Unit-8

Structure and Union

Introduction to Structure:

- Structure is a user defined data type.
- A structure is a collection of one or more variables, possibly of different data types (e.g. int, float, char) grouped together under a single name for convenient handling.
- Structure help to organize data, particularly in large programs, because they allow a group of related variables to be treated as a single unit.
- Once the structure has been declared, we can create a variable of its type.
- In some language, structures are know as **records** and the elements of structure are known as **fields** or **members** or **components**.

Declaration of Structure:

Syntax: (For defining structure)

```
struct tagname
{
    data_type element1;
    data_type element2;
    -----------------------
    -----------------------
};
```

Where **struct** is a keyword and we must use for defining structure and **tagname** is a structure name and we can give any name to the structure.

Note: Every structure must end with a semicolon.

Example:

```
struct emp
ombined a single data type of user defined type.
{
    int eid;
    char ename[20];
    float esalary;
}e;
```

Note: we can identify the structure with the help of tag_name or identity name.

Declaration of Structure Variable:

Method-1:

```
struct book
{
    int pages;
    char author[20];
    float price;
}b;
```

Where **b** is the structure variable for the book structure.

Continue…

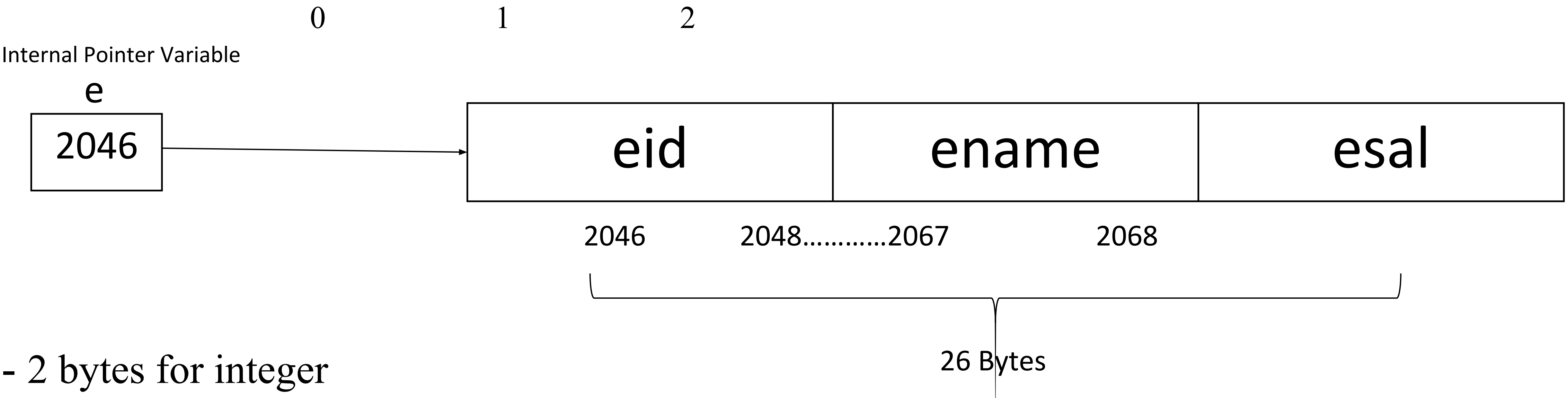
Method-2:

```
struct book
{
    int pages;
    char author[20];
    float price;
};
struct book b;
```

Memory Allocation of Structure:

-Just defining the structure, it does not get the memory allocation.

-Whenever we declare variable then only it gets memory allocation.



- 2 bytes for integer
- 20 bytes for ename because each character occupies 1 bytes
- 4 bytes for float.

Initialization and accessing of structure:

- Once we allocate the memory, then we can access(retrieve and store)the elements of that structure.
- Structure variable is access with the help of accessor and accessor is a dot operator (.)
- Example: e.eid, e.ename, e.esalary

Initialization:

Method-1:

```
struct book
{
    int pages;
    char author[20];
    float price;
}b={100, “Ram”, 545.5};
```

Continue…

Method-2:

```
struct book
{
    int pages;
    char author[20];
    float price;
};
struct book b={100, “Ram”, 545.5};
```

Using dot operator:

```
struct book
{
    int pages;
    char author[30];
    float price;
};
struct book b;
b.pages=100;
strcpy(b.author, “Ram”);
b.price=545.5;
```

Example of access of structure element:

```
#include<stdio.h>
#include<conio.h>
struct emp
{
        int eid;
        char ename[20];
        float esalary;
};
void main()
{
        struct emp e={101, "Kiran", 54000.5};
        printf("Your Details:\n");
        printf("EID=%d\n",e.eid);
        printf("ENAME=%s\n",e.ename);
        printf("ESALARY=%.2f",e.esalary);
        getch();
}
```

Program to find the size of the structure:

```
#include<stdio.h>
#include<conio.h>
struct emp
{
        int eid;
        char ename[20];
        float esalary;
};
void main()
{
        struct emp e;
        printf("Size of emp:%d Bytes\n",sizeof(e));
        printf("Size of emp:%d Bytes",sizeof(struct emp));
        getch();
}
```

Local Structure	Global Structure
- Declaration of structure inside a particular function is called local structure.	- Declaration of structure outside of all the function is called global structure.
- It is accessible only inside this function.	- It is accessible from every where of the program.

