



Lecture 2 [CPE112]

ARRAY & STRUCTURE

Lecturer: Dr. Piyanit Wepulanon & Dr. Taweechai Nuntawisuttiwong

Original version by: Assoc. Prof. Dr. Natasha Dejdumrong



OUTLINES

- ★ Introduction to Array
- ★ Declaration of Array
- ★ Accessing the Elements of Array
- ★ Storing Values in Array
- ★ Operations on Array
- ★ Pointer & Array
- ★ 2D Array
- ★ String
- ★ Structure
- ★ Union

ARRAY

Marks1	Marks5	Marks9	Marks13	Marks17
<input type="text"/>				
Marks2	Marks6	Marks10	Marks14	Marks18
<input type="text"/>				
Marks3	Marks7	Marks11	Marks15	Marks19
<input type="text"/>				
Marks4	Marks8	Marks12	Marks16	Marks20
<input type="text"/>				

Figure 3.1 Twenty variables for 20 students

ARRAY

- ★ An array is a collection of **similar data elements**.
- ★ These data elements have the **same data type**.
- ★ The elements of the array are stored in **consecutive memory locations** and are **referenced by an index** (also known as the subscript).
- ★ The subscript is an ordinal number which is used to **identify an element of the array**.

Declaration of Array

★ Arrays are declared using the following syntax:

```
type name[size];
```

```
int marks[10];
```

1 st element	2 nd element	3 rd element	4 th element	5 th element	6 th element	7 th element	8 th element	9 th element	10 th element
----------------------------	----------------------------	----------------------------	----------------------------	----------------------------	----------------------------	----------------------------	----------------------------	----------------------------	-----------------------------

marks[0] marks[1] marks[2] marks[3] marks[4] marks[5] marks[6] marks[7] marks[8] marks[9]

Figure 3.2 Memory representation of an array of 10 elements

Declaration of Array

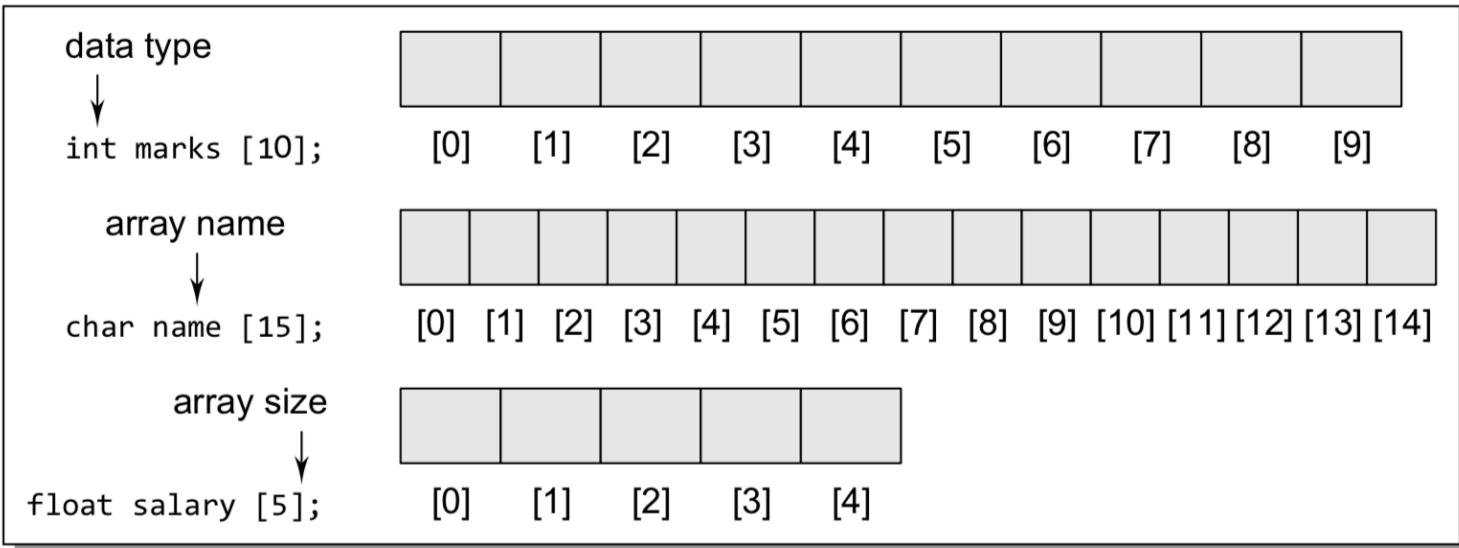


Figure 3.3 Declaring arrays of different data types and sizes

Declaration of Array

A

```
int marks[10];
```

B

```
int ARRSIZE = 10;  
int* marks = calloc(ARRSIZE,sizeof(int));
```

Accessing the Elements of Array

- ★ Arrays can be accessed using the following syntax:

```
// Set each element of the array to -1
int i, marks[10];
for(i=0;i<10;i++)
    marks[i] = -1;
```

