Pointers and Arrays

Hyung-Sin Kim

Computing for Data Science

Graduate School of Data Science, Seoul National University

Pointer – Motivation (Call by Value)

- Swapping function in C

```
#include <stdio.h>

void swap(int firstVal, int secondVal);

int main(void) {
    int valA = 7;
    int valB = 5;
    printf(“Before Swap: valA = %d, valB = %d\n”, valA, valB);
    swap(valA, valB);
    printf(“After Swap: valA = %d, valB = %d\n”, valA, valB);
    return 0;
}

void swap(int firstVal, int secondVal) {
    int tempVal;
    tempVal = firstVal;
    firstVal = secondVal;
    secondVal = tempVal;
    printf(“In Swap: firstVal = %d, secondVal = %d\n”, firstVal, secondVal);
}
```

What do you see on your screen?

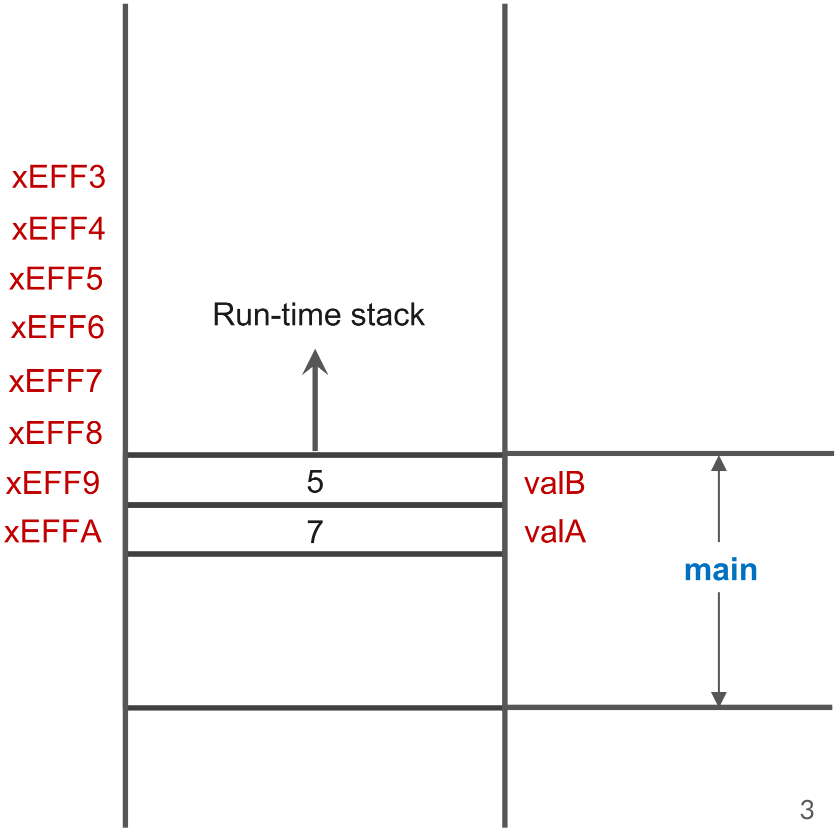
Local variables firstVal and secondVal **die** when function swap ends!

Swapping does not happen!

Pointer – Motivation (Call by Value)

- Swapping function in C

```
#include <stdio.h>

void swap(int firstVal, int secondVal);

int main(void) {
    int valA = 7;
    int valB = 5;
    printf(“Before Swap: valA = %d, valB = %d\n”, valA, valB);
    swap(valA, valB);
    printf(“After Swap: valA = %d, valB = %d\n”, valA, valB);
    return 0;
}

void swap(int firstVal, int secondVal) {
    int tempVal;
    tempVal = firstVal;
    firstVal = secondVal;
    secondVal = tempVal;
    printf(“In Swap: firstVal = %d, secondVal = %d\n”, firstVal, secondVal);
}
```

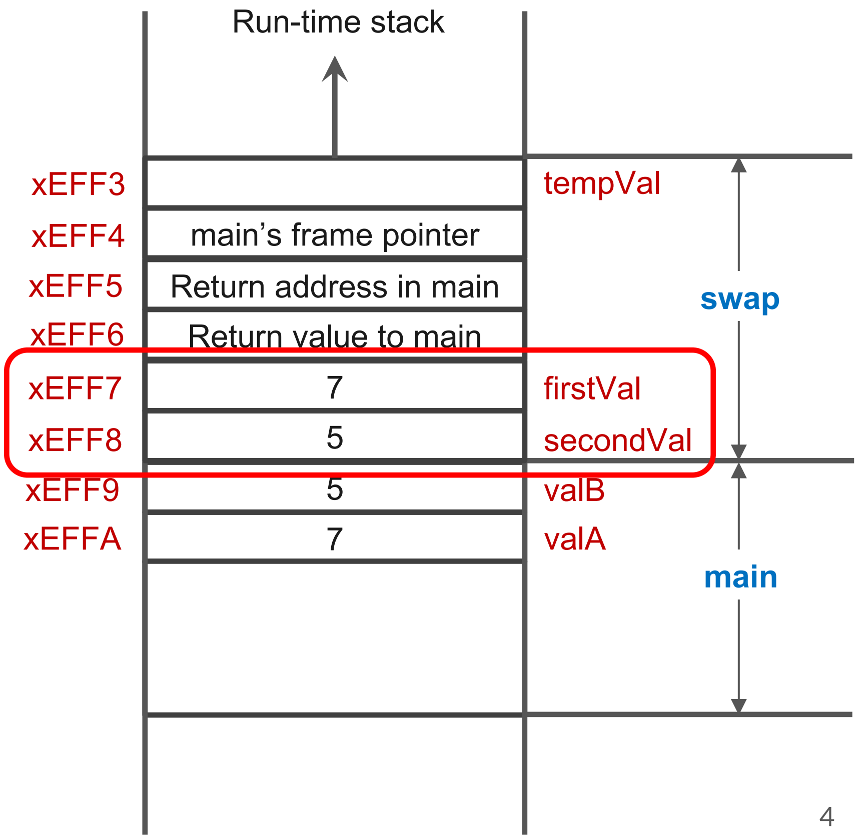


Pointer – Motivation (Call by Value)

Call by value
Same value but a
separate memory object!

