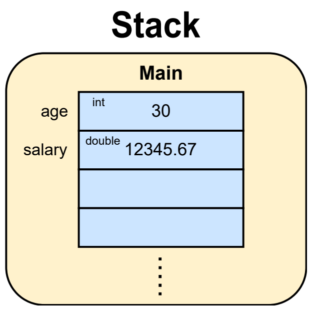
Examples

In this lesson, we'll look at some code and analyze how the Stack and the Heap behave

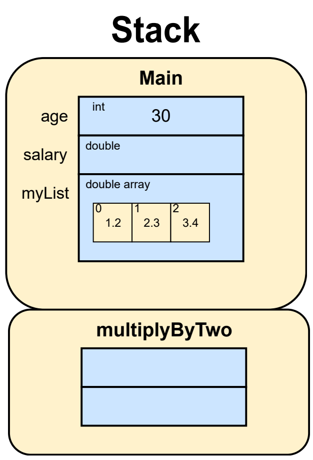
Here is a short program that creates its variables on the **stack**. It looks like the other programs we have seen so far.

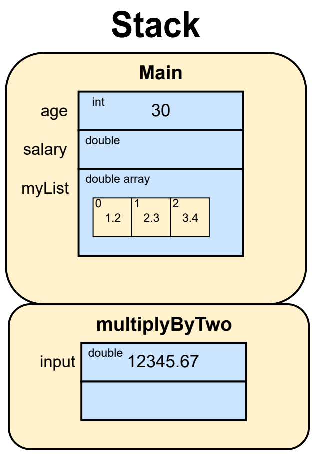
```
#include <stdio.h>

double multiplyByTwo (double input) {
  double twice = input * 2.0;
  return twice;
}

int main (int argc, char *argv[])
{
  int age = 30;
  double salary = 12345.67;
  double myList[3] = {1.2, 2.3, 3.4};

  printf("double your salary is %.3f\n", multiplyByTwo(salary));

  return 0;
}
```

1 of 9

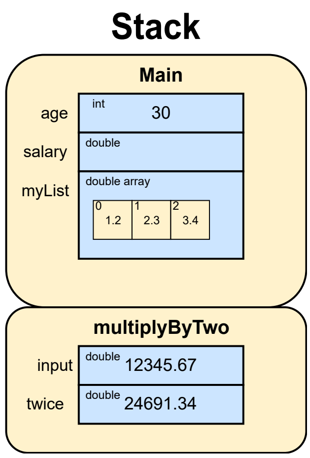

Stack
Main
age
int
30
salary
double
12345.67

Stack

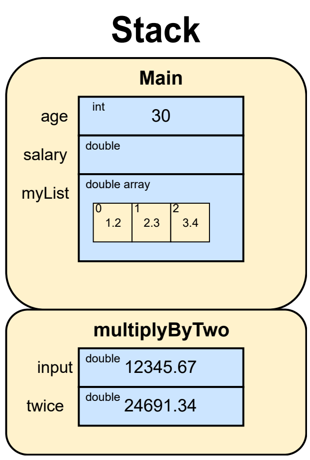

Stack
Main
age
int
30
salary
double
myList
double array
0
1.2
1
2.3
2
3.4
multiplyByTwo



