Programming with C I

Fangtian Zhong
CSCI 112

Gianforte School of Computing
Norm Asbjornson College of Engineering
E-mail: fangtian.zhong@montana.edu

2024.02.14

Objectives

- To learn how to declare and use arrays for storing collections of values of the same type
- To understand how to use a subscript to reference the individual values in an array
- To learn how to process the elements of an array in sequential order using loops
- To understand how to pass individual array elements and entire arrays through function arguments

Objectives

- To learn a method for searching an array
- To learn a method for sorting an array
- To learn how to use multidimensional arrays for storing tables of data
- To understand the concept of parallel arrays
- To learn how to declare and use your own data types

Basic Terminology

data structure

- a composite of related data items stored under the same name

array

- a collection of data items of the same type

Declaring and Referencing Arrays

array element

- a data item that is part of an array

subscripted variable

- a variable followed by a subscript in brackets, designating an array element

array subscript

- a value or expression enclosed in brackets after the array name, specifying which array element to access

Declaring and Referencing Arrays

double x[8];

Array x

x[0]	x[1]	x[2]	x[3]	x[4]	x[5]	x[6]	x[7]
16.0	12.0	6.0	8.0	2.5	12.0	14.0	-54.5

Array Initialization

```
int prime_lt_100[] = {2, 3, 5, 7, 11, 13, 17, 19,
             23, 29, 31, 37, 41, 43, 47, 53, 59, 61,
             67, 71, 73, 79, 83, 89, 97}
```

```
char vowels[] = {‘a’, ‘e’, ‘i’, ‘o’, ‘u’, ‘y’}
```

Using for Loops for Sequential Access

```
for (i = 0; i < SIZE; ++i)
        square[i] = i * i;
```

Array square

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]
0	1	4	9	16	25	36	49	64	81	100

Table Statements That Manipulate Array x

Statement	Explanation
printf(“%.1f, x[0]);	Displays the value of x[0], which is 16.0.
x[3] = 25.0;	Stores the value 25.0 in x[3].
sum = x[0] + x[1];	Stores the sum of x[0] and x[1], which is 28.0 in the variable sum.
sum += x[2]	Adds x[2] to sum. The new sum is 34.0.
x[3] += 1.0;	Adds 1.0 to x[3]. The new x[3] is 26.0;
x[2] = x[0] + x[1];	Stores the sum of x[0] and x[1] in x[2]. The new x[2] is 28.0.

Array x

x[0]	**x[1]**	**x[2]**	**x[3]**	**x[4]**	**x[5]**	**x[6]**	**x[7]**
16.0	12.0	28.0	26.0	2.5	12.0	14.0	-54.5

Array Subscripts

- **Syntax:**

  aname [subscript]