- Swapping function in C

```
#include <stdio.h>

void swap(int firstVal, int secondVal);

int main(void) {
    int valA = 7;
    int valB = 5;
    printf("Before Swap: valA = %d, valB = %d\n", valA, valB);
    swap(valA, valB);
    printf("After Swap: valA = %d, valB = %d\n", valA, valB);
    return 0;
}

void swap(int firstVal, int secondVal) {
    int tempVal;
    tempVal = firstVal;
    firstVal = secondVal;
    secondVal = tempVal;
    printf("In Swap: firstVal = %d, secondVal = %d\n", firstVal, secondVal);
}
```

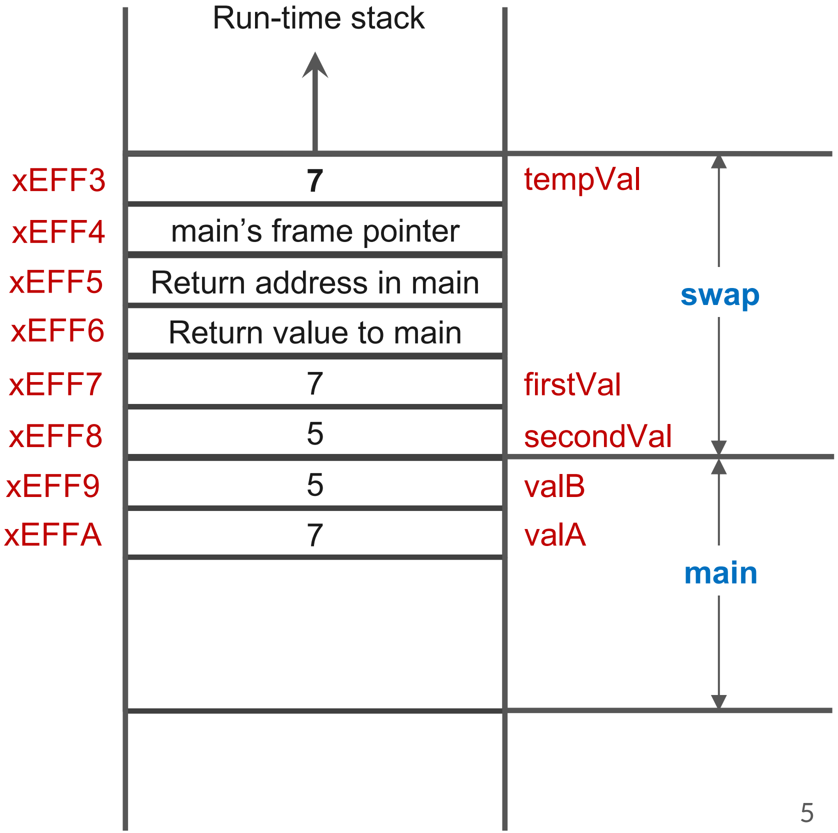


xEFF3		tempVal
xEFF4	main's frame pointer	
xEFF5	Return address in main	
xEFF6	Return value to main	
xEFF7	7	firstVal
xEFF8	5	secondVal
xEFF9	5	valB
xEFFA	7	valA

Pointer – Motivation (Call by Value)

- Swapping function in C

```
#include <stdio.h>

void swap(int firstVal, int secondVal);

int main(void) {
    int valA = 7;
    int valB = 5;
    printf("Before Swap: valA = %d, valB = %d\n", valA, valB);
    swap(valA, valB);
    printf("After Swap: valA = %d, valB = %d\n", valA, valB);
    return 0;
}

void swap(int firstVal, int secondVal) {
    int tempVal;
    tempVal = firstVal;
    firstVal = secondVal;
    secondVal = tempVal;
    printf("In Swap: firstVal = %d, secondVal = %d\n", firstVal, secondVal);
}
```



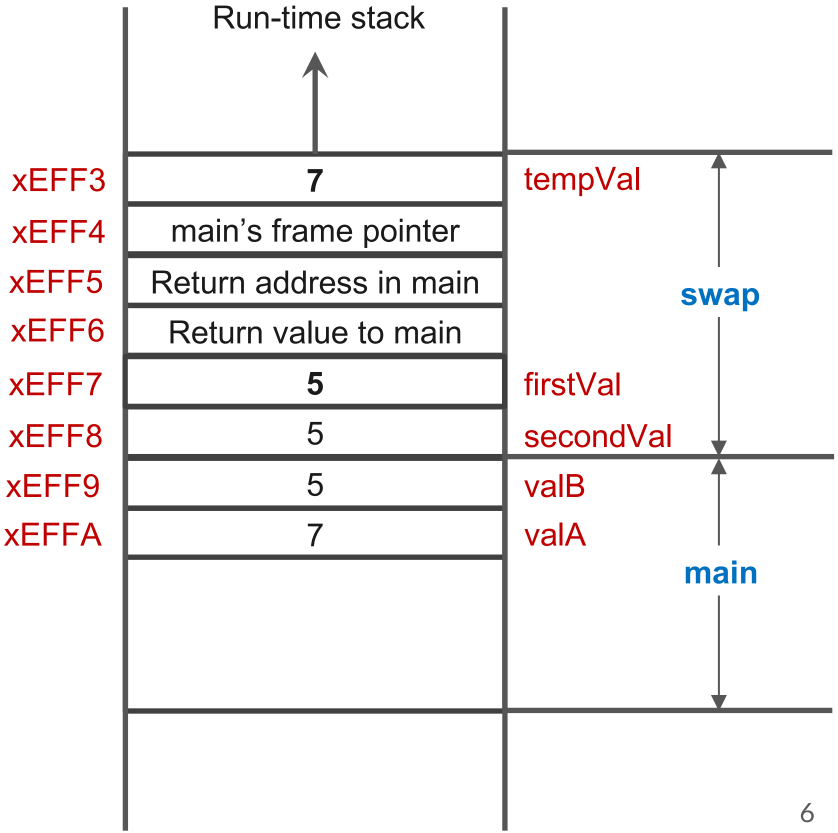
Run-time stack

xEFF3	7	tempVal	swap
xEFF4	main’s frame pointer		
xEFF5	Return address in main		
xEFF6	Return value to main		
xEFF7	7	firstVal	
xEFF8	5	secondVal	
xEFF9	5	valB	main
xEFFA	7	valA	

Pointer – Motivation (Call by Value)

