BORG AL ARAB TECHNOLOGICAL UNIVERSITY

CODE: ITS501

ADVANCED PROGRAMMING IN C

Third Year –Information Technology Program Fall 2026

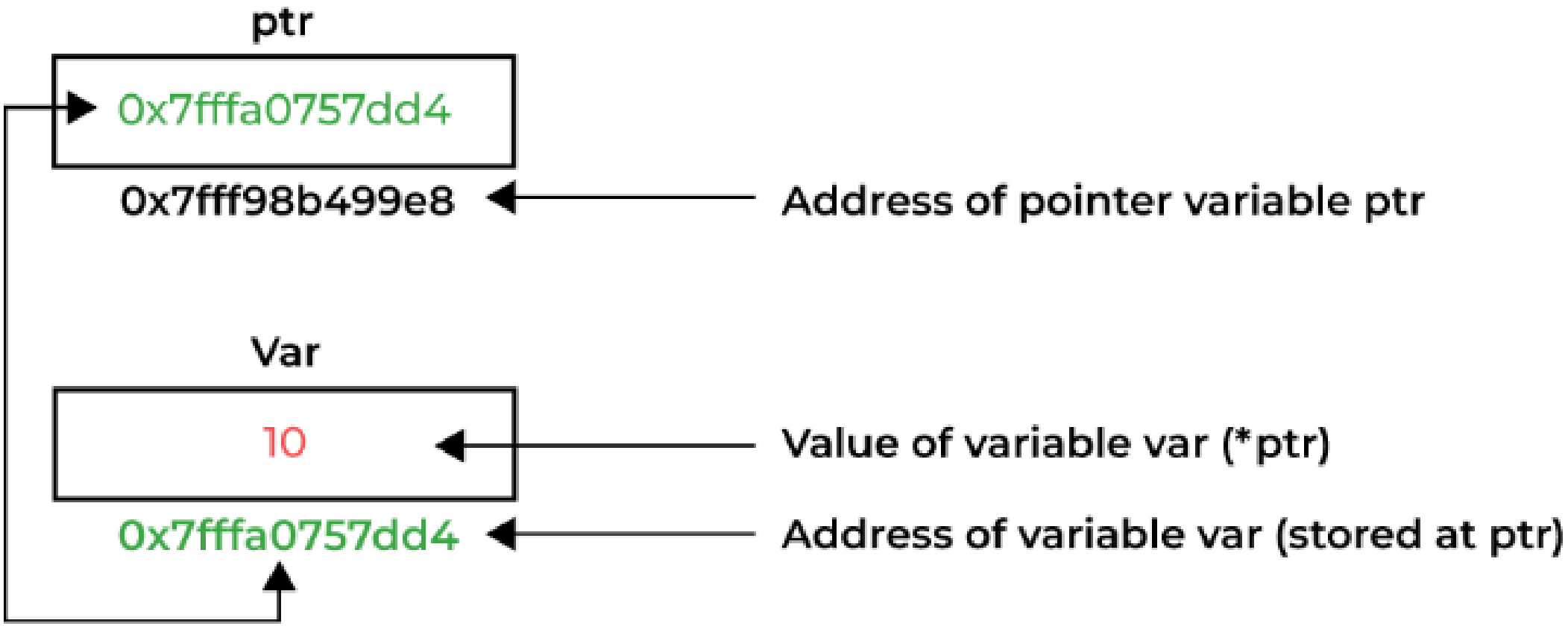
LECTURE 5

INTRODUCTION TO POINTERS IN C PROGRAMMING

WHAT ARE POINTERS?

- A pointer is a variable that stores the memory address of another variable.
- They are a core feature of C programming, providing powerful control over memory.