Figure 3.4 Code to initialize each element of the array to -1

-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]

Figure 3.5 Array marks after executing the code given in Fig. 3.4

Calculating the address of array elements

★ The formula to perform this calculation is,

Address of data element, $A[k] = BA(A) + w(k - \text{lower_bound})$

Here, A is the array, k is the index of the element of which we have to calculate the address, BA is the base address of the array A , and w is the size of one element in memory, for example, size of int is 2.

Calculating the address of array elements

Address of data element, $A[k] = BA(A) + w(k - \text{lower_bound})$

Example 3.1 Given an array `int marks[] = {99, 67, 78, 56, 88, 90, 34, 85}`, calculate the address of `marks[4]` if the base address = 1000.

Solution

99	67	78	56	88	90	34	85
marks[0]	marks[1]	marks[2]	marks[3]	marks[4]	marks[5]	marks[6]	marks[7]
1000	1002	1004	1006	1008	1010	1012	1014

We know that storing an integer value requires 2 bytes, therefore, its size is 2 bytes.

$$\begin{aligned}\text{marks[4]} &= 1000 + 2(4 - 0) \\ &= 1000 + 2(4) = 1008\end{aligned}$$

Calculating the Length of an Array

The length of an array is given by the number of elements stored in it. The general formula to calculate the length of an array is

$$\text{Length} = \text{upper_bound} - \text{lower_bound} + 1$$

where `upper_bound` is the index of the last element and `lower_bound` is the index of the first element in the array.

Calculating the Length of an Array

Example 3.2 Let $\text{Age}[5]$ be an array of integers such that

$\text{Age}[0] = 2, \text{Age}[1] = 5, \text{Age}[2] = 3, \text{Age}[3] = 1, \text{Age}[4] = 7$

Show the memory representation of the array and calculate its length.

Solution

The memory representation of the array $\text{Age}[5]$ is given as below.

2	5	3	1	7
$\text{Age}[0]$	$\text{Age}[1]$	$\text{Age}[2]$	$\text{Age}[3]$	$\text{Age}[4]$

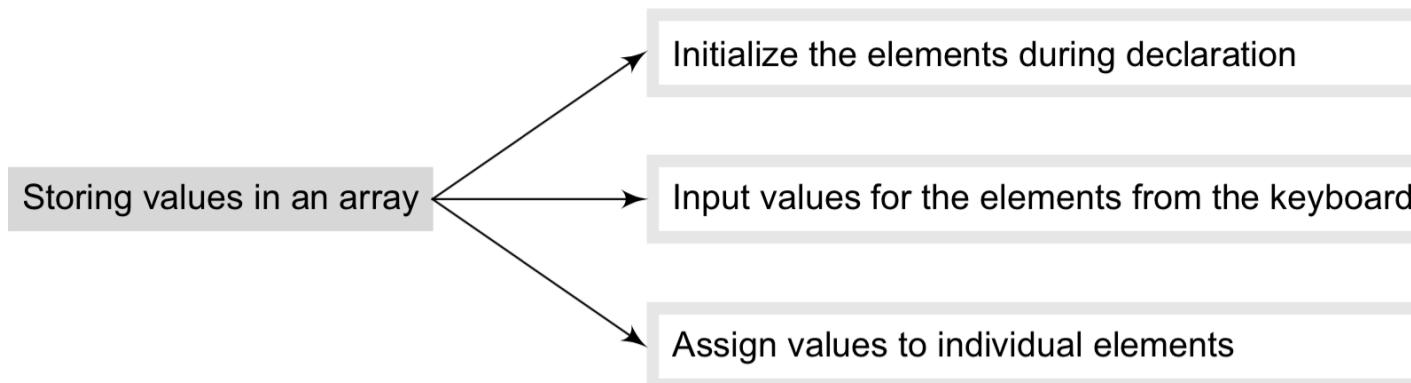
$$\text{Length} = \text{upper_bound} - \text{lower_bound} + 1$$

Here, $\text{lower_bound} = 0$, $\text{upper_bound} = 4$

Therefore, $\text{length} = 4 - 0 + 1 = 5$

Storing Values in Arrays

- ★ When we declare an array, we are just allocating space for its elements; no values are stored in the array.
- ★ There are three ways to store values in an array.