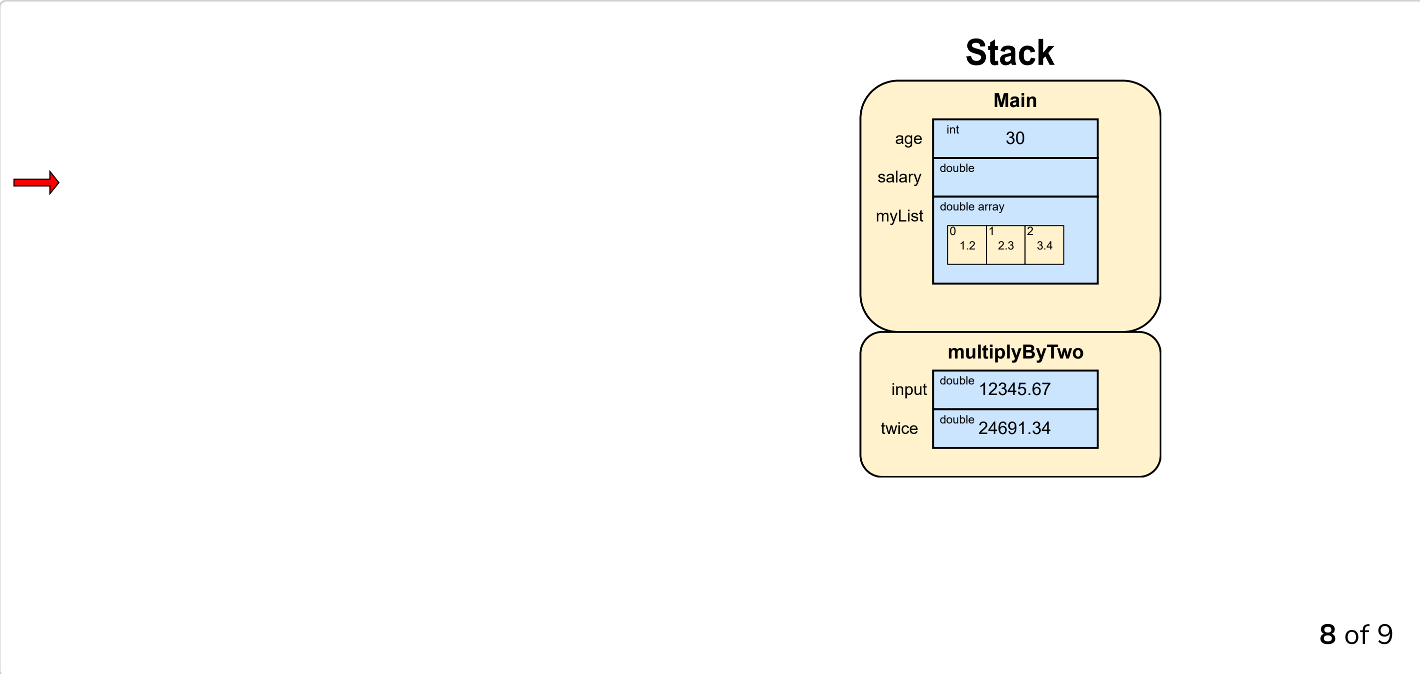
5 of 9

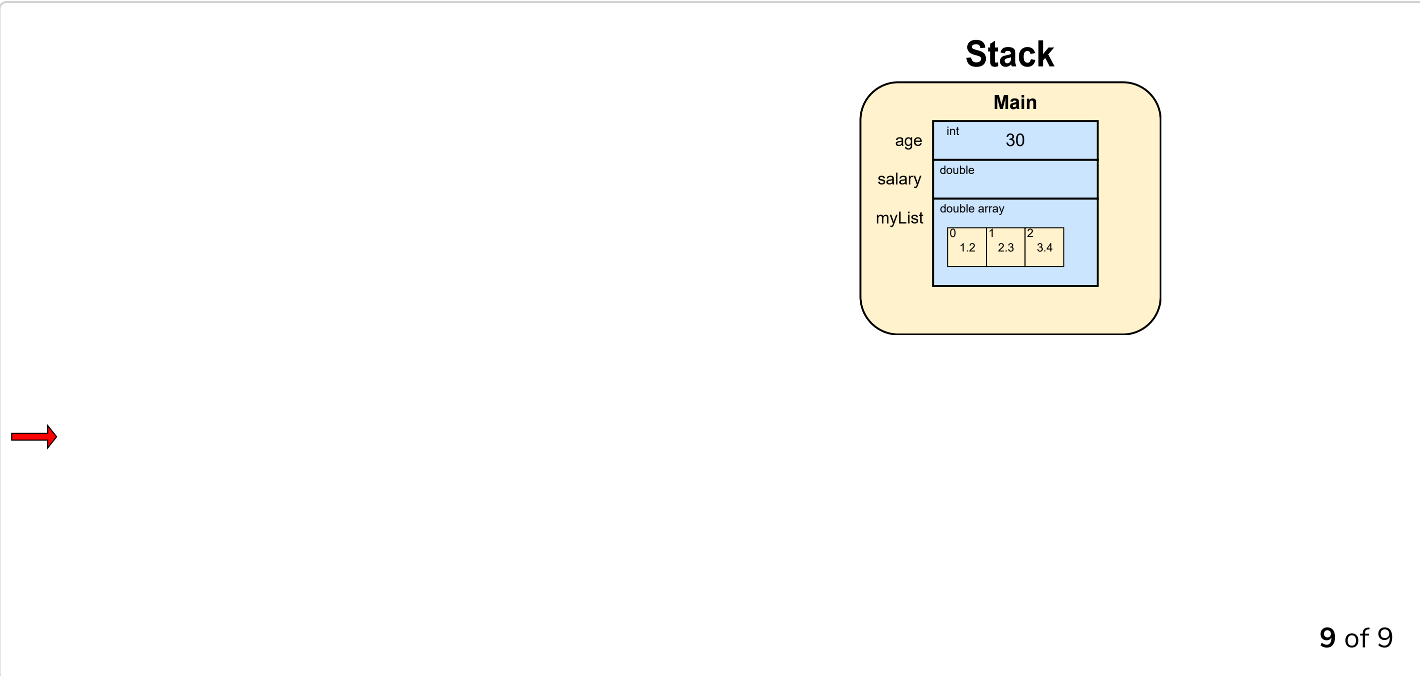


6 of 9



7 of 9



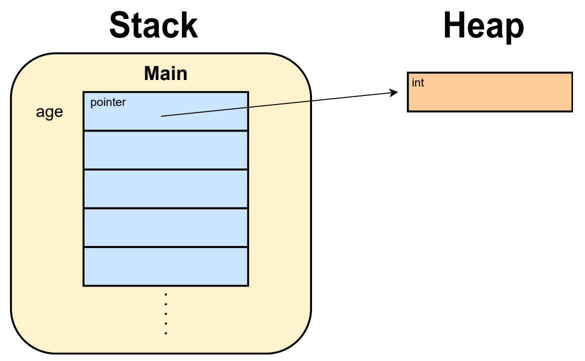


On lines 10, 11 and 12 we declare variables: an `int`, a `double`, and an array of three doubles. These three variables are pushed onto the stack as soon as the `main()` function allocates them. When the `main()` function exits (and the program stops) these variables are popped off of the stack. Similarly, in the function `multiplyByTwo()`, the `twice` variable, which is a `double`, is pushed onto the stack as soon as the `multiplyByTwo()` function allocates it. As soon as the `multiplyByTwo()` function exits, the `twice` variable is popped off the stack and is gone forever.

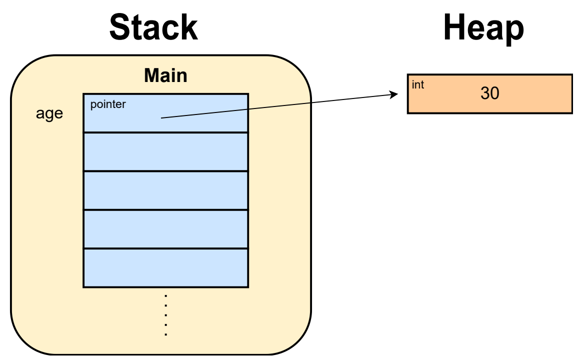
As a side note, there is a way to tell C to keep a stack variable around, even after its creator function exits, and that is to use the `static` keyword when declaring the variable. A variable declared with the `static` keyword thus becomes something like a global variable, but one that is only visible inside the function that created it. It's a strange construction, one that you probably won't need except under very specific circumstances.