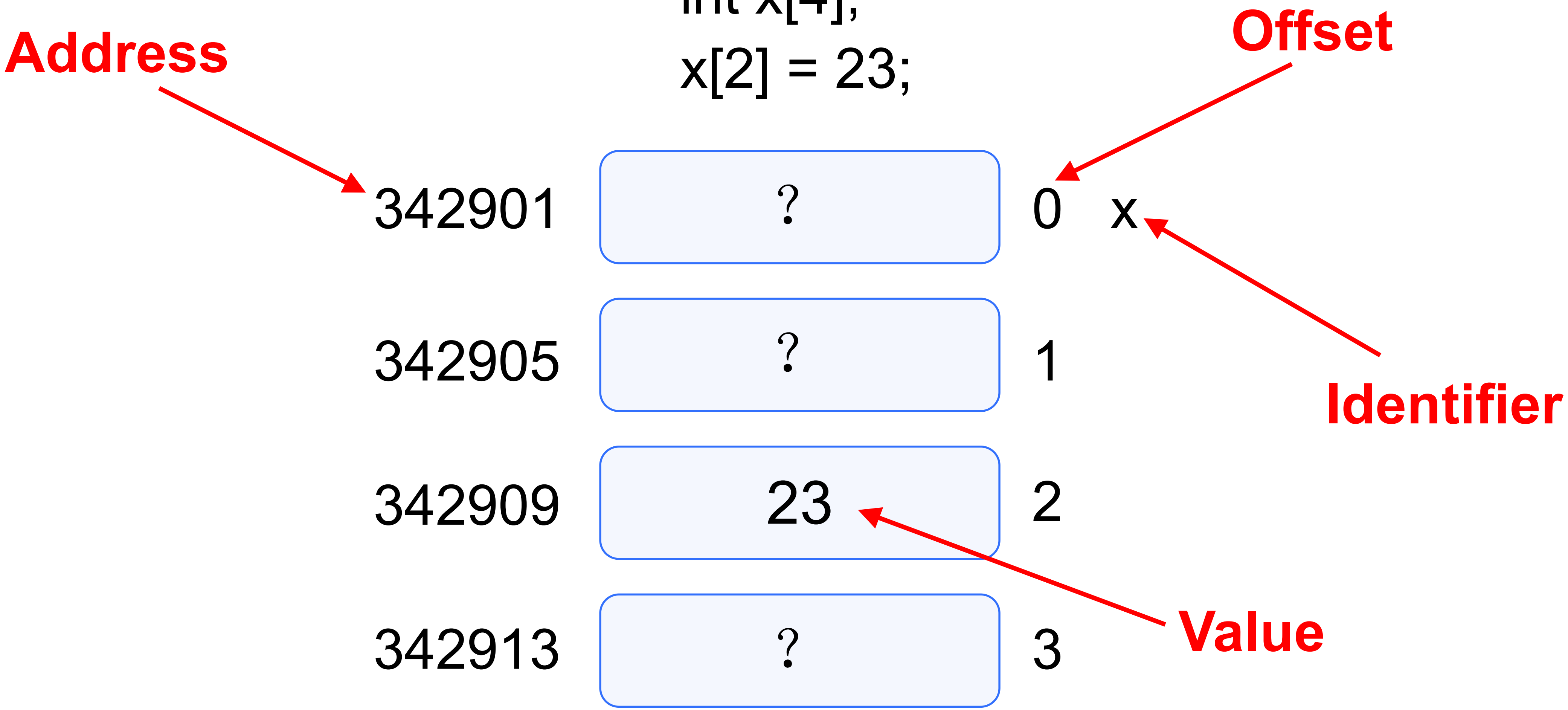
- **Examples:**

```
x[3]
x[i + 1]
```

Array x

x[0]	x[1]	x[2]	x[3]	x[4]	x[5]	x[6]	x[7]
16.0	12.0	6.0	8.0	2.5	12.0	14.0	-54.5

```
int x[4];
x[2] = 23;
```