Example:
main ( )
{
struct local
{
int a,b;
};
struct local l; ---it is accessible
}
check ( )
{
struct local l; ---it is not accessible
}
	Example:
	struct global
	{
	int a,b;
	};
	main ( )
	{
	struct global g; ---it is accessible
	}
	check ( )
	{
	struct local g; ---it is accessible
	}

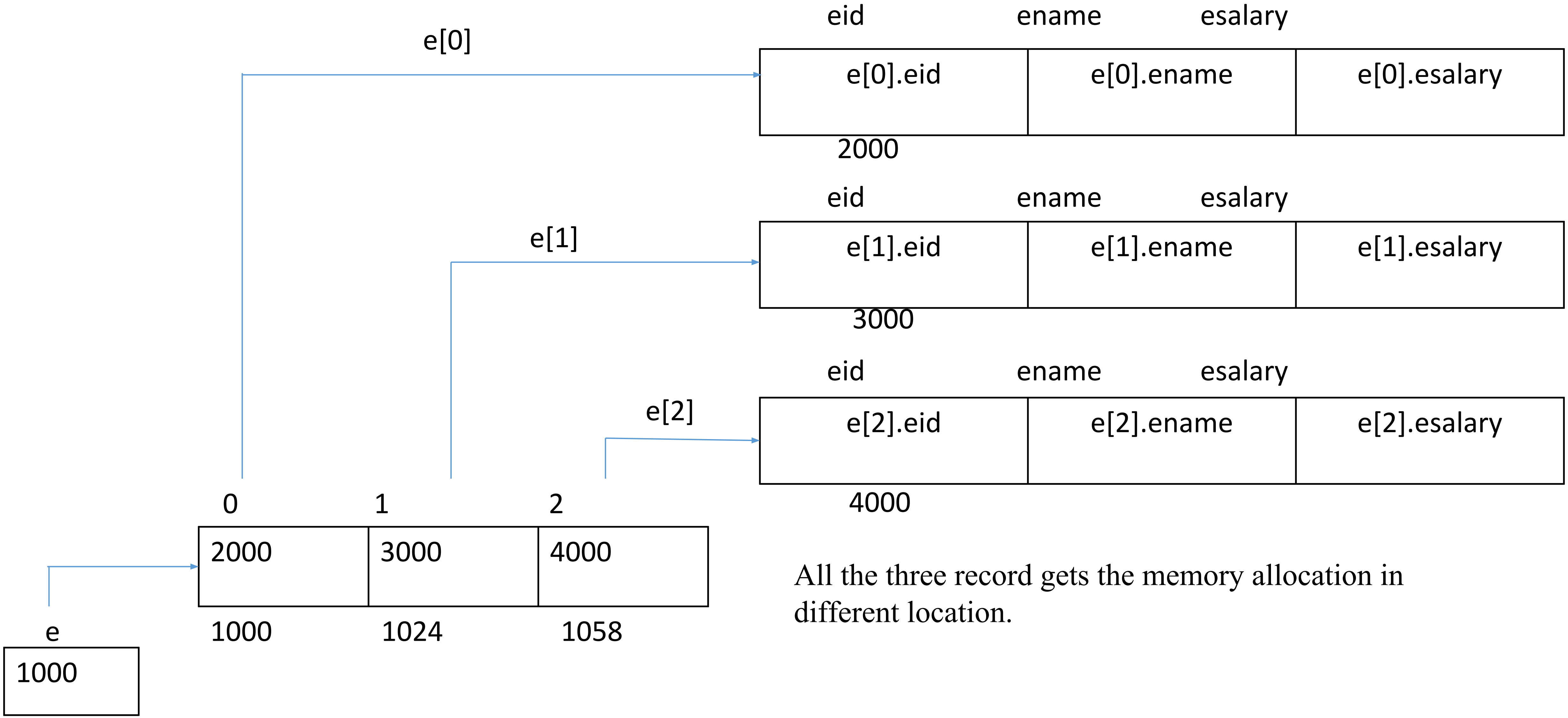
Array of Structure:

- A collection of similar type of structure placed in a common variable name is called array of structure.
- Declaring an array of structure is same as declaring an array of fundamental types. Since an array is a collection of elements of the same type. In an array of structures, each element of an array is of the structure type.

Example:

```
struct emp
{
    int eid;
    char ename[20];
    float esalary;
};
struct emp e[3];
```

How actually memory gets allocate ?



e[0]

eid ename esalary

e[0].eid e[0].ename e[0].esalary

2000

e[1]

eid ename esalary

e[1].eid e[1].ename e[1].esalary

3000

e[2]

eid ename esalary

e[2].eid e[2].ename e[2].esalary

4000

0 1 2

2000 3000 4000

1000 1024 1058

e

1000

All the three record gets the memory allocation in different location.

Union:

- Union is a user defined data type.
- In Union, we can store any type of data but we can’t store all the elements at a time. So we can store one by one element.
- We can process all the elements of union one after another when it is required.
- Structure is more easy and more flexible than union.
- In union, we can define n number of elements at a time but we can’t access all the elements at a time (i.e. we can process only one element at a time).
- To access the element in union, we also use the dot operator.

Declaration of Union:

Syntax:

```
union tagname
{
    data-type element-1;
    data-type element-2;
    ……………………………….
    data-type element-n;
};
```

Example:

```
union std
{
    int i;
    float h;
    char c;
};
union std u;
```

Here, all the three variable sharing the same memory location.

Example:

```
#include<stdio.h>
#include<conio.h>
union std
{
        int a;
        int b;
};
void main()
{
        union std u;
        u.a=40;
        printf("b=%d\n",u.b);
        u.b=50;
        printf("a=%d",u.a);
        getch();
}
```