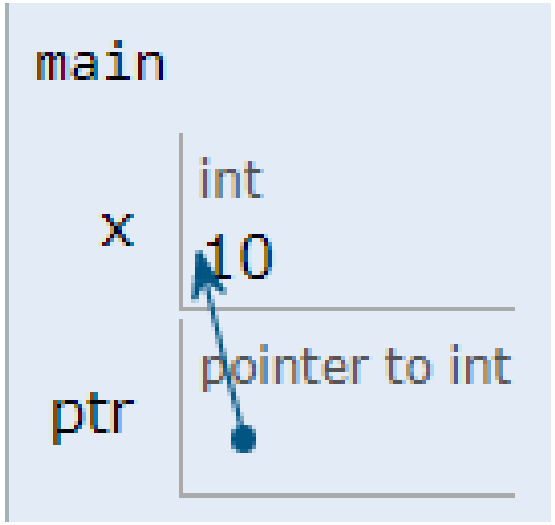
WHAT ARE POINTERS?



```
int main() {

int x = 10;
int *ptr = &x; // ptr stores the address of x
  return 0;
}
```

Stack



The variable ptr will hold the address of x.

WHY USE POINTERS?

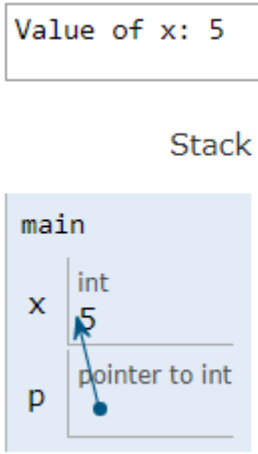
- To access and use memory directly.
- To work with arrays and strings efficiently.
- To enable dynamic memory allocation.
- For passing variables by reference in functions.

DECLARING AND USING POINTERS

- Syntax: data_type *pointer_name;

```
1  int main() {
2
3  int x = 5;
4  int *p = &x; // Pointer p holds the address of x
5  printf("Value of x: %d", *p); // Dereferencing
6    return 0;
7  }
```

```
Value of x: 5
```



DEREFERENCING A POINTER

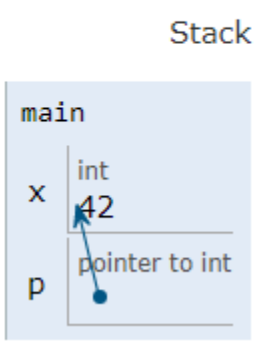
- Dereferencing: Accessing the value stored at the pointer's address.

```
int main() {

int x = 42;
int *p = &x;
printf("Address: %p, Value: %d", p, *p);
  return 0;
}
```

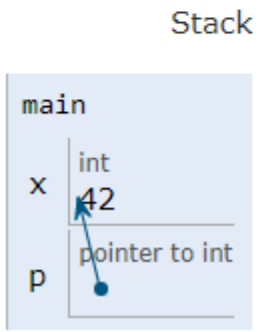
Print output (drag lower right corner to resize)

```
Address: 0xfff000bd4, Value: 42
```



```c
int main() {
    int x = 42;
    int *p = &x;
    printf("Address of pointer p: %p \n Address of x using pointer=value of the pointer: %p \n Value of x using pointer: %d\n", &p, p, *p);
    printf("Address of value of x: %d \n Address of x: %p", x, &x);
    return 0;

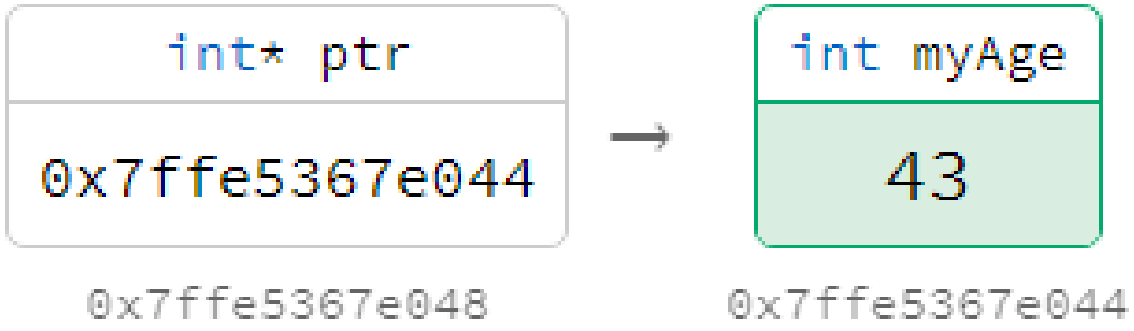
    }
```