Here is another version of this program that allocates all of its variables on the **heap** instead of the stack:

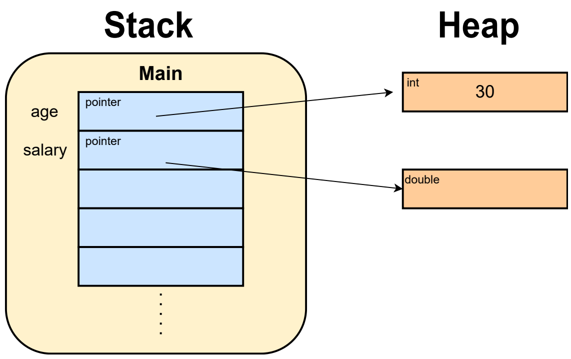
```
#include <stdio.h>
#include <stdlib.h>

double *multiplyByTwo (double *input) {
  double *twice = (double*)malloc(sizeof(double));
  *twice = *input * 2.0;
  return twice;
}

int main (int argc, char *argv[])
{
  int *age = (int*)malloc(sizeof(int));
  *age = 30;
  double *salary = (double*)malloc(sizeof(double));
  *salary = 12345.67;
  double *myList = (double*)malloc(3 * sizeof(double));
  myList[0] = 1.2;
  myList[1] = 2.3;
  myList[2] = 3.4;

  double *twiceSalary = multiplyByTwo(salary);

  printf("double your salary is %.3f\n", *twiceSalary);

  free(age);
  free(salary);
  free(myList);
  free(twiceSalary);

  return 0;
}
```

