Unit 5 - Arrays

5.1 - one-dimensional Arrays: Declaration, Initialization, Traversal

↑ collection of single ^ data type

Declaration of 1-Dim. Array

Syntex:

```
data_type array_name [size];
```

Example:

```
int marks[5];
```

↑ type of element — ↑ array name — number of element

Initialization

a- At the time of declaration

```
int marks[5] = {90, 80, 70, 60, 95};
```

b- partial initialization

```
int num[5] = {1, 2};
```

∴ remaining element become zero

c- without specifing size

```
int arr[ ] = {10, 20, 30, 40};
```

∴ size becomes 4 automatically

Traversal - means accessing each element of array using a loop.

```
#include <stdio.h>
int main() {
    int arr[ ] = {10, 20, 30, 40, 50};
    int i;
    for (i = 0; i < 5; i++) {
        printf("%d", arr[i]);
    }
    return 0;
}
```

5.2

Two dimensional arrays

1- Declaration

Syntax:

```
data_type array_name [rows][columns];
```

Example:

```
int a[3][4]
```

initialization

A- complie-time intialization

```
int a[2][3] = {
                {1,2,3},
                {4,5,6}
              };
```

B- initialization without inner braces.

```
int a[2][3] = {1,2,3,4,5,6};
```

C- partial initialization.

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

∴ Remaining elements become 0.

Traversal.

using nested loops

```
for (i=0; i<rows; i++) {
    for (j=0; j<columns; j++) {
        printf ("%d", a[i][j]);

    }
    Printf ("\n");

}
```

2- matrix operation

a- matrix addition & sub

program

```
for (i=0; i<m, i++){
    for (j=0; j<n; j++){
     c[i][j] = A[i][j] ± B[i][j];
    }
}
```

b- matrix multiplication

program:

```
for (i=0; i<m; i++){
    for (j=0; j<P; j++){
      c[i][j] =0;
        for (k=0; k<n; k++){
          c[i][j] += A[i][k] * B[k][j];
        }
    }
}
```

c- Transpose of matrix

$$AT[j][i] = A[i][j]$$

Program:

```
for (i=0; i<m; i++){
    for (j=0; j<n; j++){
      AT[j][i] = A[i][j];
    }
}
```

5.3.

^ multi - dimensional array.

int a[2][3][4];

This means:

2 block
each block contain 3 rows
each row has 4 columns

Total 2×3×4 = 24

Traversal of multi

```
for (i=0; i<x; i++) {
    for (j=0; j<y; j++) {
        for (k=0; k<z; k++) {
            printf("%d", a[i][j][k]);
        }
    }
}
```

5.4

^ Array operation

. searching

1- linear

problem - search for 25 in the array

[10, 20, 25, 30, 40]

Program

```
int main() {
    int arr[5] = {10, 20, 25, 30, 40};
    int key = 25;
    int i, found = -1;
    for (i=0; i<5; i++) {
        if (arr[i] == key) {
            found = i;
            break;
        }
    }
```

```
if (found != -1)
    printf("%d found at index %d", key, found);
    else
        printf("Element not found");
    return 0;
    }
```

Explaination

i	arr[i]	key	match
0	10	25	no
1	20	25	NO
2	25	25	yes — stop

found at index 2

2- Binary search

Search for 30 in

[10, 20, 25, 30, 40, 50]

Expected output:

30 found at indyo 3

Example

low	high	mid	arr[mid]	compare with key=30	Action
0	5	2	25	30 > 25	Search right → low=3
3	5	4	40	30 < 40	Search left → high=3
3	3	3	30	30 == 30	found

5.5

Array-based problems: Sum, average, max, min

1- Sum of array elements

Problem:

find the sum of the array.

[10, 20, 30, 40, 50]

Sum = Sum + Array[i]

Dry Run

i	arr[i]	Sum
0	10	10
1	20	30
2	30	60
3	40	100
4	50	150

sum = 150.

2- Average

Average = Sum / no. of elements

Average = 150 / 5

= 30.00

3- Maximum Element in Array

find the largest value in

[12, 45, 7, 34, 99, 23]

[12, 45, 7, 34, 99, 23]

Dry run

i	arr[i]	max
0	12	12
1	45	45
2	7	45
3	34	45
4	99	99
5	23	99

max = arr[0];

max = 99. //

4- minimum

problem: find the Smallest value

[12, 45, 7, 34, 99, 23]

Dry run.

i	arr[i]	min
0	12	12
1	45	12
2	7	7
3	34	7
4	99	7
5	23	7

min = 7.

Passing Arrays to function

In C, arrays are always passed to function by refrence
means C only sends address of
first element

Unit 6- strings and structures

6.1- Character arrays and strings: Declaration, initialization

What is character array?

Example

```
char a[5] = {'H', 'E', 'L', 'L', 'O'};
```

So it looks like;

H | E | L | L | O

2 - What is strings?