Code

```c
int myAge = 43;
int* ptr = &myAge;
printf("%d\n", myAge);
```

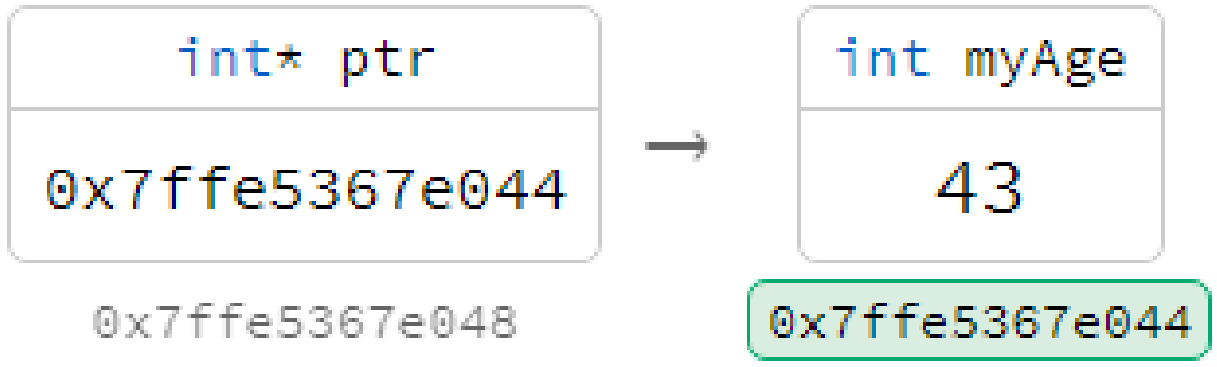
Memory



Code

```c
int myAge = 43;
int* ptr = &myAge;
printf("%p\n", &myAge);
```

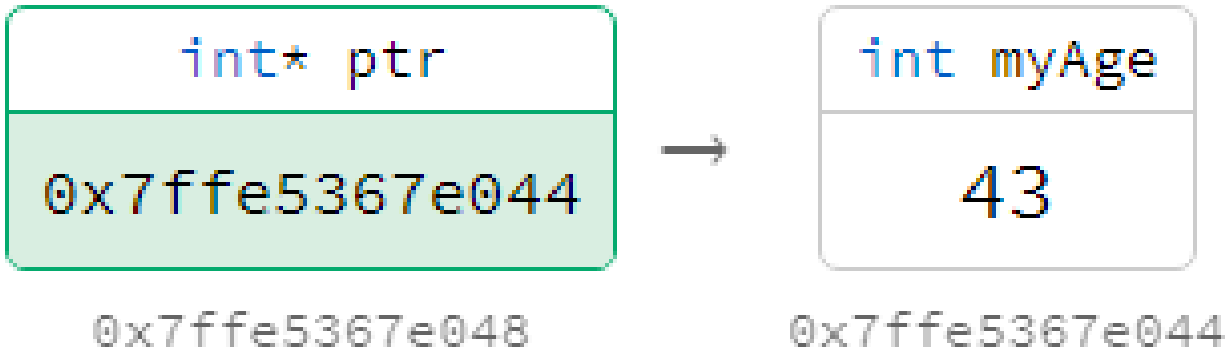
Memory



Code

```c
int myAge = 43;
int* ptr = &myAge;
printf("%p\n", ptr);
```

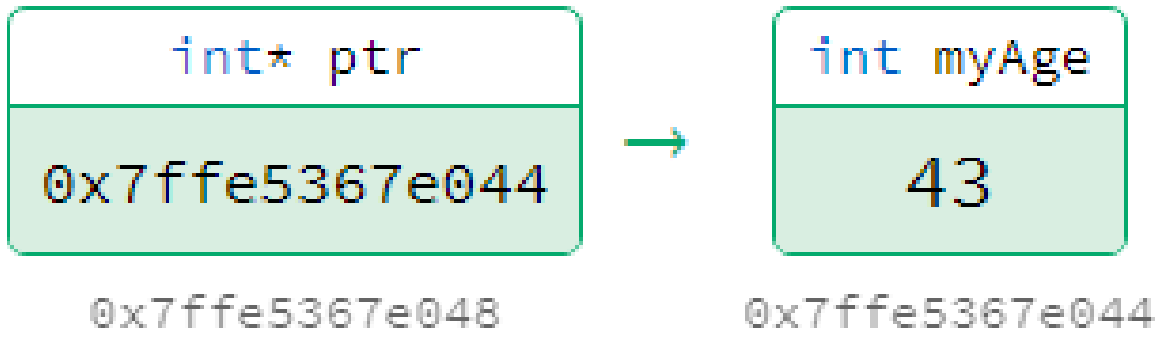
Memory



Code

```c
int myAge = 43;
int* ptr = &myAge;
printf("%d\n", *ptr);
```

Memory



- Create a pointer variable called **ptr**, that points to the int variable myAge:

```
int myAge = 43;
    = &myAge;
```

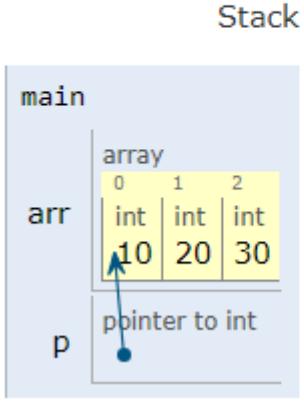
POINTERS AND ARRAYS

- Arrays and pointers are closely related in C.
- The name of an array is a pointer to its first element.

```
1  int main() {
2
3  int arr[3] = {10, 20, 30};
4  int *p = arr;
5  printf("First element: %d", *p);
6    return 0;
7  }
```

Print output (drag lower right corner to resize)

```
First element: 10
```



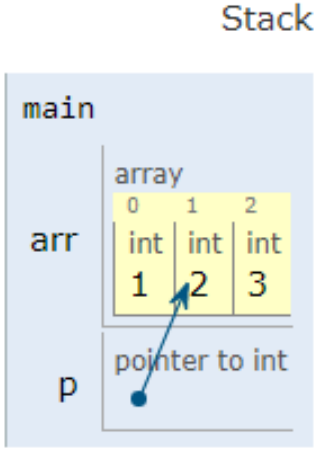
POINTER ARITHMETIC

```
int main() {


int arr[3] = {1, 2, 3};
int *p = arr;
printf("%d ", *p);     // Output: 1
p++;
printf("%d", *p);      // Output: 2
   return 0;
}
```

Print output (drag lower right corner to resize)

```
1 2
```



POINTER ARITHMETIC

```