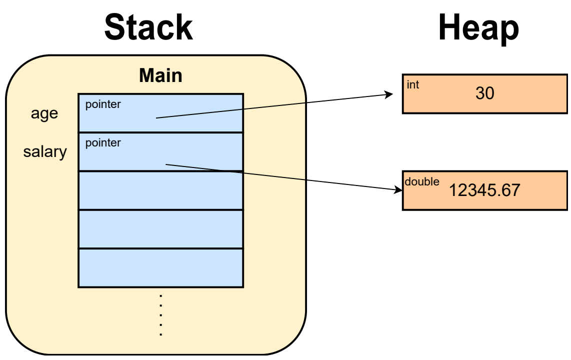


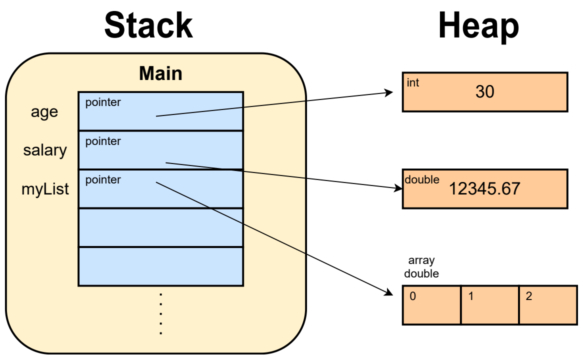
Stack
Heap
Main
age
pointer
int

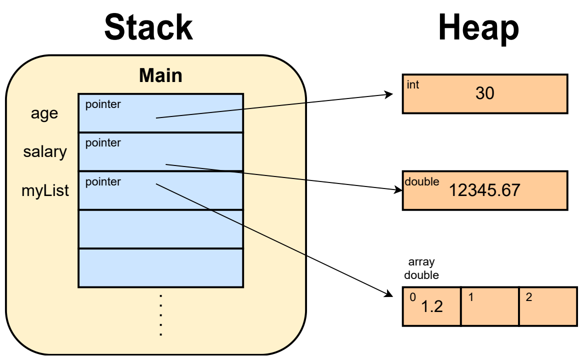


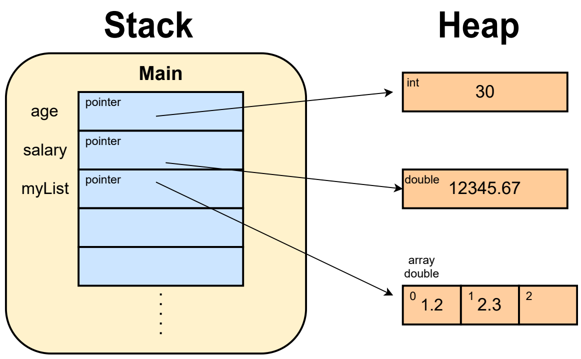
Stack
Heap
Main
age
pointer
int
30

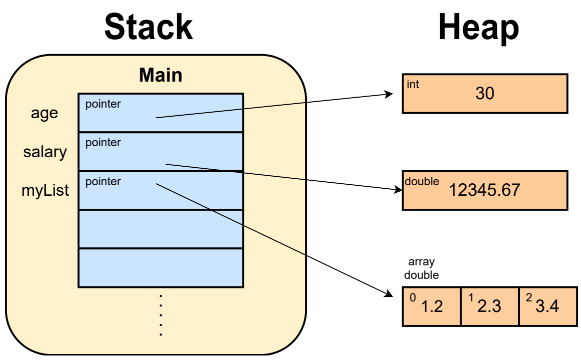

Stack
Heap
Main
age
salary
pointer
pointer
int
30
double

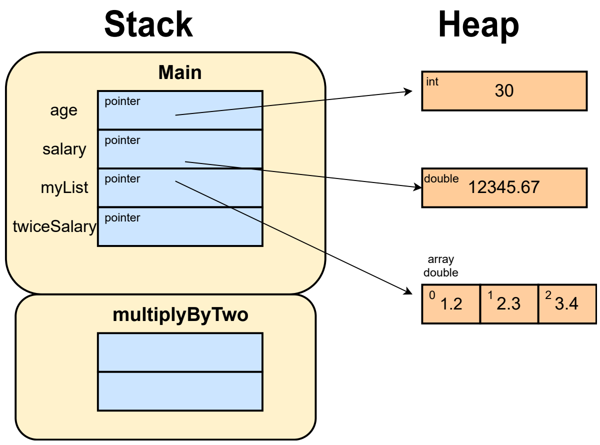

Stack
Heap
Main
age
salary
pointer
pointer
int
30
double
12345.67


Stack
Heap
Main
age
salary
myList
pointer
pointer
pointer
int
30
double
12345.67
array
double
0
1
2

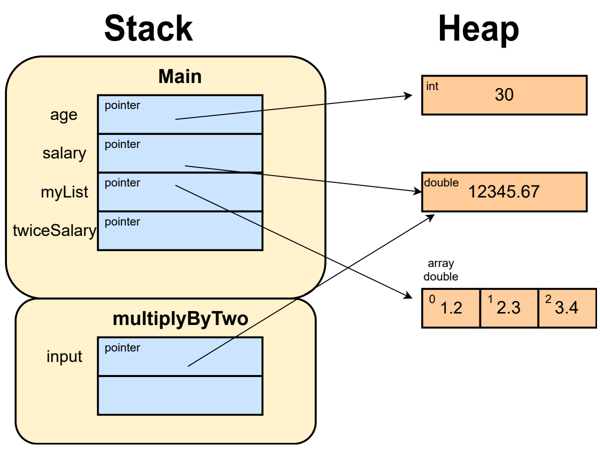

Stack
Heap
Main
age
salary
myList
pointer
pointer
pointer
int
30
double
12345.67
array
double
0
1.2
1
2


Stack
Heap
Main
age
salary
myList
pointer
pointer
pointer
int
30
double
12345.67
array
double
0
1.2
1
2.3
2

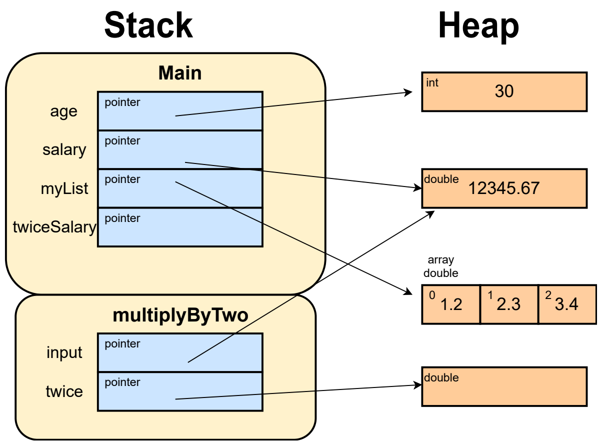

Stack
Heap
Main
age
salary
myList
pointer
pointer
pointer
int
30
double
12345.67
array
double
0
1.2
1
2.3
2
3.4