Address

Offset

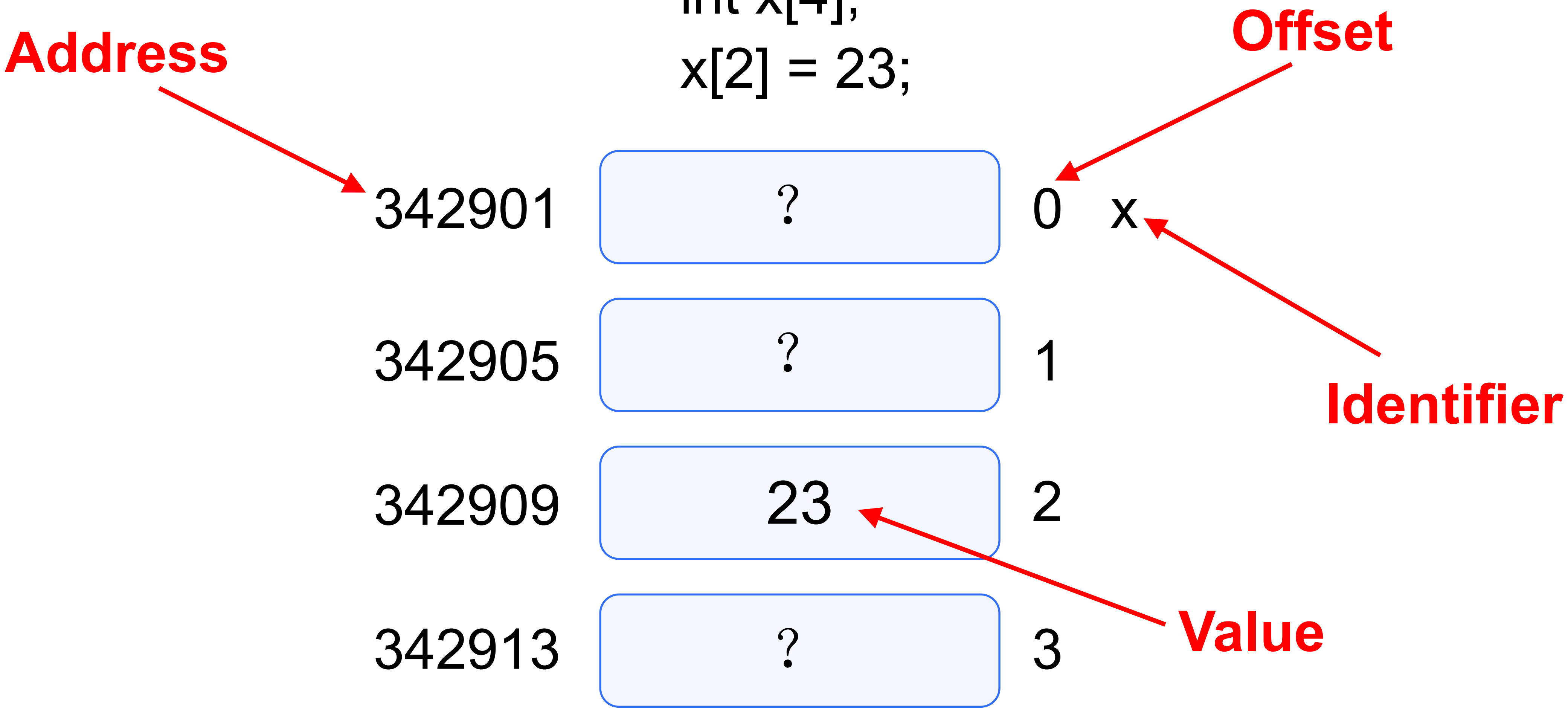
Identifier

Value

342901	?	0	x
342905	?	1	
342909	23	2	
342913	?	3	

What’s at x[5]?

```
int x[4];
x[2] = 23;
```



342901	?	0	x
342905	?	1	
342909	23	2	
342913	?	3	

Partially Filled Arrays

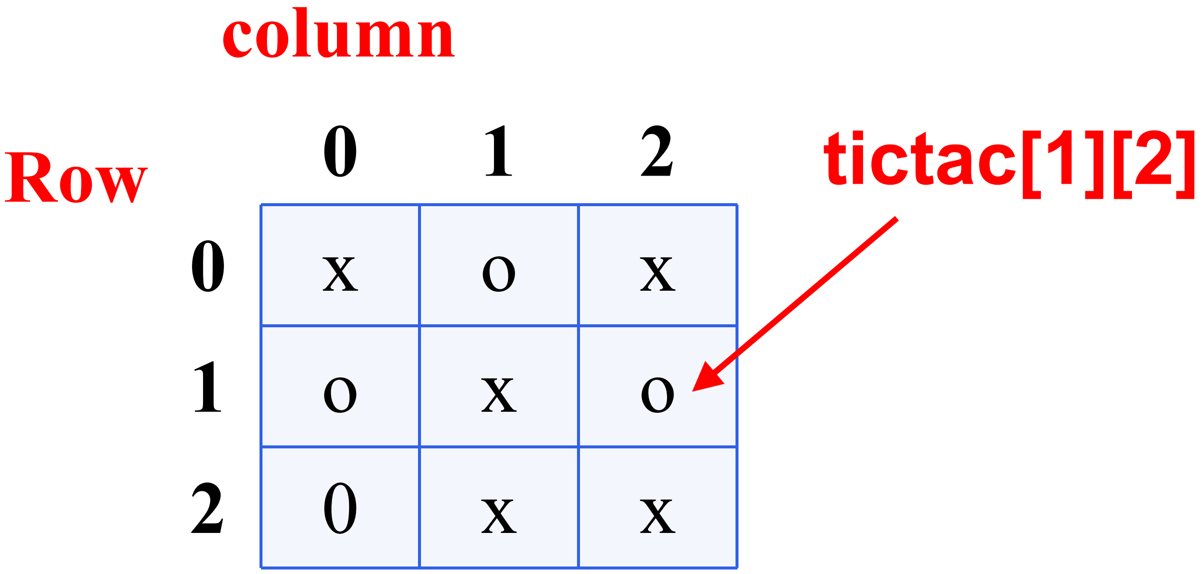
- A program may need to process many lists of similar data but the lists may not all be the same length.
- In order to reuse an array for processing more than one data set, you can declare an array large enough to hold the largest data set anticipated.
- Then your program should keep track of how many array elements are actually in use.

Multidimensional Arrays

multidimensional array

```
type arr_name[dim1val][dim2val]
tictac[3][3]
```

Figure A Tic-tac-toe Board Stored as Array tictac



column

Row

tictac[1][2]