//array elements are stored next to each other in memory. So if a //pointer
points to one element, adding 1 moves it to the next one:
#include <stdio.h>
int main() {
  int myNumbers[4] = {25, 50, 75, 100};
  int *p = myNumbers;  // points to myNumbers[0]

  printf("%d\n", *p);
  printf("%d\n", *(p + 1));
  printf("%d\n", *(p + 2));
  printf("%d\n", *(p + 3));
  printf("%d\n", *(p + 4));
  return 0;
}
```

POINTER ARITHMETIC

Answer
25
75
100
5424

POINTER ARITHMETICINCREMENT AND DECREMENT

```
//You can move a pointer with ++ and -- (and with += / -=):
Example
int myNumbers[3] = {10, 20, 30};
int *p = myNumbers;  // myNumbers[0]

printf("%d\n", *p); // 10
p++;           // move to myNumbers[1]
printf("%d\n", *p); // 20
p--;           // back to myNumbers[0]
printf("%d\n", *p); // 10

p += 2;        // jump to myNumbers[2]
printf("%d\n", *p); // 30
```

POINTER ARITHMETIC

Pointer Arithmetic Depends on Type
Not all pointers move the same way.
When you add 1 to a pointer, it moves forward by the size of the thing it points to - not just by 1 byte.
For example:
An int* pointer moves by the size of an integer (usually 4 bytes).
A char* pointer moves by the size of a character (1 byte).
So if both pointers start at memory address 1000:
int* → p + 1 would move to address 1004
char* → p + 1 would move to address 1001

