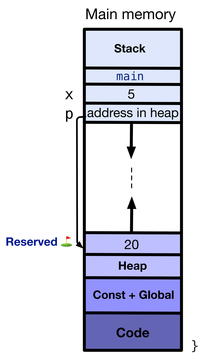



```
#include <iostream>
using namespace std;

int main(){
  int x = 10;
  int* p = NULL;
  p = &x;
  *p = 5;
}
```

Will reveal in the next figure




```
#include <iostream>
using namespace std;

int main(){
  int x = 10;
  int* p = NULL;
  p = &x;
  *p = 5;
  p = new int;
  *p = 20;
  cout << "Value at p: " << *p << endl;
  cout << "Value of x: " << x << endl;
}
```

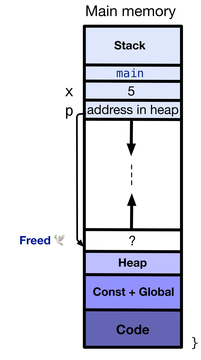
1 Allocate memory for an `int` on the heap

2 return its address and store it in p

prints 20

prints 7

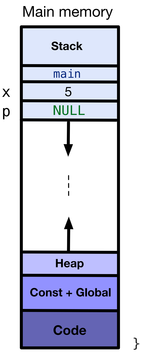
Will reveal in the next figure



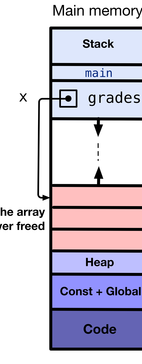

```
#include <iostream>
using namespace std;

int main(){
  int x = 10;
  int* p = NULL;
  p = &x;
  *p = 5;

  p = new int;



  *p = 20;

  cout << "Value at p: " << *p << endl;
  cout << "Value of x: " << x << endl;


  delete p;
}
```

free address stored in p (the heap address)

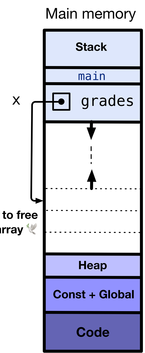
Will reveal in the next figure




```
#include <iostream>
using namespace std;

int main(){
  int x = 10;
  int* p = NULL;
  p = &x;
  *p = 5;

  p = new int;



  *p = 20;

  cout << "Value at p: " << *p << endl;
  cout << "Value of x: " << x << endl;


  delete p;
  p = NULL;
  return 0;
}
```




```
class Student {
  private:
    int* grades;
  public:
    Student(int size) {
      grades = new int[size]; // Dynamically allocate an array
    }
};

int main(void){
  Student x(3); // grades point to an array of 3 integers
  return 0;
} // x goes out of scope, but the array was never freed
```




```
class Student {
  private:
    int* grades;
  public:
    Student(int size) {
      grades = new int[size]; // Dynamically allocate an integer
    }
    ~Student() {
      delete[] grades; // Free the dynamically allocated array
    }
};

int main(void){
  Student x(3); // dynamically allocates an array of 3 integers
  return 0;
} // x goes out of scope, but the array was never freed
```