Initializing Arrays during Declaration

★ The elements of an array can be initialized at the time of declaration, just as any other variable. When an array is initialized, we need to provide a value for every element in the array.

marks[0]	90
marks[1]	82
marks[2]	78
marks[3]	95
marks[4]	88

```
int marks[5]={90, 82, 78, 95, 88};
```

Figure 3.7 Initialization of array marks[5]

Initializing Arrays during Declaration

```
int marks [5] = {90, 45, 67, 85, 78};
```

90	45	67	85	78
[0]	[1]	[2]	[3]	[4]

```
int marks [5] = {90, 45};
```

90	45	0	0	0
[0]	[1]	[2]	[3]	[4]

Rest of the elements are filled with 0's

```
int marks [] = {90, 45, 72, 81, 63, 54};
```

90	45	72	81	63	54
[0]	[1]	[2]	[3]	[4]	[5]

```
int marks [5] = {0};
```

0	0	0	0	0
[0]	[1]	[2]	[3]	[4]

Figure 3.8 Initialization of array elements

Inputting Values from Keyboard

```
int i, marks[10];
for(i=0;i<10;i++)
    scanf("%d", &marks[i]);
```

Figure 3.9 Code for inputting each element of the array

Assigning Values to Individual Elements

```
int i, arr1[10], arr2[10];
arr1[10] = {0,1,2,3,4,5,6,7,8,9};
for(i=0;i<10;i++)
    arr2[i] = arr1[i];
```

Figure 3.10 Code to copy an array at the individual element level

```
// Fill an array with even numbers
int i,arr[10];
for(i=0;i<10;i++)
    arr[i] = i*2;
```

Figure 3.11 Code for filling an array with even numbers

Operations on Arrays

★ There are a number of operations that can be preformed on arrays. These operations include:

- ★ Traversing an array
- ★ Inserting an element in an array
- ★ Searching an element in an array
- ★ Deleting an element from an array
- ★ Merging two arrays
- ★ Sorting an array in ascending or descending order.

Traversing an Array

- ★ Traversing the data elements of an array, A, can include printing every element, counting the total number of elements, or performing any process on these elements.
- ★ Since, array is a linear data structure (because all its elements form a sequence), traversing its elements is very simple and straightforward.

```
Step 1: [INITIALIZATION] SET I = lower_bound
Step 2: Repeat Steps 3 to 4 while I <= upper_bound
Step 3:      Apply Process to A[I]
Step 4:      SET I = I + 1
            [END OF LOOP]
Step 5: EXIT
```

Figure 3.12 Algorithm for array traversal

Traversing an Array

★ Write a program to read and display n numbers using an array.

```
#include <stdio.h>
#include <conio.h>
int main() {
    int i, n, arr[20];
    clrscr();
    printf("\n Enter the number of elements in the array : ");
    scanf("%d", &n);

    for(i=0;i<n;i++) {

        printf("\n arr[%d] = ", i);
        scanf("%d",&arr[i]);
    }
    printf("\n The array elements are ");
    for(i=0;i<n;i++)
        printf("\t %d", arr[i]);
    return 0;
}
```

Output

```
Enter the number of elements in the array : 5
arr[0] = 1
arr[1] = 2
arr[2] = 3
arr[3] = 4
arr[4] = 5
The array elements are      1   2   3   4   5
```

Traversing an Array

★ Write a program to find the mean of n numbers using arrays.

```
#include <stdio.h>
#include <conio.h>
int main() {
    int i, n, arr[20], sum = 0;
    float mean = 0.0;
    clrscr();
    printf("\n Enter the number of elements in the array : ");
    scanf("%d", &n);
    for(i=0;i<n;i++)
    {
        printf("\n arr[%d] = ", i);
        scanf("%d",&arr[i]);
    }
    for(i=0;i<n;i++)
        sum += arr[i];
    mean = (float)sum/n;
    printf("\n The sum of the array elements = %d", sum);
    printf("\n The mean of the array elements = %.2f", mean);
    return 0;
}
```