POINTER ARITHMETIC

```
#include <stdio.h>
int main() {
  int myNumbers[2] = {1, 2};
  char letters[] = "Hi"; // 'H', 'i', '\0'
  int *pi = myNumbers;   // int pointer
  char *pc = letters;    // char pointer
  printf("%p\n", pi);
  printf("%p\n", pi + 1); // moves by sizeof(int)
  printf("%p\n", pi + 2); // moves by sizeof(int)
  printf("%p\n", pc);
  printf("%p\n", pc + 1); // moves by 1 byte
  return 0;
}
```

POINTER ARITHMETIC

0x7fff6e98fdcc
0x7fff6e98fdd0
0x7fff6e98fdd4
0x7fff6e98fdd5
0x7fff6e98fdd6

- What will this code print?

```
int myNumbers[4] = {25, 50, 75, 100};
int *p = myNumbers;
printf("%d", *(p + 2));
```

- 25
- 50
- 75
- 100

LOOPING WITH POINTERS

- you learned how to loop through an array using *(ptr + i).
- Now let's look at another way - by **moving the pointer itself** inside the loop. Each time the pointer is increased (p++), it moves to the next element in memory:

LOOPING WITH POINTERS

```
#include <stdio.h>

int main() {
  int myNumbers[4] = {25, 50, 75, 100};
  int *p = myNumbers;  // start of array

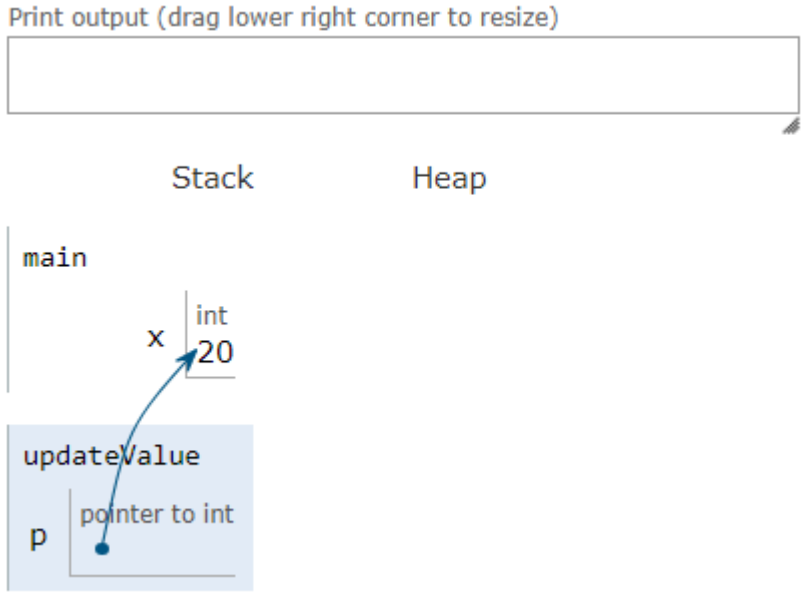
  for (int i = 0; i < 4; i++) {
    printf("%d\n", *p);
    p++; // move to next element
  }
  return 0;
}
```

LOOPING WITH POINTERS

25
50
75
100

POINTERS AND FUNCTIONS

```
1  void updateValue(int *p) {
2      *p = 20;
3  }
4
5  int main() {
6      int x = 10;
7      updateValue(&x);
8      printf("x: %d", x);
9      return 0;
10
11     }
```

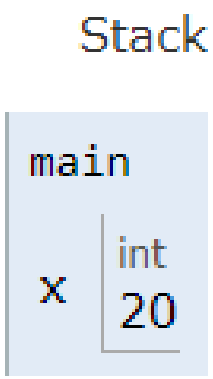


POINTERS AND FUNCTIONS

```
1  void updateValue(int *p) {
2      *p = 20;
3  }
4
5  int main() {
6      int x = 10;
7      updateValue(&x);
8      printf("x: %d", x);
9      return 0;
10
11     }
```

Print output (drag lower right corner to resize)

```
x: 20
```



VISUALIZE A POINTER TO POINTER

You can also have a **pointer that points to another pointer**. This is called a **pointer to pointer** (or "double pointer").

