User Defined Datatypes (UDT)

- User defined data types are collection of standard data types
- UDTs are used to create our own data type with the help of primitive data types.
- In UDT mainly we have,
    - Structures
    - Unions
    - Enum
    - Typedef

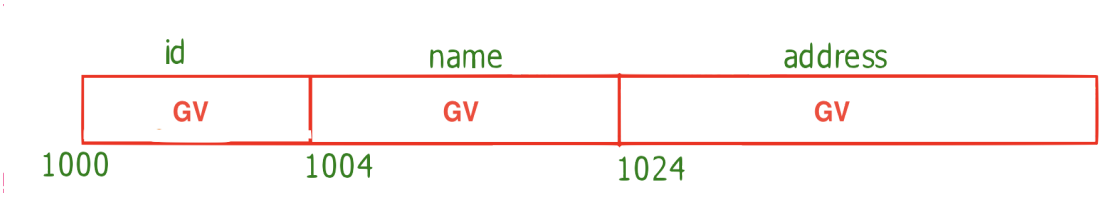
1. Structures

- Structure is a collection of the same or different data type under one common name.
- Syntax:
    - struct structure_name
      {
      ///structure members
      };
    - To create structure variable
        - struct structure_name variable_name;

- Example:
  struct Student
  {
  int id;
  char name[20];
  char address[30];
  };

    - Memory will not be allocated when u define/write structure
    - Memory is allocated when you create structure variable

  struct Student st1;



- Memory allocated will be continuous and also depending on the structure padding
- Here, 84 bytes are allocated
- To access structure members/fields will use the .(dot) operator.
  E.g., st1.id = 100;

Structure copy

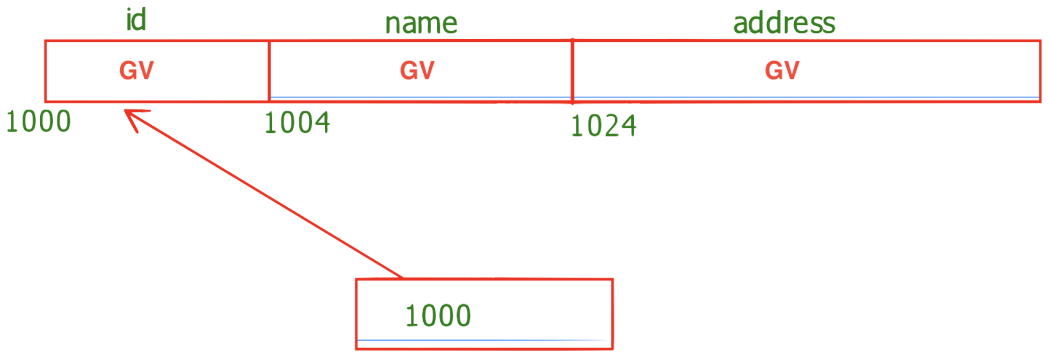
- In the case of arrays we learned that copying one array to another is not straight forward, we need to use a loop.
- But in the case of structure, it can be copied by using assignment operator
- Example:
  - struct Student s1 = {10,"Rama","Bangalore"};

```
struct Student s2;

s2 = s1;
```

Structure Pointers

- We can have structure pointer to store the address of structure variable
- For example:

```
struct Student *sptr

struct Student s1;
struct Student *sptr = &s1;
```