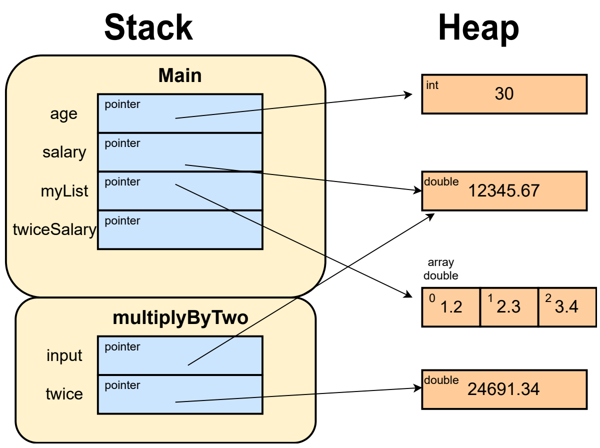


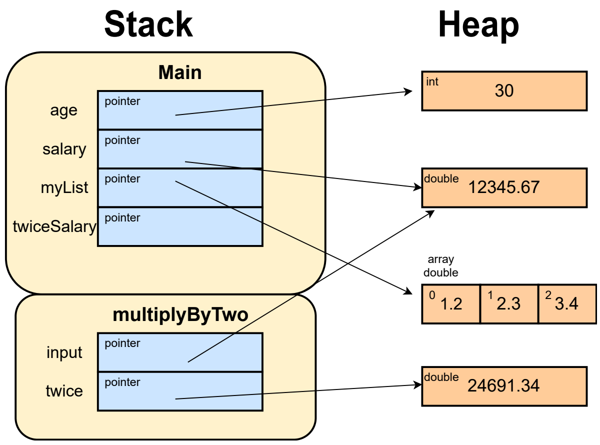
Stack
Heap
Main
age
salary
myList
twiceSalary
pointer
pointer
pointer
pointer
multiplyByTwo
int
30
double
12345.67
array
double
0 1.2
1 2.3
2 3.4

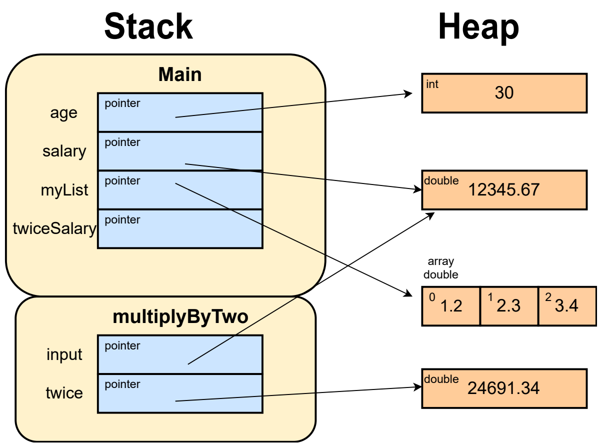


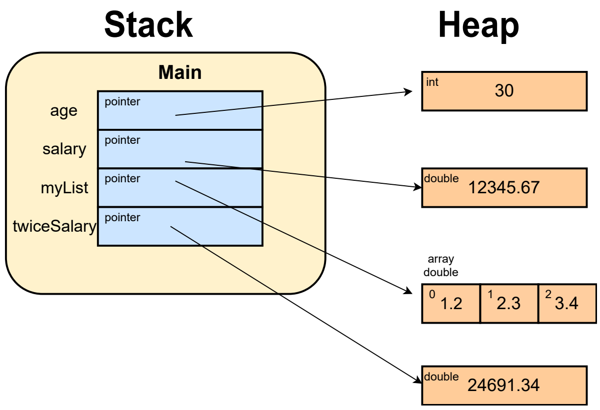
Stack
Heap
Main
age
salary
myList
twiceSalary
pointer
pointer
pointer
pointer
multiplyByTwo
input
pointer
int
30
double
12345.67
array
double
0 1.2
1 2.3
2 3.4