- Swapping function in C

```
#include <stdio.h>

void swap(int firstVal, int secondVal);

int main(void) {
    int valA = 7;
    int valB = 5;
    printf("Before Swap: valA = %d, valB = %d\n", valA, valB);
    swap(valA, valB);
    printf("After Swap: valA = %d, valB = %d\n", valA, valB);
    return 0;
}

void swap(int firstVal, int secondVal) {
    int tempVal;
    tempVal = firstVal;
    firstVal = secondVal;
    secondVal = tempVal;
    printf("In Swap: firstVal = %d, secondVal = %d\n", firstVal, secondVal);
}
```



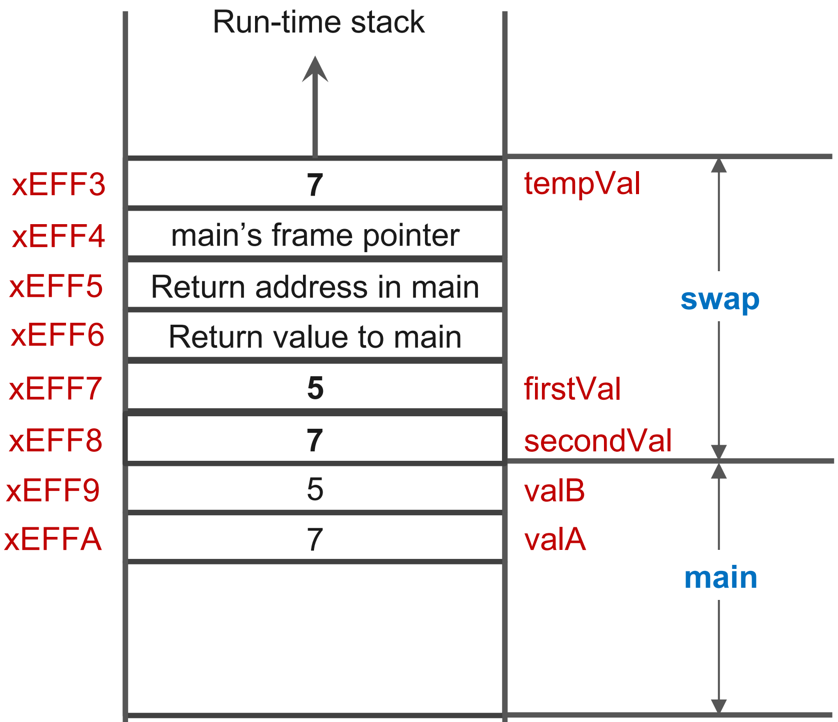
Run-time stack

xEFF3	**7**	tempVal
xEFF4	main's frame pointer	
xEFF5	Return address in main	
xEFF6	Return value to main	
xEFF7	**5**	firstVal
xEFF8	5	secondVal
xEFF9	5	valB
xEFFA	7	valA

Pointer – Motivation (Call by Value)

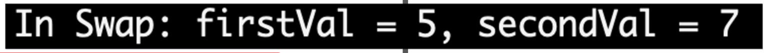
- Swapping function in C

```
#include <stdio.h>

void swap(int firstVal, int secondVal);

int main(void) {
    int valA = 7;
    int valB = 5;
    printf("Before Swap: valA = %d, valB = %d\n", valA, valB);
    swap(valA, valB);
    printf("After Swap: valA = %d, valB = %d\n", valA, valB);
    return 0;
}

void swap(int firstVal, int secondVal) {
    int tempVal;
    tempVal = firstVal;
    firstVal = secondVal;
    secondVal = tempVal;
    printf("In Swap: firstVal = %d, secondVal = %d\n", firstVal, secondVal);
}
```



Run-time stack

xEFF3	**7**	tempVal
xEFF4	main's frame pointer	
xEFF5	Return address in main	
xEFF6	Return value to main	
xEFF7	**5**	firstVal
xEFF8	**7**	secondVal
xEFF9	5	valB
xEFFA	7	valA



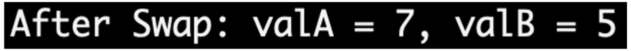
```
In Swap: firstVal = 5, secondVal = 7
```

Pointer – Motivation (Call by Value)

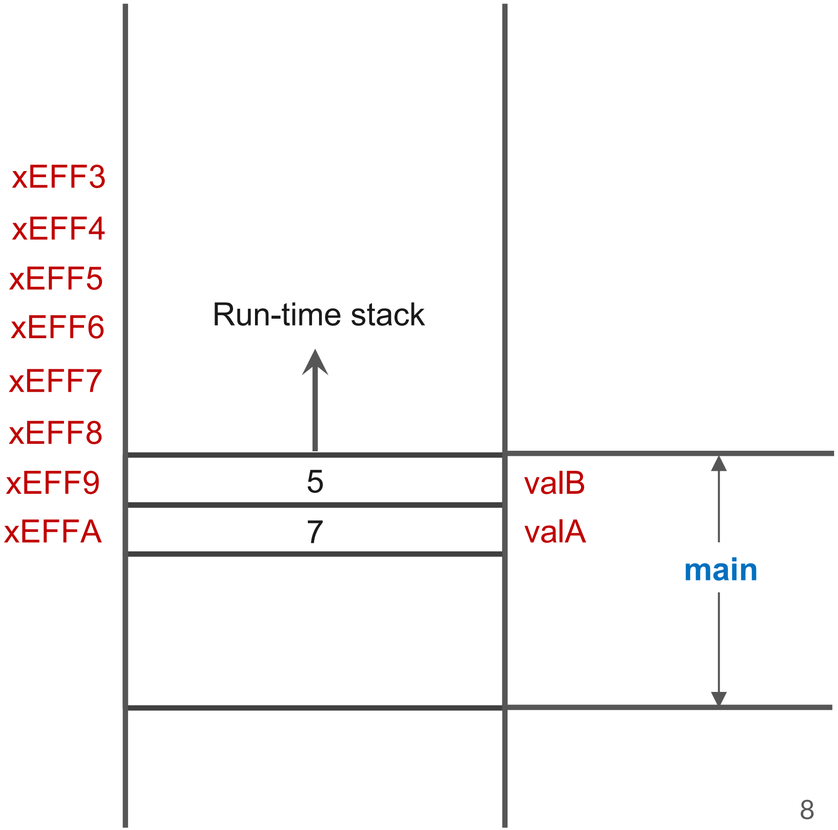
All local variables (swap scope) are gone!