	0	1	2
0	x	o	x
1	o	x	o
2	0	x	x

Using Array Elements as Function Arguments

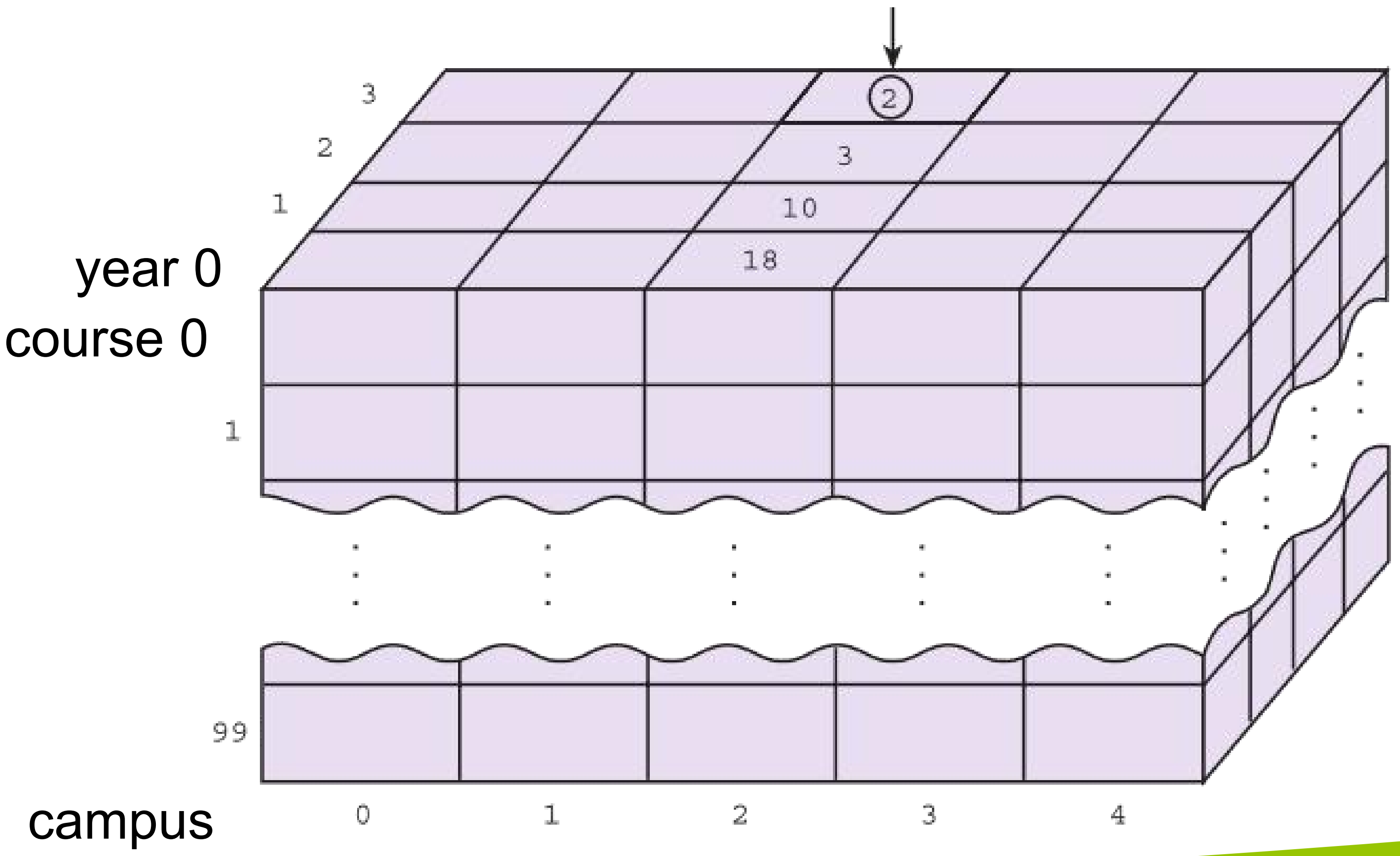
```
scanf(“%lf”, &x[i]);
```

Figure Function to Check Whether Tic-tac-toe Board is Filled

```
/* Check Whether a tic-tac-toe is completely filled.          */
int                                                            */
filled(char ttt_brd[3][3])        /* input  -tic-tac-toe board
{
       int r, c,     /* row ad column subscripts      */
           ans；     /* whether or not board filled   */

       / * Assumes board is filled until blank is found          */
       for (r = 0; r < 3; ++r)
           for (c = 0; c < 3; ++c)
               if (ttt_brd[r][c] == ‘ ‘)
                   ans = 0;

       return (ans);
}
```

Figure Three-Dimensional Array enroll

Number of seniors (year 3) taking course 0 at campus 2 enroll [0][2][3]



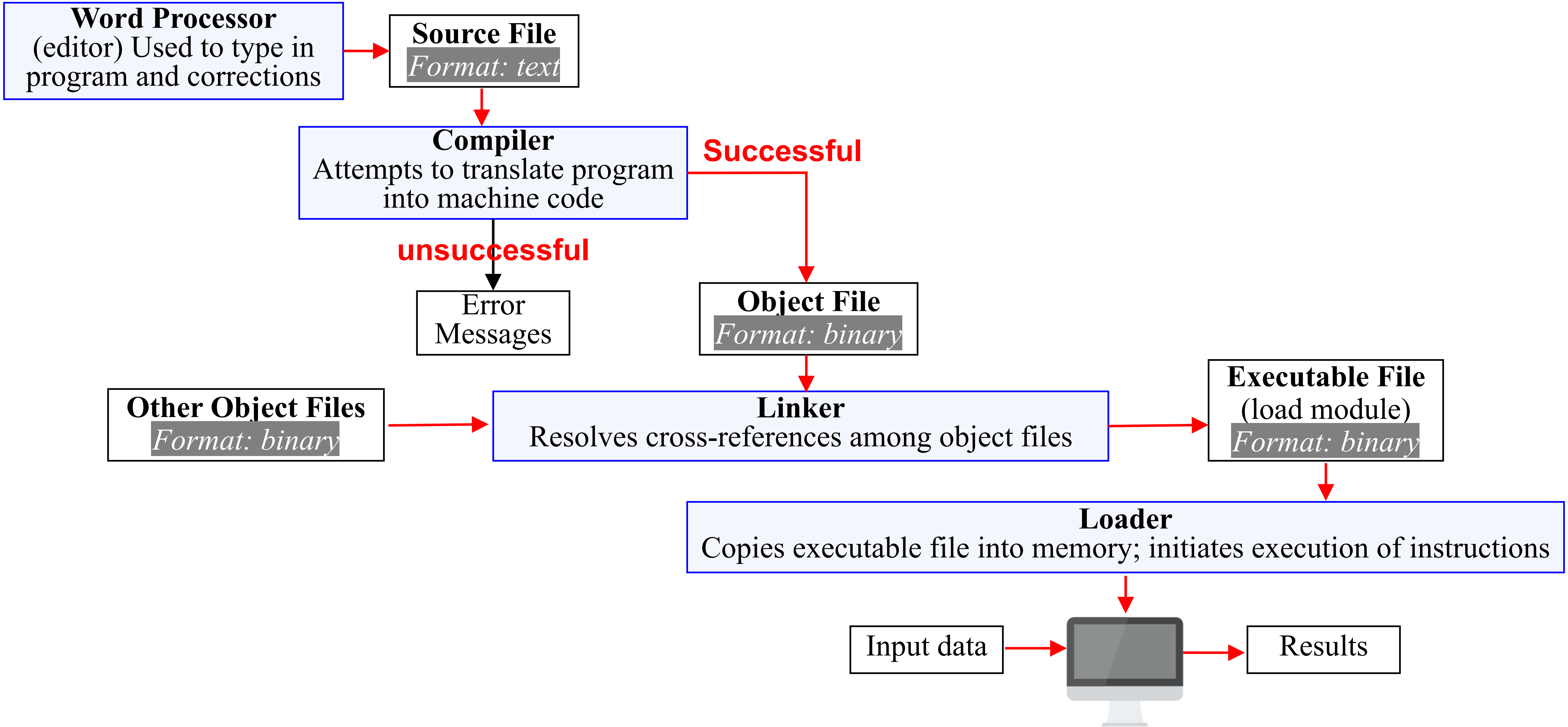
Array Arguments

- **We can write functions that have arrays as arguments.**

- **Such functions can manipulate some, or all, of the elements corresponding to an actual array argument.**