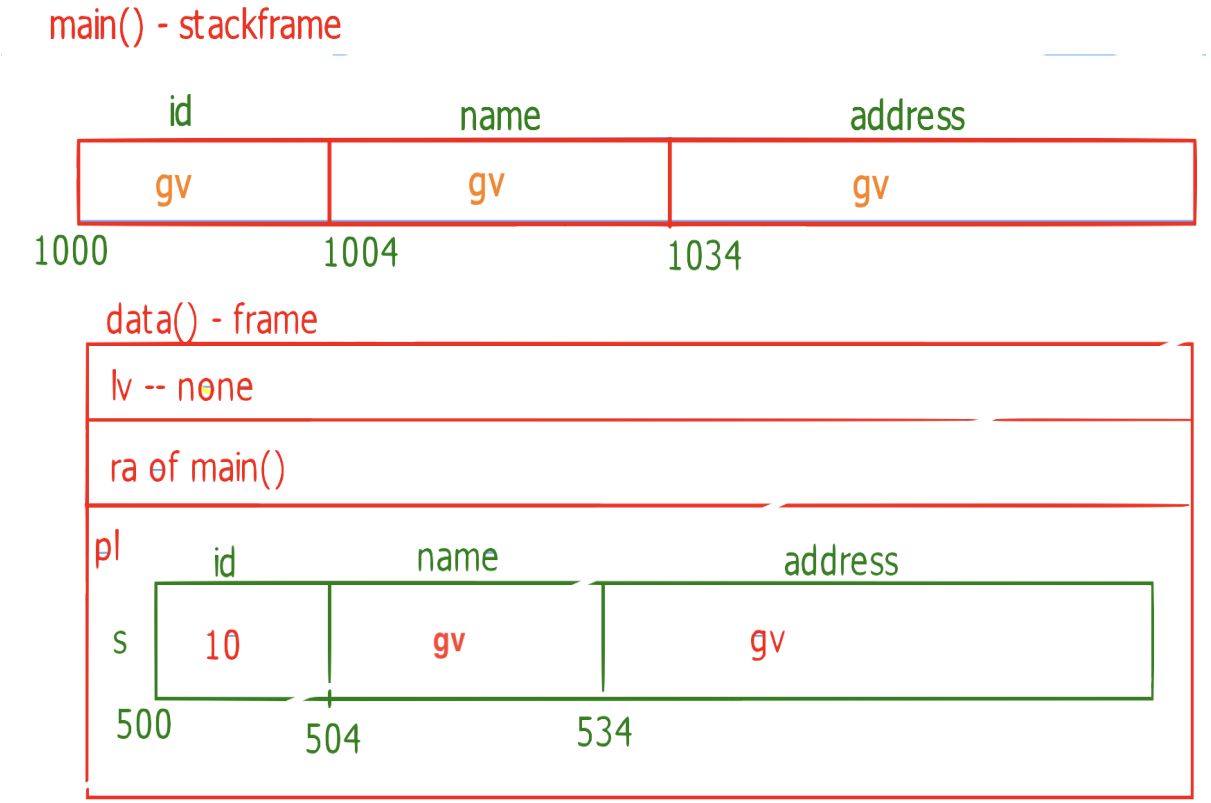


How to access structure members or fields?

(*sptr).id = 10; or sptr -> id = 10
(*sptr).name; or sptr -> name
(*sptr).address or sptr->address

Passing structure to function

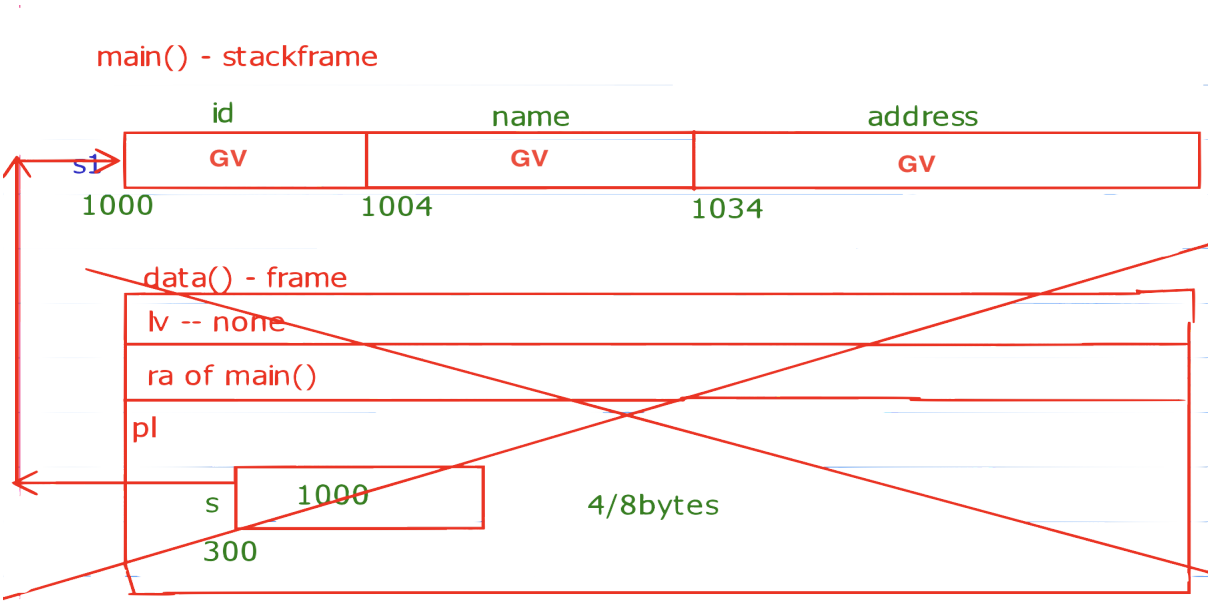
- Pass by value : Not recommended in case of structure because structure is huge user defined data type so it will take more space in stackframe