- Swapping function in C

```
#include <stdio.h>

void swap(int firstVal, int secondVal);

int main(void) {
    int valA = 7;
    int valB = 5;
    printf("Before Swap: valA = %d, valB = %d\n", valA, valB);
    swap(valA, valB);
    printf("After Swap: valA = %d, valB = %d\n", valA, valB);
    return 0;
}

void swap(int firstVal, int secondVal) {
    int tempVal;
    tempVal = firstVal;
    firstVal = secondVal;
    secondVal = tempVal;
    printf("In Swap: firstVal = %d, secondVal = %d\n", firstVal, secondVal);
}
```



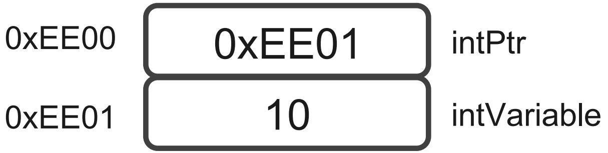
```
After Swap: valA = 7, valB = 5
```



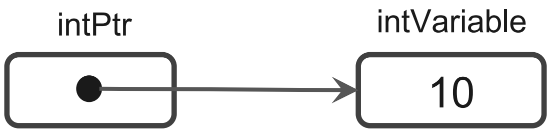
How can we make the swap function impact the arguments?

Pointer – Declaration

- A pointer variable contains an address of a memory object (e.g., variable)
  - <type> *<name>
    - int *ptr; → ptr is a variable that contains an address of an integer variable
    - char *ptr; → ptr is a variable that contains an address of a character variable
- Address operator & and indirection operator *
  - int intVariable = 10; // Assume that intVariable’s address is 0xEE01
  - int *intPtr;
  - intPtr = &intVariable;
    - Now intPtr contains intVariable’s address
    - *intPtr is the value in the memory object that intPtr points to (i.e., intVariable’s value, 10)
    - “*intPtr = *intPtr + 2” is the same as “intVariable = intVariable + 2”



0xEE00	0xEE01	intPtr
0xEE01	10	intVariable

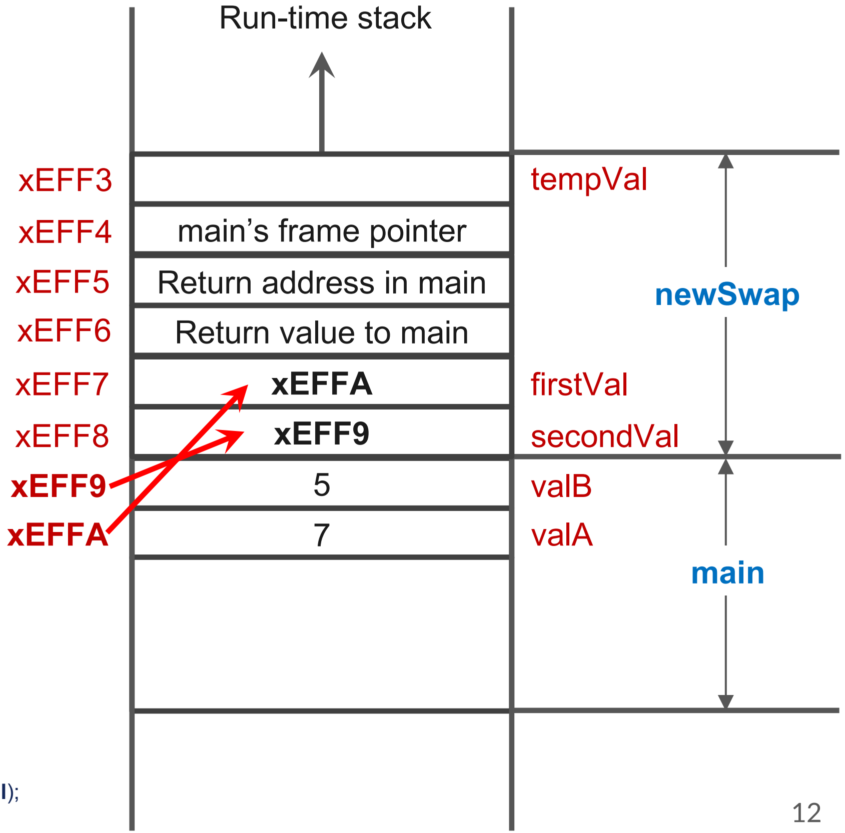


Let’s print some values!

Pointer – Swap (Call by Reference)

- Swapping function in C

```
#include <stdio.h>

void newSwap(int *firstVal, int *secondVal);

int main(void) {
   int valA = 7;
   int valB = 5;
   printf(“Before Swap: valA = %d, valB = %d\n”, valA, valB);
   newSwap(&valA, &valB);
   printf(“After Swap: valA = %d, valB = %d\n”, valA, valB);
   return 0;
}

void newSwap(int *firstVal, int *secondVal) {
   int tempVal;
   tempVal = *firstVal;
   *firstVal = *secondVal;
   *secondVal = tempVal;
   printf(“In Swap: firstVal = %d, secondVal = %d\n”, *firstVal, *secondVal);
}
```