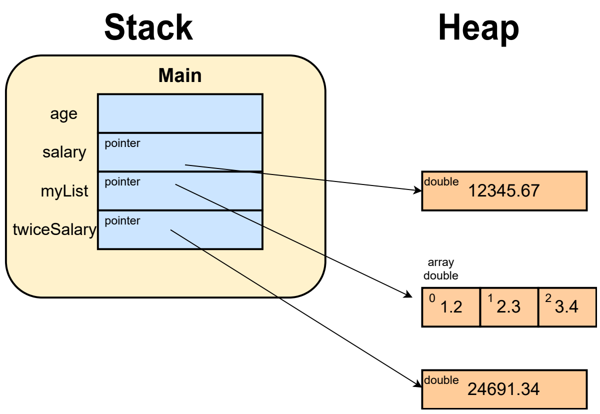

Stack
Heap
Main
age
salary
myList
twiceSalary
pointer
pointer
pointer
pointer
multiplyByTwo
input
twice
pointer
pointer
int
30
double
12345.67
array
double
0
1.2
1
2.3
2
3.4
double

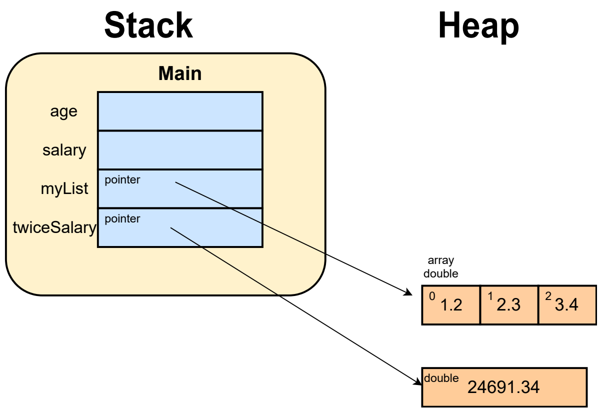

Stack
Heap
Main
age
salary
myList
twiceSalary
pointer
pointer
pointer
pointer
multiplyByTwo
input
twice
pointer
pointer
int
30
double
12345.67
array
double
0
1.2
1
2.3
2
3.4
double
24691.34


Stack
Heap
Main
age
salary
myList
twiceSalary
pointer
pointer
pointer
pointer
multiplyByTwo
input
twice
pointer
pointer
int
30
double
12345.67
array
double
0
1.2
1
2.3
2
3.4
double
24691.34

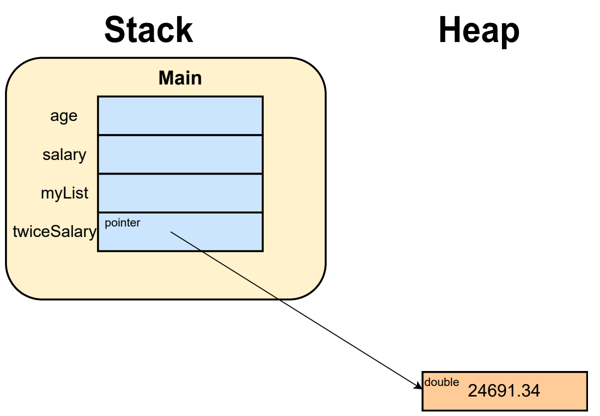

Stack
Heap
Main
age
salary
myList
twiceSalary
pointer
pointer
pointer
pointer
multiplyByTwo
input
twice
pointer
pointer
int
30
double
12345.67
array
double
0
1.2
1
2.3
2
3.4
double
24691.34


Stack
Heap
Main
age
salary
myList
twiceSalary
pointer
pointer
pointer
pointer
int
30
double
12345.67
array
double
0
1.2
1
2.3
2
3.4
double
24691.34