Variable scope

- Part of a program where a variable is accessible
- Lifetime of a variable



Word Processor
(editor) Used to type in program and corrections

Source File
Format: text

Compiler
Attempts to translate program into machine code

Successful

unsuccessful

Error Messages

Object File
Format: binary

Other Object Files
Format: binary

Linker
Resolves cross-references among object files

Executable File
(load module)
Format: binary

Loader
Copies executable file into memory; initiates execution of instructions

Input data

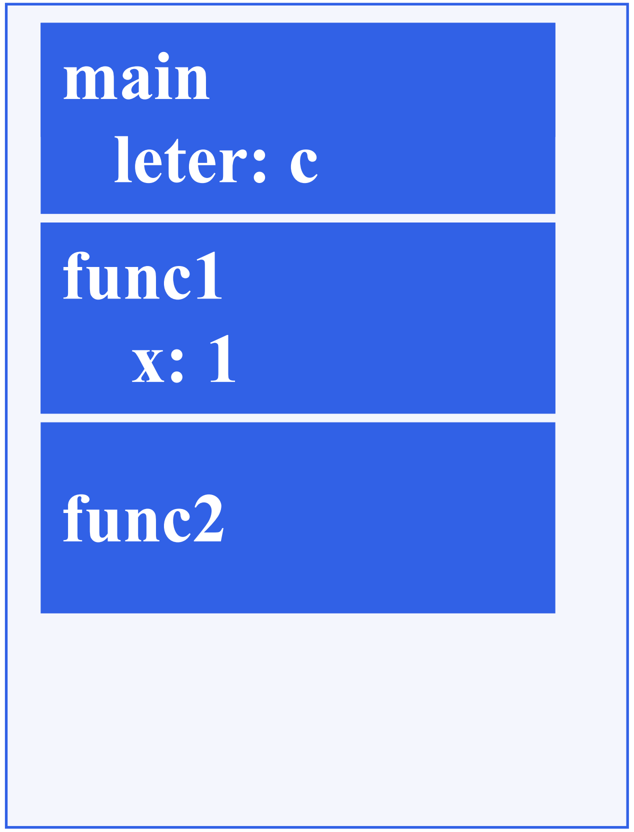
Results

What happens when we run our executable file?

```
func2() {
    printf(“%d\n”, x);
}
func1() {
    int x = 1;
    func2();
}
int main(void) {
    char letter=’c’
    func1();
}
```

out of scope!

Memory


main
leter: c

func1
x: 1

func2

What happens when we run our executable file?

```
void fill_array(
        int list[],
        int n,
        int in_value) {
    int i;
    for (i = 0;
         i < n; ++i) {
        list[i] = in_value;
    }
}
int main(void) {
    int arr[10];
    fill_array(arr, 5, 1);
}
```