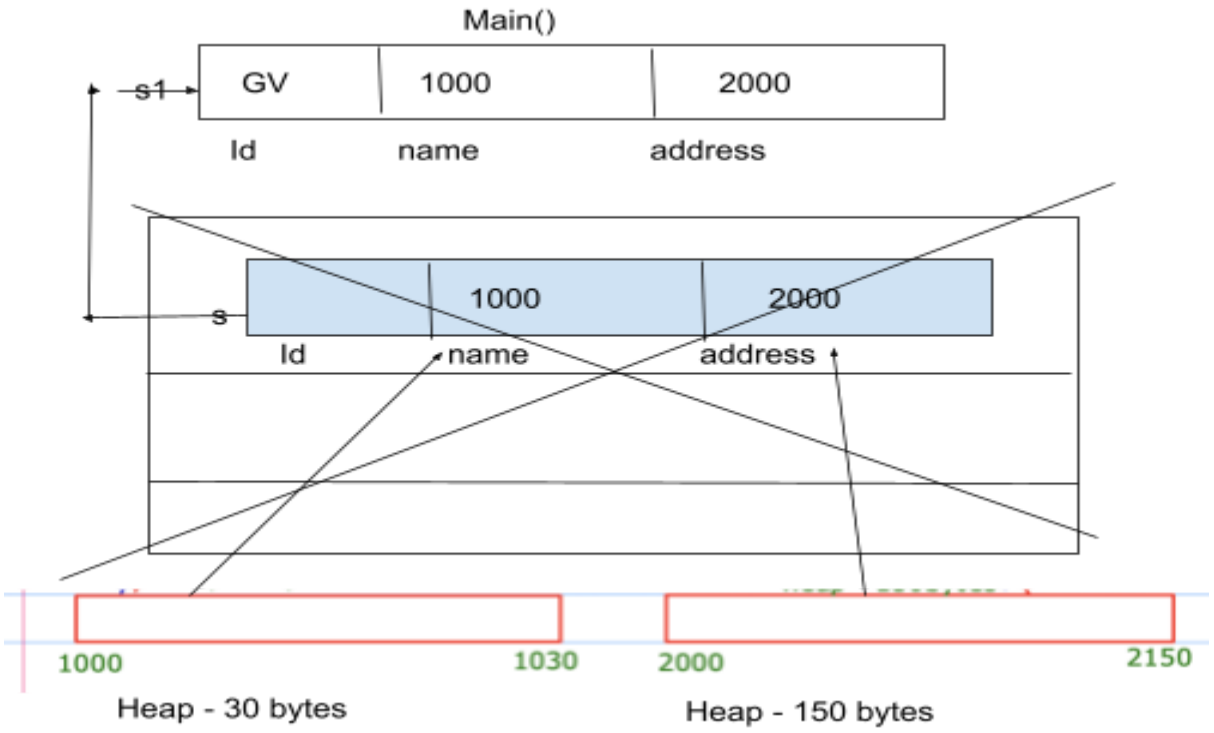
- As you can see in the above example structure takes uipto 184 bytes in stack which is huge in size, so normally pass by value method is avoided in case of structure

Pass by Reference: preferred method because takes less memory

- As you can see in pass by reference it only need 4/8 bytes of memory.