Output

```
Enter the number of elements in the array : 5
arr[0] = 1
arr[1] = 2
arr[2] = 3
arr[3] = 4
arr[4] = 5
The sum of the array elements = 15
The mean of the array elements = 3.00
```

Inserting an Element in an Array

★ If an element has to be inserted at the end of an existing array, then the task of insertion is quite simple.

```
Step 1: Set upper_bound = upper_bound + 1
Step 2: Set A[upper_bound] = VAL
Step 3: EXIT
```

Figure 3.13 Algorithm to append a new element to an existing array

Insert an Element in the Middle of an Array

★ If an element has to be inserted in the middle of an existing array, then

45	23	34	12	56	20
Data[0]	Data[1]	Data[2]	Data[3]	Data[4]	Data[5]

45	23	34	100	12	56	20
Data[0]	Data[1]	Data[2]	Data[3]	Data[4]	Data[5]	Data[6]

Insert an Element in the Middle of an Array

- (a) A, the array in which the element has to be inserted
- (b) N, the number of elements in the array
- (c) POS, the position at which the element has to be inserted
- (d) VAL, the value that has to be inserted

```
Step 1: [INITIALIZATION] SET I = N
Step 2: Repeat Steps 3 and 4 while I >= POS
Step 3:           SET A[I + 1] = A[I]
Step 4:           SET I = I - 1
                  [END OF LOOP]
Step 5: SET N = N + 1
Step 6: SET A[POS] = VAL
Step 7: EXIT
```

Figure 3.14 Algorithm to insert an element in the middle of an array.

Insert an Element in the Middle of an Array

```
Step 1: [INITIALIZATION] SET I = N
Step 2: Repeat Steps 3 and 4 while I >= POS
Step 3:     SET A[I + 1] = A[I]
Step 4:     SET I = I - 1
    [END OF LOOP]
Step 5: SET N = N + 1
Step 6: SET A[POS] = VAL
Step 7: EXIT
```

Figure 3.14 Algorithm to insert an element in the middle of an array.

Initial Data[] is given as below.

45	23	34	12	56	20
Data[0]	Data[1]	Data[2]	Data[3]	Data[4]	Data[5]

Calling `INSERT(Data, 6, 3, 100)` will lead to the following processing in the array:

45	23	34	12	56	20	20
Data[0]	Data[1]	Data[2]	Data[3]	Data[4]	Data[5]	Data[6]

45	23	34	12	56	56	20
Data[0]	Data[1]	Data[2]	Data[3]	Data[4]	Data[5]	Data[6]

45	23	34	12	12	56	20
Data[0]	Data[1]	Data[2]	Data[3]	Data[4]	Data[5]	Data[6]

45	23	34	100	12	56	20
Data[0]	Data[1]	Data[2]	Data[3]	Data[4]	Data[5]	Data[6]

Deleting an Element from an Array

★ Deleting an element from an array means removing a data element from an already existing array. If the element has to be deleted from the end of the existing array, then the task of deletion is quite simple.

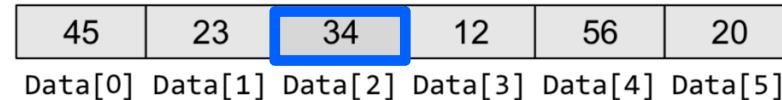
```
Step 1: SET upper_bound = upper_bound - 1  
Step 2: EXIT
```

Figure 3.15 Algorithm to delete the last element of an array

Deleting an Element from an Array

★ If the element has to be deleted from the middle of the existing array, then

.....