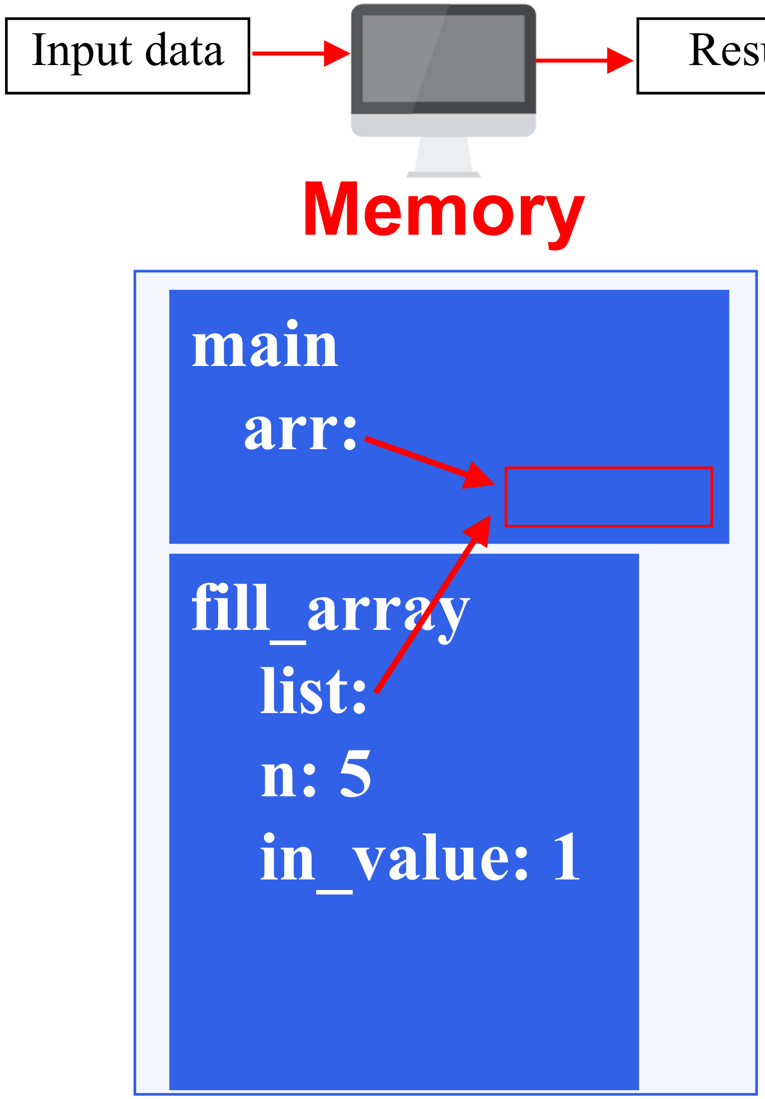
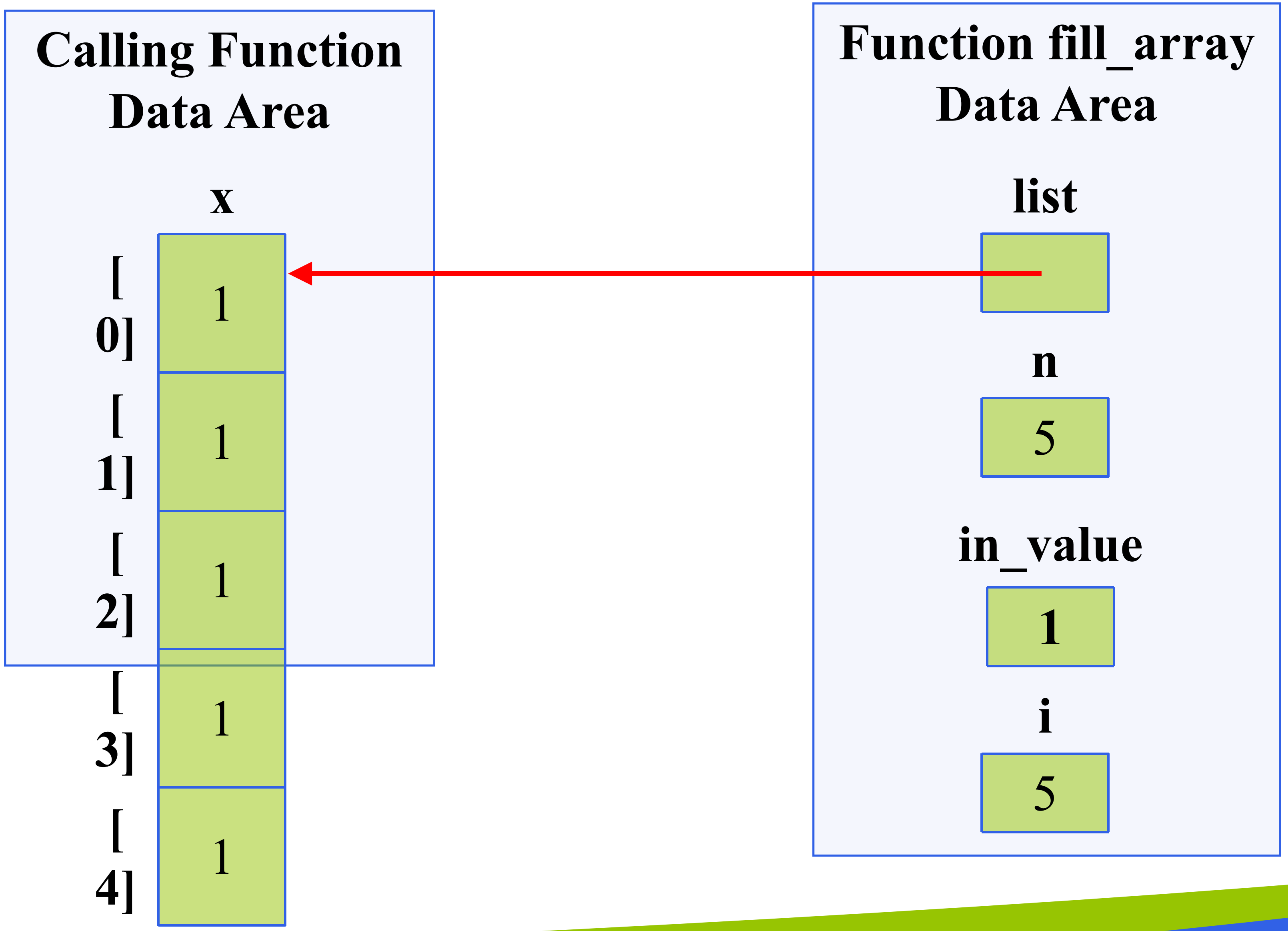


Figure Function fill_array

```
/*
 * Set all elements of its array parameter to in_value.
 * Pre: n and in_value are defined.
 * Post: list[i] = in_value, for 0 <= i < n.
 */
void
fill_array (int list[],       /* output - list of n integers        */
            int n,            /* input - number of list elements    */
            int in_value)     /* input - initial value              */
{
      int i;       /* array subscripts and loop control             */

      for (i = 0; i < n; ++i)
          list[i] = in_value;
}
```

Figure Data Areas Before Return from fill_array (x, 5, 1);



Arrays as Input Arguments

- The qualifier const allows the compiler to mark as an error any attempt to change an array element within the function.