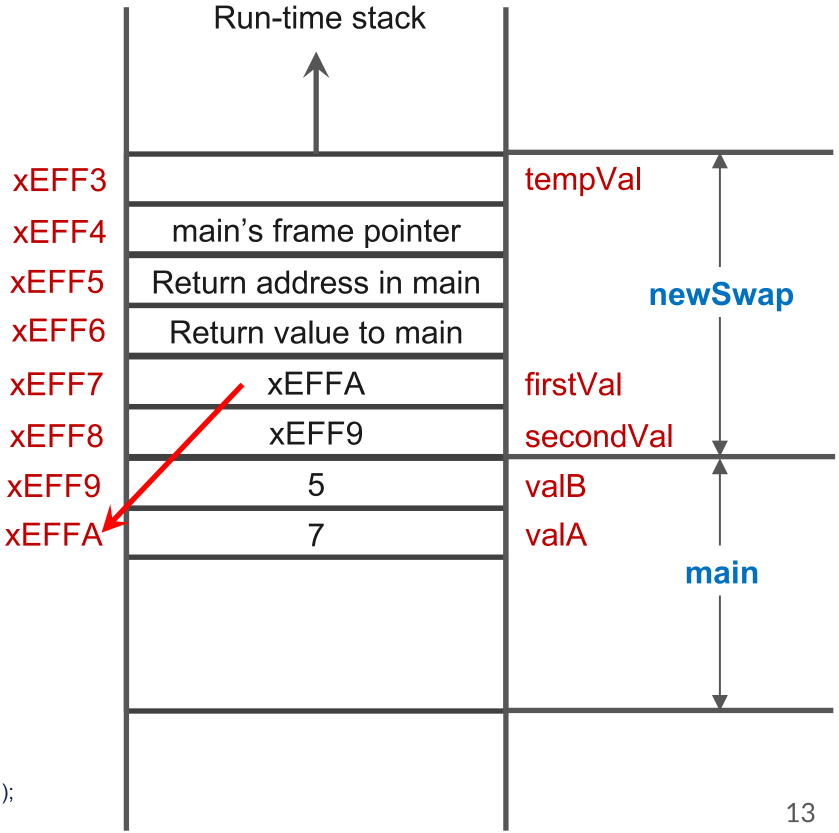
Run-time stack

xEFF3		tempVal	newSwap
xEFF4	main’s frame pointer		
xEFF5	Return address in main		
xEFF6	Return value to main		
xEFF7	xEFFA	firstVal	
xEFF8	xEFF9	secondVal	
xEFF9	5	valB	main
xEFFA	7	valA	

Pointer – Swap (Call by Reference)

- Swapping function in C

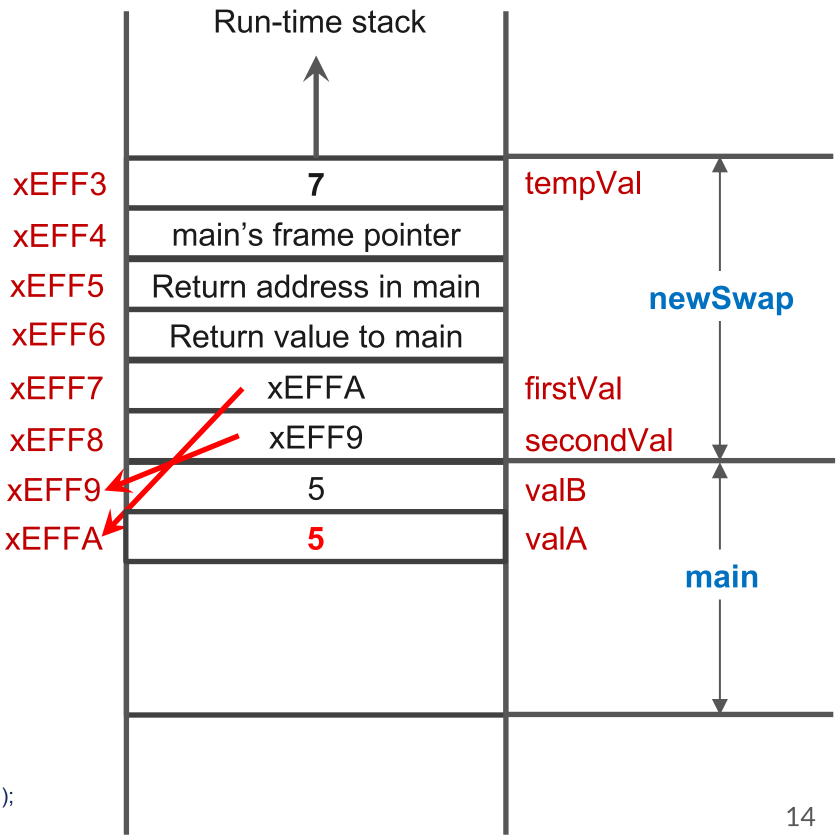
```
#include <stdio.h>

void newSwap(int *firstVal, int *secondVal);

int main(void) {
   int valA = 7;
   int valB = 5;
   printf("Before Swap: valA = %d, valB = %d\n", valA, valB);
   newSwap(&valA, &valB);
   printf("After Swap: valA = %d, valB = %d\n", valA, valB);
   return 0;
}

void newSwap(int *firstVal, int *secondVal) {
   int tempVal;
   tempVal = *firstVal;
   *firstVal = *secondVal;
   *secondVal = tempVal;
   printf("In Swap: firstVal = %d, secondVal = %d\n", *firstVal, *secondVal);
}
```



Pointer – Swap (Call by Reference)

- Swapping function in C

```
#include <stdio.h>

void newSwap(int *firstVal, int *secondVal);

int main(void) {
    int valA = 3;
    int valB = 4;
    printf(“Before Swap: valA = %d, valB = %d\n”, valA, valB);
    newSwap(&valA, &valB);
    printf(“After Swap: valA = %d, valB = %d\n”, valA, valB);
    return 0;
}

void newSwap(int *firstVal, int *secondVal) {
    int tempVal;
    tempVal = *firstVal;
    *firstVal = *secondVal;
    *secondVal = tempVal;
    printf(“In Swap: firstVal = %d, secondVal = %d\n”, *firstVal, *secondVal);
}
```