Delete an Element from the Middle of Array

The algorithm **DELETE** will be declared as **DELETE(A, N, POS)**. The arguments are:

- (a) **A**, the array from which the element has to be deleted
- (b) **N**, the number of elements in the array
- (c) **POS**, the position from which the element has to be deleted

```
Step 1: [INITIALIZATION] SET I = POS
Step 2: Repeat Steps 3 and 4 while I <= N - 1
Step 3:           SET A[I] = A[I + 1]
Step 4:           SET I = I + 1
                  [END OF LOOP]
Step 5: SET N = N - 1
Step 6: EXIT
```

Figure 3.16 Algorithm to delete an element from the middle of an array

Delete an Element from the Middle of Array

```
Step 1: [INITIALIZATION] SET I = POS
Step 2: Repeat Steps 3 and 4 while I <= N - 1
Step 3:           SET A[I] = A[I + 1]
Step 4:           SET I = I + 1
              [END OF LOOP]
Step 5: SET N = N - 1
Step 6: EXIT
```

Figure 3.16 Algorithm to delete an element from the middle of an array

45	23	34	12	56	20
Data[0]	Data[1]	Data[2]	Data[3]	Data[4]	Data[5]
45	23	12	12	56	20
Data[0]	Data[1]	Data[2]	Data[3]	Data[4]	Data[5]
45	23	12	56	56	20
Data[0]	Data[1]	Data[2]	Data[3]	Data[4]	Data[5]
45	23	12	56	20	20
Data[0]	Data[1]	Data[2]	Data[3]	Data[4]	Data[5]
45	23	12	56	20	
Data[0]	Data[1]	Data[2]	Data[3]	Data[4]	

Figure 3.17 Deleting elements from an array

Merging two Arrays

★ Merging two arrays in a third array means first copying the contents of the first array into the third array and then copying the contents of the second array into the third array.

★ Hence, the merged array contains the contents of the first array followed by the contents of the second array.

Array 1-	90	56	89	77	69
----------	----	----	----	----	----

Array 2-	45	88	76	99	12	58	81
----------	----	----	----	----	----	----	----

Array 3-	90	56	89	77	69	45	88	76	99	12	58	81
----------	----	----	----	----	----	----	----	----	----	----	----	----

Figure 3.18 Merging of two unsorted arrays

Merging two Arrays

- ★ If the arrays are unsorted, then merging the arrays is very simple, as one just needs to copy the contents of one array into another.
- ★ But merging is not a trivial task when the two arrays are sorted and the merged array also needs to be sorted.

Array 1-	20	30	40	50	60
----------	----	----	----	----	----

Array 2-	15	22	31	45	56	62	78
----------	----	----	----	----	----	----	----

Array 3-	15	20	22	30	31	40	45	50	56	60	62	78
----------	----	----	----	----	----	----	----	----	----	----	----	----

Passing Array to Functions

★ Like variables of other data types, we can also pass an array to a function.

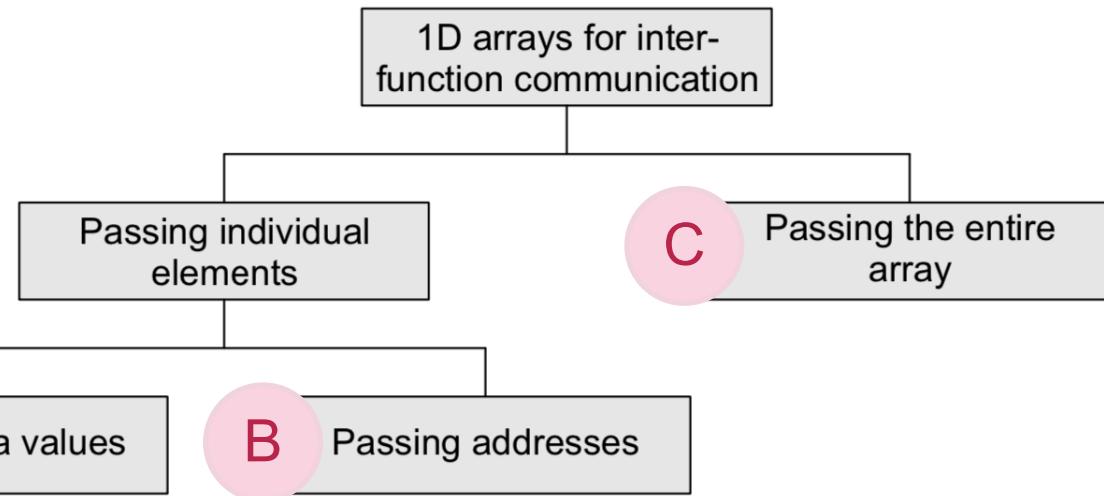


Figure 3.20 One dimensional arrays for inter-function communication

Passing Array to Functions

```
int main()
{
    int arr[5] = {1,2,3,4,5};
    func1(arr[3]);
    func2(&arr[3]);
    func3(arr);
    return 0;
}
```