Returning structure from function

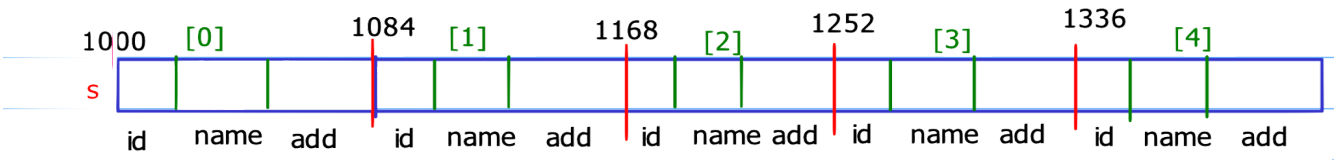


- Like a normal variable a structure can be returned from function if its created in the heap like below:

Array of structure

- Array of structure is a collection of more than 2 structure variables
- For example,
    - struct Student
      {
      int id;
      char name[20];
      char address[60];
      };

    struct Student s[5];



Accessing array of structure

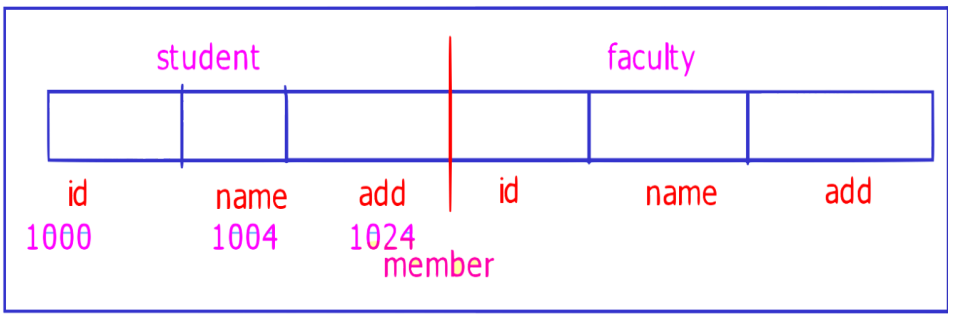
- Like a normal array we can access array of structure through loop:

```
for(i = 0; i < 5; i++)
{
        printf("%d %s %s\n", s[i].id,s[i].name,s[i].address);
}
```

Nested Structures

```
struct College
{
    struct Student
    {
    int id;
    char name[20];
    char address[60];
    } student;

    struct
    {
    int id;
    char name[20];
    char address[60];
    } faculty;
};
```

struct College member;



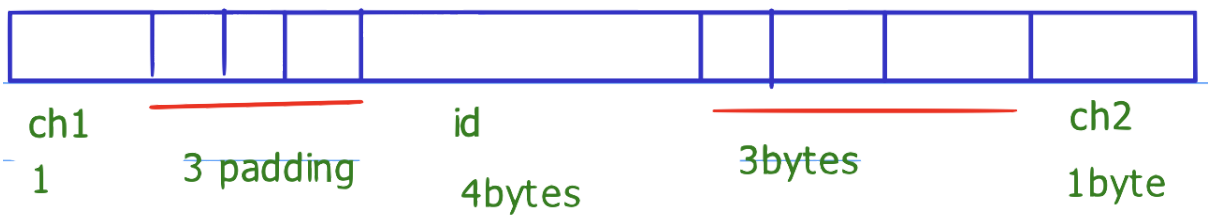
member.student.id = 101
member.student.name
memebr.student.address

member.faculty.id = 11
member.faculty.name
member.faculty.address

Structure Padding

- Structure padding is a process of adding useless bytes in between members of structure to make processor work easy.
- Structure padding is done based on following points:
    - based on the member arrangement
    - based on the largest member size
    - based on word size
- For example:

```
struct Student
{
        char ch1;
        int id;
      char ch2;
}
```



2. Unions

- Like structures, union is also a user defined datatype which can have the same or different types of data.
- Major difference between structure and union is, union shares the memory of the largest member among the other members.
- E.g.,
    - union Test
      {
            char option;
            Int id;
            double height;
      };
- In above example height is biggest member of the union so, the sizeof the union will be 8bytes and that memory is shared among the others

3. Typedefs

- Typedefinition is used to create a new name to the xisting datatype
- E.g,
    - typedef unsigned long int uli;

- Typedef makes complex definition simpler like in the above example whenever you need to create a unsigned long int you dont have to write that full name, simply one can create like:
    - uli n1;
- Its that simple to use and typedef is used with all of the datatypes

4. Enum

- Enum is a another user defined datatype which is used to group named constant under a single name.
- It is called as set of named integral values.
- For example:
    - enum boolean
      {
      false,
      true
      }
- In the enums first named constant always will have 0 as initialised value and next member will be incremented by 1.
- So, false is having value 0 and true is having value 1.
- We can also initialised own integral vlaues to the enum members like,
    - enum boolean
      {
      green = 90,
      Blue = 40
      }