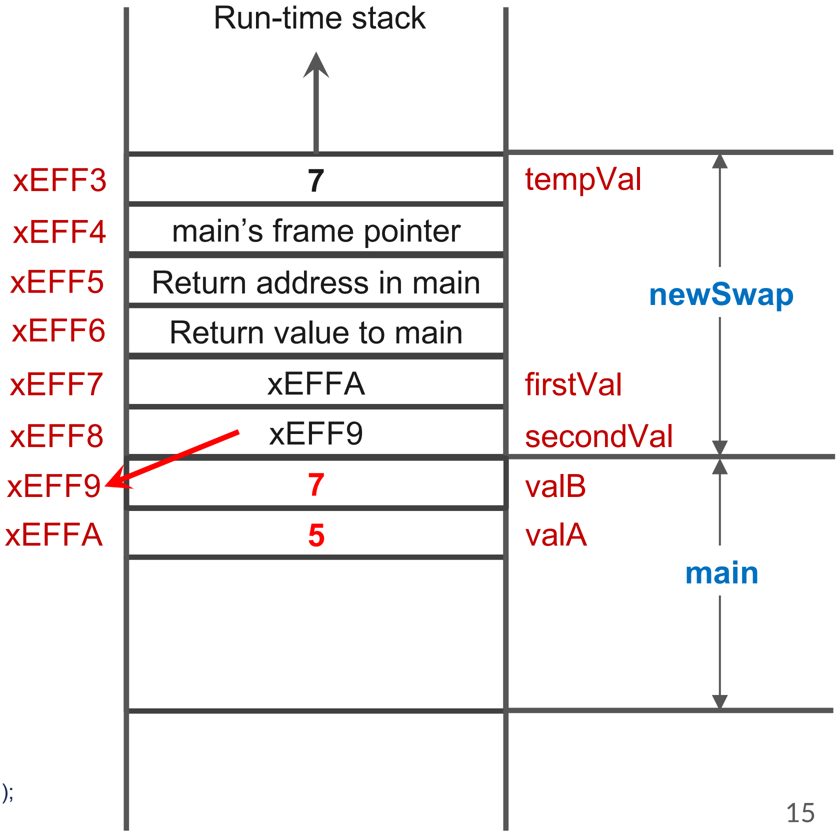
Run-time stack

xEFF3	7	tempVal
xEFF4	main’s frame pointer	
xEFF5	Return address in main	
xEFF6	Return value to main	
xEFF7	xEFFA	firstVal
xEFF8	xEFF9	secondVal
xEFF9	5	valB
xEFFA	5	valA

Pointer – Swap (Call by Reference)

- Swapping function in C

```
#include <stdio.h>

void newSwap(int *firstVal, int *secondVal);

int main(void) {
   int valA = 3;
   int valB = 4;
   printf(“Before Swap: valA = %d, valB = %d\n”, valA, valB);
   newSwap(&valA, &valB);
   printf(“After Swap: valA = %d, valB = %d\n”, valA, valB);
   return 0;
}

void newSwap(int *firstVal, int *secondVal) {
   int tempVal;
   tempVal = *firstVal;
   *firstVal = *secondVal;
   *secondVal = tempVal;
   printf(“In Swap: firstVal = %d, secondVal = %d\n”, *firstVal, *secondVal);
}
```



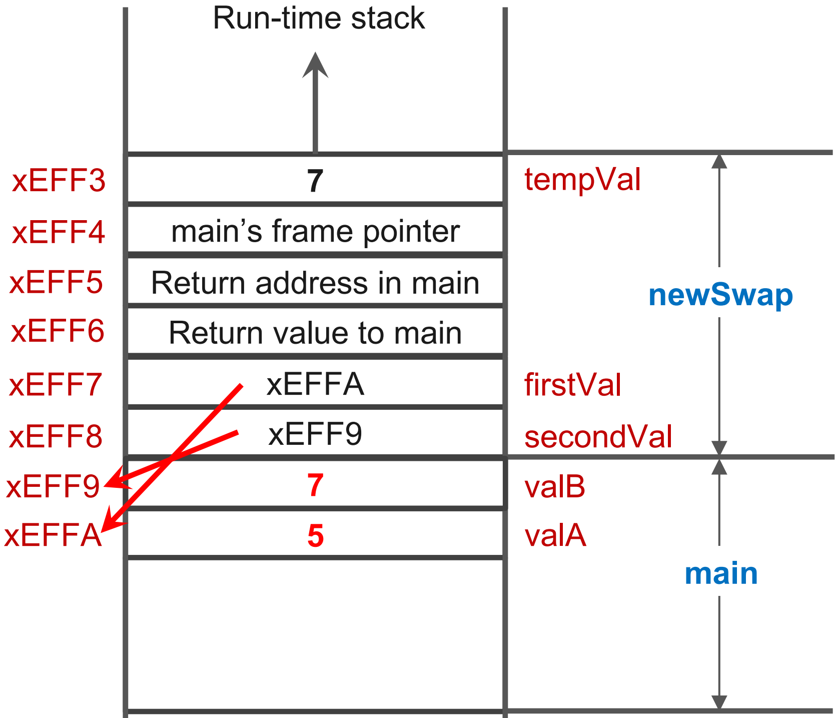
Run-time stack

xEFF3	7	tempVal	newSwap
xEFF4	main’s frame pointer		
xEFF5	Return address in main		
xEFF6	Return value to main		
xEFF7	xEFFA	firstVal	
xEFF8	xEFF9	secondVal	
xEFF9	7	valB	main
xEFFA	5	valA	

Pointer – Swap (Call by Reference)

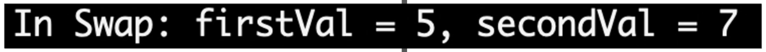
- Swapping function in C

```
#include <stdio.h>

void newSwap(int *firstVal, int *secondVal);

int main(void) {
    int valA = 3;
    int valB = 4;
    printf("Before Swap: valA = %d, valB = %d\n", valA, valB);
    newSwap(&valA, &valB);
    printf("After Swap: valA = %d, valB = %d\n", valA, valB);
    return 0;
}

void newSwap(int *firstVal, int *secondVal) {
    int tempVal;
    tempVal = *firstVal;
    *firstVal = *secondVal;
    *secondVal = tempVal;
    printf("In Swap: firstVal = %d, secondVal = %d\n", *firstVal, *secondVal);
}
```



Run-time stack

xEFF3	7	tempVal
xEFF4	main's frame pointer	
xEFF5	Return address in main	
xEFF6	Return value to main	
xEFF7	xEFFA	firstVal
xEFF8	xEFF9	secondVal
xEFF9	7	valB
xEFFA	5	valA



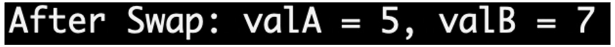
```
In Swap: firstVal = 5, secondVal = 7
```

Pointer – Swap (Call by Reference)