- String is a character array that end with a null character ('\0')

Ex.

```
char s[] = "HELLO";
```

Store as: H | e | l | l | l | o | \0

Easy Declaration;

① Fixed size:

```
char name[10];
```
- 10 boxes, can store max 9 character + \0.

② Initialize

```
char Name[] = "Sandip";
```

c automatically adds \0.

s | a | n | d | i | p | \0

Basic String operations: length, copy, concatenate, compare, reverse

- ✓ length (strlen)
- ✓ copy (strcpy)
- ✓ concatenate (strcat)
- ✓ compare (strcmp)
- ✓ Reverse (strrev - custom method.

1- String Length (strlen)

↑ how many character are in the string (except \0)

Ex.

```
char name[] = "Bhutan";
printf("%d", strlen(name));
```

output : 6

2- string copy (strcpy)

```
char S1[20], S2[20];
strcpy(S1, "Hello");
strcpy(S2, S1);
printf("%s", S2);
```

output : Hello

3- concatenate (strcat)

- Join two strings.

Ex -

```
char S1[20] = "Hello";
char S2[20] = "world";

strcat(S1, S2);
printf("%s", S1);
```

output: Hello world

4- compare (strcmp)

- compares two strings.

Return:

0 — strings are equal

positive — S1 > S2

Negative — S2 < S2

Ex.

```
char S1[] = "Apple";
char S2[] = "Apple";
if (strcmp(S1,S2) == 0)
    printf("Same");
else
    printf("Different");
```

Output:

Same

Another example:

strcmp("Bhutan", "India");

Result — Negative (because 'B' < 'I')

6.3

Array of strings and string Handling in function

Example Declaration:

1- fixed-size array of strings (2D Array)

```
char country [3][10] = {
                        "Bhutan",
                        "India",
                        "Nepal"
                       };
```

This means:

3 strings
each string can store up to 9 chars + \0

Diagram

country [0]: Bhutan \0

country [1]: India \0

country [2]: Nepal \0

printing Array in strings.

```
#include <stdio.h>
int main() {
        char countries [3][10] = {"Bhutan", "India",
                                  "Nepal"}

for (int i=0; i<3; i++) {
    printf("%s\n", countries[i]);

    }

return 0;
}
```

Output:

- Bhutan
- India
- Nepal

String Handling in function

Just like arrays, strings are also passed by the refrence. This means the following get the orginal memory address, not a copy.

Note

6.4 Structures: Declaration, initialization, Accessing Members.

A structures is user-defined data type that allows you to store different types of data under one name

Think of it as a box that contains different items

Ex- A student has.
- name (string)
- age (int)
- marks (float)

we group them together using struct.

1- Declaration

Syntax:

```
struct structure_name {
    data_type member1;
    data_type member2;
    data_type member3;
};
```

Ex

```
struct student {
    char name[20];
    int age;
    float marks;
};
```

creating structure variables

method 1 (outside main)

- `struct Student S1, S2;`

method 2 (inside main)

-
```
int main(){
    struct Student S1;
}
```

Initialization

method 1: - Direct initialization

```
struct Student S1 = {"Sandip", 20, 88.5};
```

method 2: Assigning each member

```
struct Student S2;
strcpy(S2.name, "karma");   // because name is string
S2.age = 21;
S2.marks = 90.0;
```

Accessing structure members

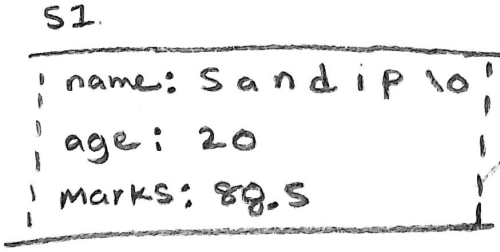
use dot operator (.)

```
Printf("%s", S1.name);
Printf("%d", S1.age);
Printf("%f", S1.marks);
```

code sample

```
#include <stdio.h>
#include <string.h>
Struct Student {
        Char name[20];
        int age;
        float marks;
    };
int main(){
        Struct Student S1;
    // Assigning value
strcpy(S1.name, "Sandip");
S1.age = 20;
S1.marks = 88.5;
    // printing value
Printf("Name: %s\n", S1.name);
Printf("Age: %d\n", S1.age);
Printf("Marks: %.2f\n", S1.marks);
return 0;
}
```

memory diagram



6.5 Array of Structures.

An array of structures means many structure variable stored together (like list of student).

Ex. structure

```
struct student {
        char name[20];
        int age;
        float marks;
};
```

Now if we want to store 5 student's,

```
struct student s[5];
```

→ This means.

s[0] - student 1
s[1] - student 2
s[2] - student 3
s[3] - student 4
s[4] - student 5.

each has
- name.
- age
- marks.