Figure Function to Find the Largest Element in an Array

```
/*
 * Return the largest of the first n values in array list
 * Pre: First n elements of array list are defined and n > 0
 */
int
get_max(const int list[],    /* input - list of n integers                      */
              int        n)       /* input - number of list elements to examine     */
{
          int i,
              cur_large;          /* largest value so far                              */

           / * Initial array element is largest so far                          */
           cur_large = list[0];

           /* Compare each remaining list element to the largest so far;
               save the larger
           for (i = 1; i < n; ++i)
               if (list[i] > cur_large)
                     cur_large = list[i]

           return (cur_large);
}
```

Returning an Array Result

- In C, it is not legal for a function’s return type to be an array.
- You need to use an output parameter to send your array back to the calling module.

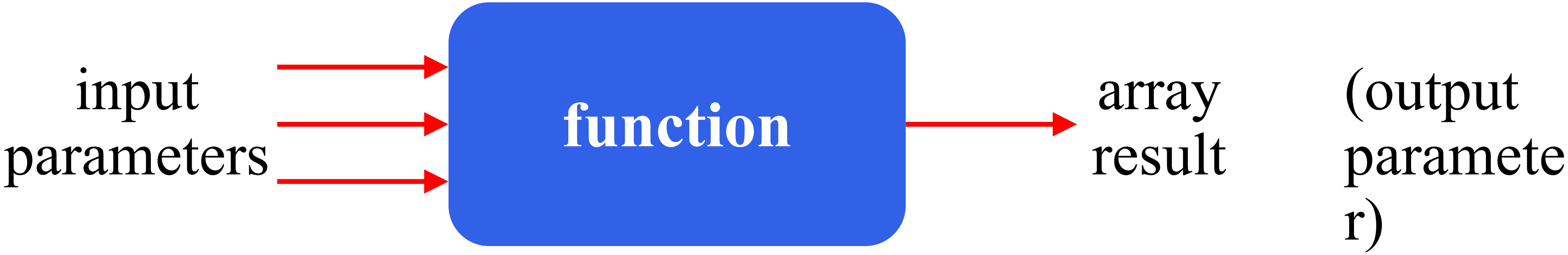
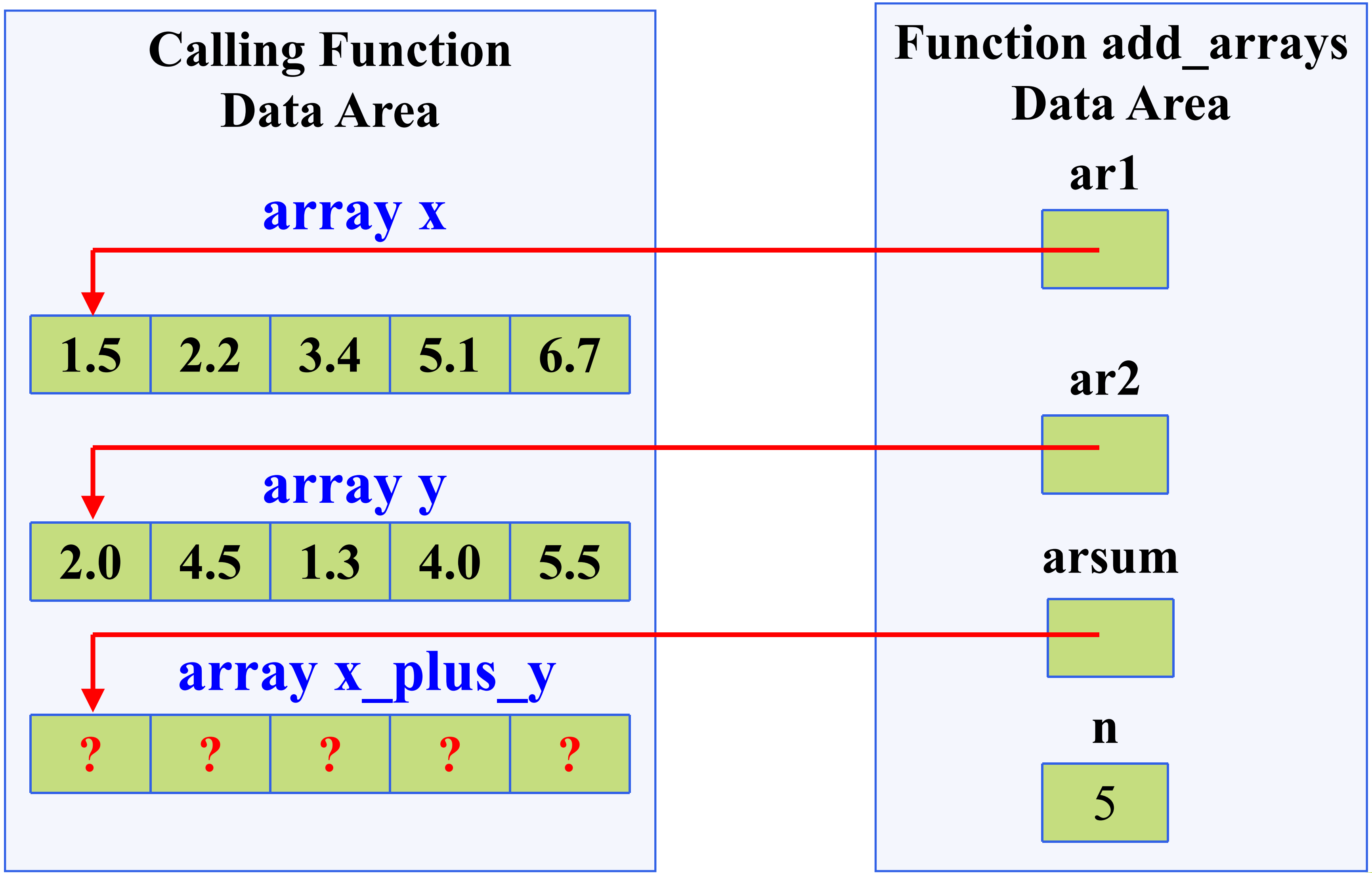