- a pointer that stores the *address of another pointer*.

VISUALIZE A POINTER TO POINTER

```
#include <stdio.h>

int main() {
  int myNum = 10;       // normal variable
  int *ptr = &myNum;    // pointer to int
  int **pptr = &ptr;    // pointer to pointer

  printf("myNum = %d\n", myNum);
  printf("*ptr = %d\n", *ptr);
  printf("**pptr = %d\n", **pptr);

  return 0;
}
```

VISUALIZE A POINTER TO POINTER

myNum = 10

*ptr = 10

**pptr = 10

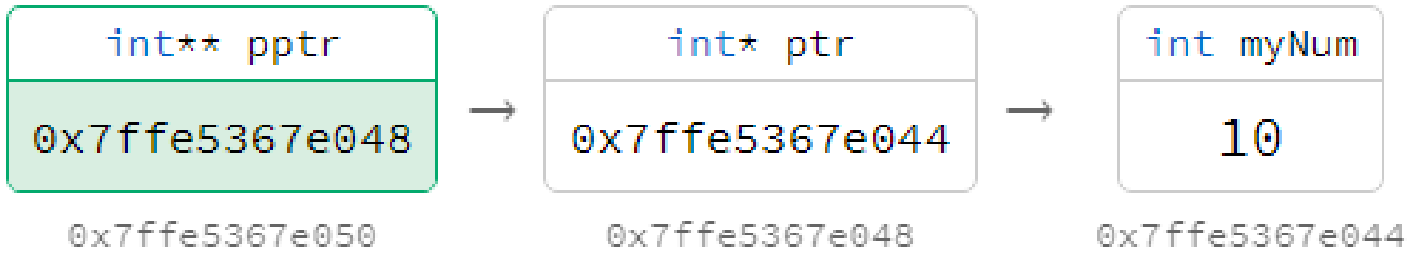
Here's what happens step by step:

- myNum holds the value 10.
- ptr holds the address of myNum.
- pptr holds the address of ptr.
- *ptr gives the value of myNum.
- **pptr also gives the value of myNum, by going through both pointers.

Code

```c
int myNum = 10;
int *ptr = &myNum;
int **pptr = &ptr;
printf("%p\n", pptr);
```

Memory



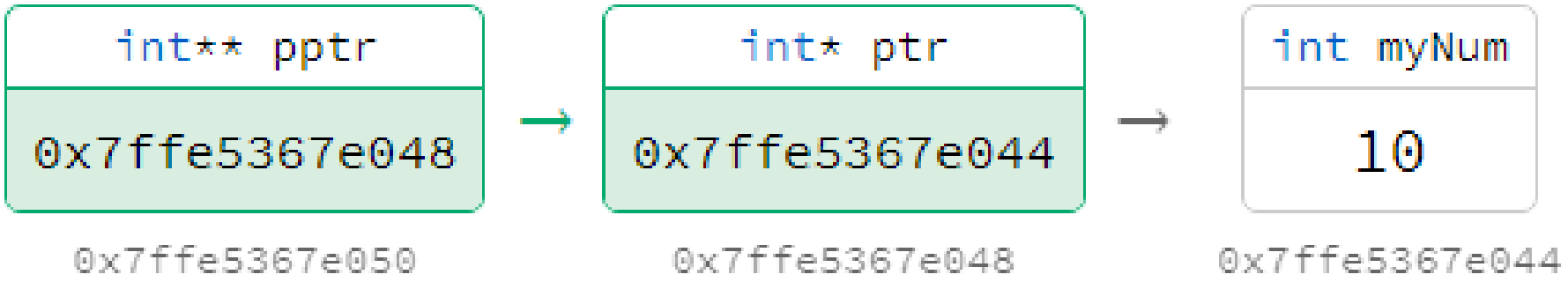
Output

```
0x7ffe5367e048
```

Code

```c
int myNum = 10;
int *ptr = &myNum;
int **pptr = &ptr;
printf("%p\n", *pptr);
```