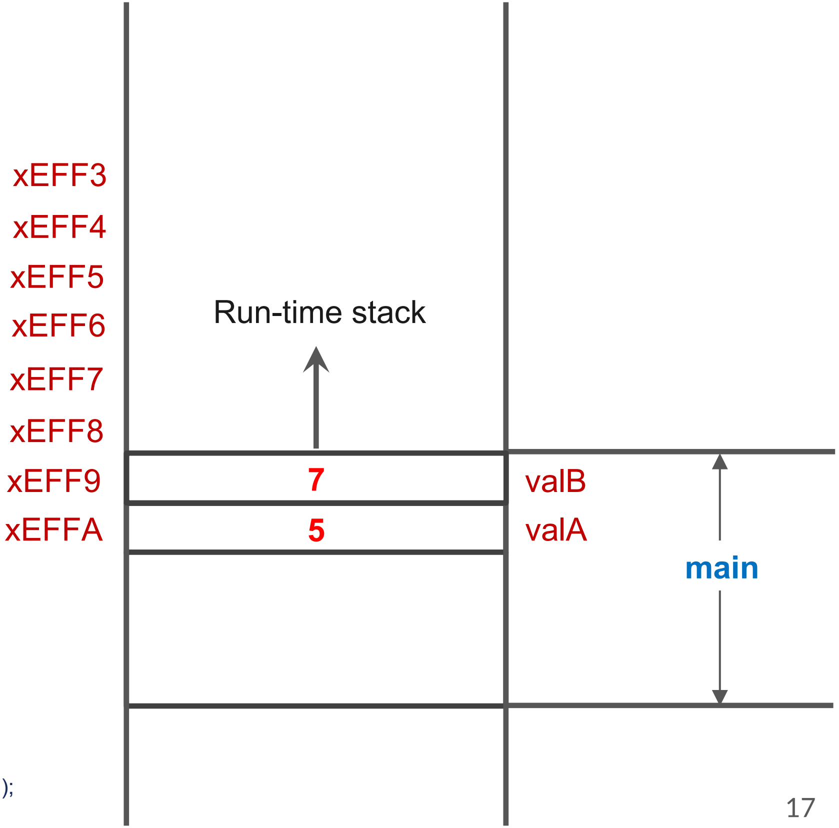
Call by reference
Directly impact
function arguments!

- Swapping function in C

```
#include <stdio.h>

void newSwap(int *firstVal, int *secondVal);

int main(void) {
    int valA = 3;
    int valB = 4;
    printf("Before Swap: valA = %d, valB = %d\n", valA, valB);
    newSwap(&valA, &valB);
    printf("After Swap: valA = %d, valB = %d\n", valA, valB);
    return 0;
}

void newSwap(int *firstVal, int *secondVal) {
    int tempVal;
    tempVal = *firstVal;
    *firstVal = *secondVal;
    *secondVal = tempVal;
    printf("In Swap: firstVal = %d, secondVal = %d\n", *firstVal, *secondVal);
}
```



```
After Swap: valA = 5, valB = 7
```



Pointer – etc.

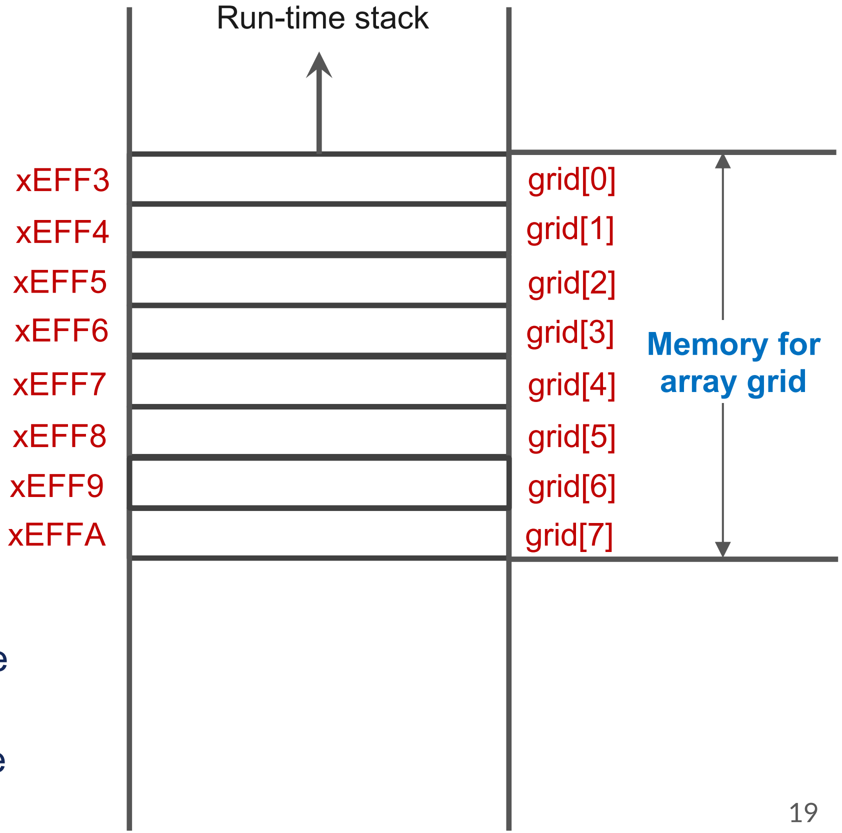
- Null pointers – A special case pointer that points to **nothing**
  - int *ptr;
  - ptr = NULL;   // NULL is a specially defined preprocessor macro that contains a value 0
  - It is useful to initialize a pointer to NULL when it does not point anything yet

- Demystifying the syntax
  - Pointer declaration (e.g., int *ptr;)
    - Declaring a variable **ptr** that, when the **indirection operator *** is applied to it, generates a value of type **int**
    - That is, ***ptr** is integer type
  - Input library function – scanf("%d", **&**input);
    - To change the value of the function argument **"input,"** scanf must have the **address** of "input"
    - If you omit **&**, C compiler will kindly give an error message

Array

- An array is a collection of **similar data items** that are stored **sequentially in memory** and accessible through a **single name** or identifier

- In contrast to lists in Python, an array in C
  - Can store only a **single** data type
  - Has a **fixed size**
- Declaration - int grid[8];
  - **grid** is an array of seven integer variables
  - The first element (grid[0]) is allocated in the lowest memory address
  - The last element (grid[7]) is allocated in the highest memory address



Array – Example

```
#include <stdio.h>
#define NUM_STUDENTS 5

int main(void) {
    int midterm[NUM_STUDENTS];
    int final[NUM_STUDENTS];
    int total[NUM_STUDENTS];

   // Input exam scores
   for (int i=0; i < NUM_STUDENTS; i++) {
        printf(“Input midterm score for student %d: ”, i);
        scanf(“%d”, &miterm[i]);
        printf(“Input final score for student %d: ”, i);
        scanf(“%d”, &final[i]);
    }
```

```
   // Calculate total scores
   for (int i=0; i < NUM_STUDENTS; i++) {
        total[i] = midterm[i] + final[i];
    }

    // Output the total scores
    for (int i=0; I < NUM_STUDENTS; i++) {
        printf(“Total score for Student %d is %d\n”, i, total[i]);
    }

    return 0;
}
```