Diagram of a function That Computes an Array Result

Figure Function to Add Two Arrays

```
/*
 * Adds corresponding elements of arrays ar1 and ar2, storing the result in arsum.
 * Processes first n elements only.
 * Pre: First n elements of ar1 and ar2 are defined. arsum’s corresponiding actual
        argument has a declared size >= n (n >= 0)
 */
void
add_arrays(const double ar1[],          /* input -                              */
           const double ar2[],          /* arrays being added                   */
           double       arsum[],        /* output - sum of corresponding        */
                                               elements of ar1 and ar2
           int          n)              /* input - number of element
                                                  paris summed                  */

{
        int i,

         / * Adds corresponing elements of ar1 and ar2                          */
         for (i = 0; i < n; ++i)
             arsum[i] = ar[i] + ar2[i];
}
```

Figure Function Data Areas for add_arrays(x, y, x_plus_y, 5);



Array Search

- Assume the target has not been found.
- Start with the initial array element.
- repeat while the target is not found and there are more array elements.

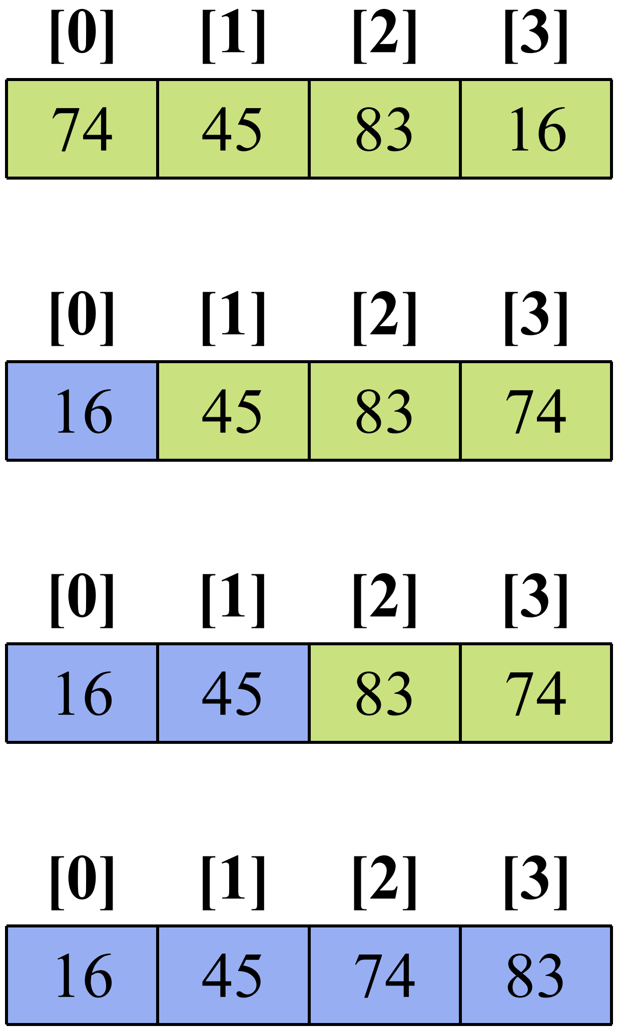
```
➤ if the current element matches the target
    •  Set a flag to indicate that the target has been found
  else
    •  Advance to the next array element.

➤ if the target was found
    •  Return the target index as the search result
  else
    •  Return -1 as the search result.
```

Selection Sort

🏅 **for each value of fill from 0 to n-2**

- Find index_of_min, the index of the smallest element in the unsorted subarray list[fill] through list[n-1]
- if fill is not the position of the smallest element (index_of_min)
  - ➢ Exchange the smallest element with the one at position fill.

Figure Trace of Selection Sort



[0]	[1]	[2]	[3]
74	45	83	16

[0]	[1]	[2]	[3]
16	45	83	74

[0]	[1]	[2]	[3]
16	45	83	74

[0]	[1]	[2]	[3]
16	45	74	83

- fill is 0. Find smallest element in subarray list[1] through list[3] and swap it with list[0].
- fill is 1. Find the smallest element in subarray list[1] through list[3] - no exchange needed.
- fill is 2. Find the smallest elment in subarray list[2] through list[3] and swap it with list [2].

Wrap Up

- A data structure is a grouping of related data items in memory.
- An array is a data structure used to store a collection of data items of the same type.

Conditional operator

- **A very compact if-else.**
- **condition ? expression2 : expression3**

means

```
if (condition)
        expression2
else
        expression3
```

THE END

Fangtian Zhong
CSCI 112

Gianforte School of Computing
Norm Asbjornson College of Engineering
E-mail: fangtian.zhong@montana.edu

2024.02.14