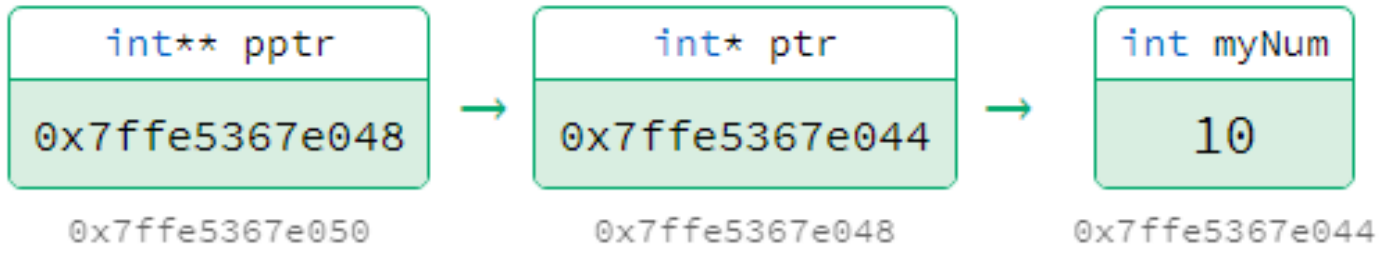
Memory



Code

```c
int myNum = 10;
int *ptr = &myNum;
int **pptr = &ptr;
printf("%d\n", **pptr);
```

Memory



Output

```
10
```

QUIZ: TEST YOUR KNOWLEDGE ON POINTERS

- **Question 1:**
- What does a pointer store?

  A. Value of a variable

  B. Address of a variable

  C. Data type of a variable

  D. Size of a variable

QUESTION 2: DECLARING POINTERS

- Which of the following is a correct pointer declaration?

  A. int *ptr;

  B. int ptr*;

  C. *int ptr;

  D. ptr int*;

QUESTION 3: DEREFERENCING A POINTER

- What will this code output?

```
int x = 10;

int *ptr = &x;

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

A. 10
B. Address of x
C. Compilation error
D. Undefined behavior

QUESTION 4: POINTERS AND ARRAYS

```
int arr[] = {1, 2, 3};
```

what is the output of this code?

```
printf("%d", *(arr + 1));
```

A. 1
B. 2
C. 3
D. Compilation error

QUESTION 5: POINTERS AND ARRAYS

- What will this code print?

```
int myNum = 10;
int *ptr = &myNum;
int **pptr = &ptr;
printf("%d", **pptr);
```

- 10
- The memory address of myNum
- The memory address of ptr
- An error

QUESTION 5: POINTERS

What does a pointer to pointer store?

- The value of a variable
- The memory address of another pointer
- Two values
- The size of a pointer

- Look at this code:
  int myNum = 10;
  int *ptr = &myNum;
  int **pptr = &ptr;
  Which expression gives the memory address of myNum?
- pptr
- *pptr
- **pptr
- &pptr

- Create a pointer to pointer named **pptr** that stores the address of **ptr**, and use it to change **myNum** to 20:

```
int myNum = 10;

int *ptr = &myNum;

int **pptr = &ptr;

**pptr = 20;
```

TRUE/FALSE QUIZ: POINTERS IN C

- A pointer stores the value of a variable.
  (False) – A pointer stores the address of a variable.
- The operator * is used to access the value at the address a pointer points to.
  (True)
- Pointers can only point to integers in C.
  (False) – Pointers can point to any data type.
- The statement int *ptr = &x; declares a pointer and assigns it the address of x.
  (True)
- Arrays and pointers are not related in C programming.
  (False) – Arrays and pointers are closely related; the array name acts as a pointer to its first element.
- The & operator is used to get the value stored at a pointer's address.
  (False) – The & operator is used to get the address of a variable.
- You can perform arithmetic operations on pointers.
  (True) – Pointer arithmetic is allowed, e.g., incrementing to access the next memory location.

SUMMARY

- Pointers are variables storing memory addresses.
- They are powerful for managing memory and enabling advanced features.
- Avoid common mistakes like uninitialized pointers.

Happy Coding