Array – Relationship with Pointer

- Example
  - int values[10]; // Without any index, **values** itself is the same as &values[0]
  - int *valPtr;
  - valPtr = values;
- valPtr and values are very similar as shown below:
  - One difference is that **valPtr** can be reassigned but **values** cannot be reassigned
  - values = newArray[xx]; will cause a compiler error

	Using a Pointer	Using Name of Array	Using Array Notation
Address of array	valPtr	values	&values[0]
0-th element	*valPtr	*values	values[0]
Address of n-th element	(valPtr + n)	(values + n)	&values[n]
n-th element	*(valPtr + n)	*(values + n)	values[n]

Array – Passing by Reference

- Averaging function

```
#include <stdio.h>
#define MAX_NUMS 5
int Print(int inputValues[]);

int main(void) {
    int mean;
    int nums[MAX_NUMS];

    printf(“Enter %d nums,\n”, MAX_NUMS);
    for (int i =0; index < MAX_NUMS; index++) {
        printf(“Input num %d: “, i);
        scanf(“%d”, &nums[i]);
    }
    mean = Average(nums);
    printf(“The average of these nums is %d\n”, mean);

    return 0;
}
```

```
int Average(int inputValues[]) {
    int sum = 0;

    for (int i=0; i < MAX_NUMS; i++) {
        sum += inputValues[i];
    }

    return (sum / MAX_NUMS);
}
```

InputValues becomes nums (== &nums[0])

All elements of **nums** can be accessed by using **InputValues**

Array – String (Array of Characters)

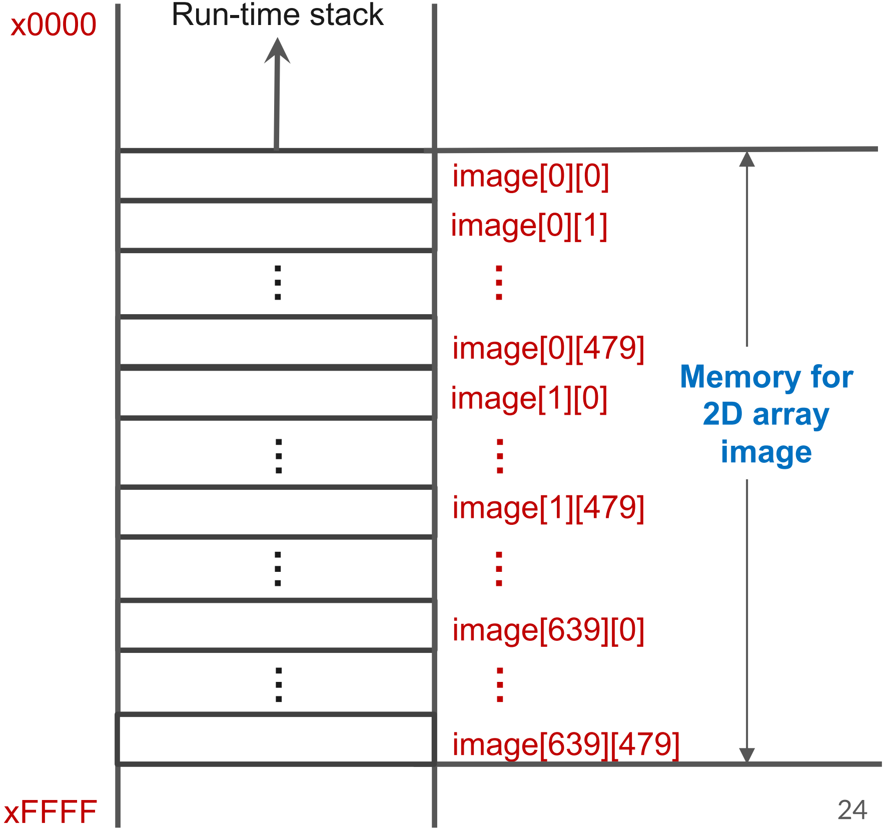
- Strings in C are simply arrays of character type
  - char word[10];
  - To store “winter” (6 characters) in word, we need to mark where the string ends
    - word[0] = ‘w’;   word[1] = ‘i’;   word[2] = ‘n’;   word[3] = ‘t’;   word[4] = ‘e’;   word[5] = ‘r’;   word[7] = **‘\0’**;
  - **\0** is the special character sequence that indicates the null character whose ASCII value is 0
    - Serves as a **sentinel** that identifies the end of a string
    - We must reserve one element for the null character, and
    - Thus, **word** can store a string comprising up to 9 characters
  - printf(“%s”, word);   // should print **winter, %s** is the format specification for string
- Strings can also be initialized within their declarations
  - char word[10] = “winter”;   printf(“%s”, word);
  - Single quotes ‘ ’ for one character, double quotes “ ” for a string
  - The null character \0 is automatically added to the end of the string

Array – Multi-dimensional Array

- 2D array – int values[ROWS][COLS];
  - Useful for processing an image (e.g., pixels in a 640x480 image)
  - All columns in a row are grouped and allocated in memory like an array

- Example
  - int image[640][480];

- C supports arrays of more dimensions
  - Consecutive elements of the rightmost index are allocated sequentially in memory



Array – Variable-length Arrays

- Array size can be a variable
    - int functionA(int len) {
    -     int values[len];
    -     …
    - }

- The size of values (len) is not known at compile time
    - In this case, C uses a different type of allocation scheme, which is out of scope of this course

- It is sometimes convenient to use variable-length arrays, which sacrifices performance due to the use of a more complex memory allocation scheme

Array – Warning

- C does not provide protection against exceeding the size of an array
    - No compile error from the following codes
        - int values[10];
        - values[13] = 10;
    - Memory objects outside of the array can be corrupted, resulting in unintended behaviors
        - One of the most common errors in C

- We often use a variable as an index for an array, such as values[i]
    - We must make sure if i is between 0 and values' size

Questions?

Thanks!