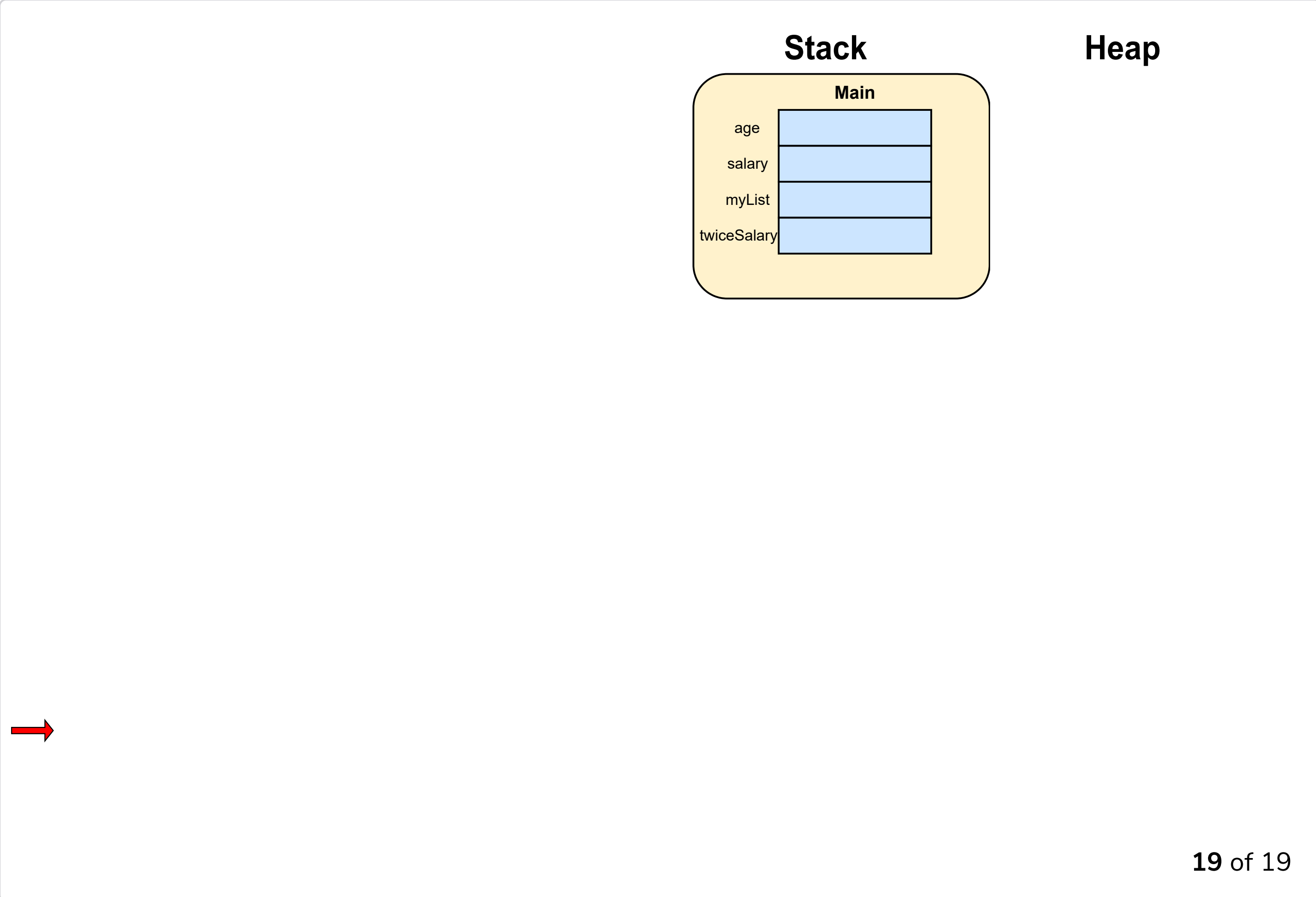

Stack
Heap
Main
age
salary
myList
twiceSalary
pointer
pointer
pointer
double
12345.67
array
double
0
1.2
1
2.3
2
3.4
double
24691.34



Stack
Heap
Main
age
salary
myList
twiceSalary
pointer
pointer
array
double
0 1.2
1 2.3
2 3.4
double
24691.34



Stack
Heap
Main
age
salary
myList
twiceSalary
pointer
double
24691.34



As you can see, using `malloc()` to allocate memory on the heap and then using `free()` to deallocate it, is no big deal, but is a bit cumbersome. The other thing to notice is that there are a bunch of star symbols * all over the place now. What are those? The answer is, they are **pointers**. The `malloc()` (and `calloc()` and `free()`) functions deal with **pointers** not actual values. We will talk more about pointers shortly. The bottom line though: pointers are a special data type in C that store **addresses in memory** instead of storing actual values. Thus on line 5 above, the `twice` variable is not a double but is a **pointer to a double**. It's an address in memory where the `double` is stored.