Address	Variable	Value
1000	arr[0]	1
1001	arr[1]	2
1002	arr[2]	3
1003	arr[3]	4
1004	arr[4]	5

A

```
void func1(int num)
{
    printf("%d", num);
}
```

B

```
void func2(int *num)
{
    printf("%d", *num);
}
```

C

```
void func3(int arr[])
{
    int i;
    for(i=0;i<5;i++)
        printf("%d", arr[i]);
}
```

Pointer & Array

Address	Variable	Value
1000		
1001	arr[0]	1
1002		
1003	arr[1]	2
1004		
1005	arr[2]	3
1006		
1007	arr[3]	4
1008		
1009	arr[4]	5
1010		
1011		
1012	ptr	
1013		

```
int main()
{
    int arr[5] = {1,2,3,4,5};
    printf("%p %p %p \n", arr, &arr, &arr[0]);

    int *ptr;
    ptr = &arr[0];
    printf("%p %p", ptr, ++ptr);
    return 0;
}
```

2D Array

★ Declaration of Array

```
data_type array_name[row_size] [column_size];  
  
int marks[3][5];
```

Rows Columns \	Col 0	Col 1	Col 2	Col 3	Col 4
Row 0	marks[0][0]	marks[0][1]	marks[0][2]	marks[0][3]	marks[0][4]
Row 1	marks[1][0]	marks[1][1]	marks[1][2]	marks[1][3]	marks[1][4]
Row 2	marks[2][0]	marks[2][1]	marks[2][2]	marks[2][3]	marks[2][4]

2D Array

- ★ Accessing the Elements of Array
- ★ Storing Values in Array
- ★ Operations on Array
- ★ Pointer & Array

<code>&arr[0][0] + 1</code>	<code>point to arr[0][1]</code>
<code>arr[0] + 1</code>	<code>point to arr[0][1]</code>
<code>arr + 1</code>	<code>point to arr[1][0]</code>
<code>&arr[0] + 1</code>	<code>point to arr[1][0]</code>

String

★Reading & Writing String

★Operations on Strings

- ★Finding length of a string
- ★Converting characters of a string into upper/lower case
- ★Appending a string into another string
- ★Comparing two strings
- ★Reversing a string
- ★Inserting/Deleteing a string in the main string
- ★Pattern matching

★Arrays of Strings

Structure

★ Declaration

★ Accessing the member of a structure

★ Copying structures

★ Nested structures

★ Arrays of structures

★ Self-referential structures

```
struct node
{
    int val;
    struct node *next;
};
```

```
struct struct-name
{
    data_type var_name;
    data_type var_name;
    .....
};
```

```
struct student stud1
= {01, "Rahul", "BCA", 45000};
```

01	Rahul	BCA	45000
r_no	name	course	fees

```
struct student stud2 = stud1;
```

01	Rahul	BCA	45000
r_no	name	course	fees

Union

- ★ In case of unions, you can only store information in one field at any one time.
- ★ Unions are used to save memory. They are useful for applications that involve multiple members, where values need not to be assigned to all the members at any one time.

Address	Variable
1000	
1001	abc.a
1002	abc.b

```
struct abc
{
    int a;
    char b;
};

a's address = 1000
b's address = 1002
```

Address	Variable
1000	
1001	
1002	

```
union abc
{
    int a;
    char b;
};

a's address = 1000
b's address = 1000
```

Union

```
typedef struct POINT1
{
    int x, y;
};

typedef union POINT2
{
    int x;
    int y;
};

int main()
{
    POINT1 P1 = {2,3};
    // POINT2 P2 ={4,5}; Illegal in case of unions
    POINT2 P2;
    P2.x = 4;
    P2.y = 5;
    printf("\n The coordinates of P1 are %d and %d", P1.x, P1.y);
    printf("\n The coordinates of P2 are %d and %d", P2.x, P2.y);
    return 0;
}
```

Output

The coordinates of P1 are 2 and 3
The coordinates of P1 are 5 and 5

Union

```
typedef struct
{
    char *name;
    bool is_robot;
    char *personality;
    int firmware_version;
}game_character;
```

```
typedef struct
{
    char *name;
    bool is_robot;
    union{
        char *personality;
        int firmware_version;
    };
}game_character;
```



Questions and Answers