Ex -

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

struct student {
        char name[20];
        int age;
        float marks;
};
int main() {
    struct student s[3];   // Array of structre
    for(int i=0; i<3; i++){
        printf("Enter detail of student %d:\n", i+1);
        printf("Name: ");
        scanf("%s", s[i].name);
        printf("Age: ");
        scanf("%d", &s[i].age);
        printf("Mark: ");
        scanf("%f", &s[i].marks);
    }
    printf("\n----- Student Details ---\n");
    for(int i=0; i<3; i++) {
        printf("Name: %s | Age: %d | Mark: %.2f\n",
                s[i].name, s[i].age, s[i].marks);
    }
    return 0;
}
```

Example : initialization array of struct

```
struct student s[3] = {
    {"Sonam", 19, 87.5},
    {"Karma", 20, 92},
    {"Tashi", 21, 85}
};
```

Memory alocation

s[0] - name, age, marks
s[1] - name, age, marks
s[2] - name, age, marks

Nested structure.

A nested structure means: A structure inside another structure.

Ex.
A student has:

name
age
marks
Address - (house no, city, state)

we can create Address structure inside Student.

Example:

```
struct Address {
        char city[20];
        int pin;
    };
struct Student {
        char name[20];
        int age;
        struct Address addr;  // structure inside structure
    };
```

Nested Structure Example

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

Struct Address {
        char city[20];
        int pin;
      }
Struct student {
        char name[20];
        int age;
        Struct Address addr;
    }
int main() {
      Struct student s;
       strcpy(s.name, "Sandip");
       s.age = 20;
       strcpy(s.addr.city, "Thimphu");
       s.addr.pin = 11001;

       printf("Name: %s\n", s.Name);
       printf("Age: %d\n", s.age);
       printf("city: %s\n", s.addr.city);
       printf("pin : %d \n", s.addr.pin);

       return 0;

      }
```

* How to ACCESS Nested Structure members?
  - use two dot.

    Ex- s.addr.city.
        s.addr.pin.

6.6. Unions and Differences from Structures

A union is like a structure, but with one major difference
- All members share the same memory location.
- only one member can hold a value at a time.

This save memory.

Declaration

Syntax.

```
union Data {
    int i;
    float f;
    char c[10];
};
```

This create a union named Data.

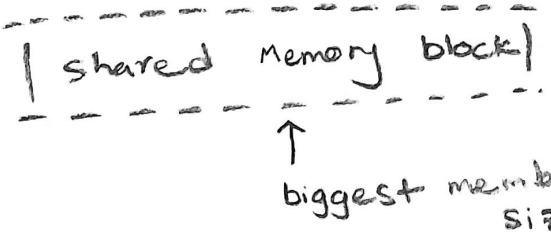
Creating union variable

```
union Data d1;
```

Initialization

```
d1.i = 10;
d1.f = 2.5;
strcpy (d1.c, "Hello");
```

* Remember:
setting one member overwrites previous data because they share the same memory.



if union has
- int - 4 bytes
- float - 4 bytes
- char[10] - 10 bytes.

Total = 10 bytes (size of largest member).

Structure Vs Union Memory

structure:

```
struct A {
    int i;          // 4 bytes
    float f;        // 4 bytes
    char c[10];     // 10 bytes
};
```

Total memory = 4+4+10 = 18 bytes

union:

```
union A {
    int i;
    float f;
    char c[10];
};
```

Total memory = 10 bytes [size of largest number]

Feature	Structure	union
memory	sum of all member	size of largest number
store	All member at once	only one member at a time
memory Allocation	Seperate memory for each member	share memory for all member
usage	complex record	memory-saving Suation

Accessing union member

Same as Structure — using dot operator (.)

```
printf("%d", d1.i);
printf("%f", d1.f);
printf("%s", d1.c);
```

Access	All number Valid	only last assigned member valid.

structure	union
- store multiple value at same time	- store only one value at a time.
- total of all member	- max of all member
- used when you need all member active	- when you need only one type of value at a time
- student record, bank acc. detail.	- temp- humidity sensor.

6.7. User-Defined Types: typedef, enum.

1- typedef - used to give a new name (alias) to an existing data type

why we used?
- makes code clean
- easy to reuse complex types
- improves readability.

Ex-

```
#include <stdio.h>
typedef int Number;
int main() {
    Number x = 70 ; // same as : int x = 10.
    printf("%d", x);
    return 0;
}
```

Ex -

```
struct Student {
    int roll;
    char name[20];
};
struct Student S1;
```

with typedef.

```
typedef struct {
    int roll;
    char name[20];
} student;
student S1; //
```

No need to write struct

2- enum (Enumeration) - is used to create a set of named integer constants.

why used?
- make code easy to understand
- instead of using random numbers.

Ex 1

```
include <stdio.h>
enum Day { sun, Mon, Tue, Wed,
           Thu, Fri, Sat };
int main() {
    enum Day today = wed;
    printf("%d", today); // print 3( index start from 0)
    return 0;
}
```

	typedef	enum
use for	- structure, array, basic types	- limited / related values.