C PROGRAMING

by Akshita Chanchlani @ Sunbeam Infotech

Day 5 : Dynamic Memory Allocation and Structure

Dynamic Memory Allocation

Dynamic Memory Allocation

- Dynamic memory can be requested using malloc,calloc,realloc function
- Such requested memory will be in control of user programmer.
- On demand programmer request memory utilse same and once job is finished programmer has to release such dynamic memory.
- We can shrink or grow dynamic memory at runtime.
- malloc,calloc,realloc function provides memory from heap section.
- If request is successful they will return base address of memory else return NULL.

- void* malloc(int size);
- void* calloc(int count,int elesize);
- void* realloc(void *memblock,int size);
- The address returned by these functions should be type-casted to the required pointer type.
- Calloc() allocates an array in memory with elements initialized to zero.
- Changes size of allocated memory block.

malloc requested memory is always assigned with garbage, where as calloc requested memory is by default initialised with 0.

Memory leakage

- If memory is allocated dynamically, but not released is said to be "memory leakage".
  - Such memory is not used by OS or any other application as well, so it is wasted.
  - In modern OS, leaked memory gets auto released when program is terminated.
  - However for long running programs (like web-servers) this memory is not freed.
  - More memory leakage reduce available memory size in the system, and thus slow down whole system.
- In Linux, valgrind tool can be used to detect memory leakage.

```
int main() {
    int *p = (int*) malloc(20);
    int a = 10;
    // ...
    p = &a; // here addr of  allocated block is lost, so this memory can never be freed.
    // this is memory leakage
    // ...
    return 0;
}
```

Dangling pointer

- Pointer keeping address of memory that is not valid for the application, is said to be "dangling pointer".
- Any read/write operation on this may abort the application. In Linux it is referred as "Segmentation Fault".
- Examples of dangling pointers
  - After releasing dynamically allocated memory, pointer still keeping the old address.
  - Uninitialized (local) pointer
  - Pointer holding address of local variable returned from the function.
- It is advised to assign NULL to the pointer instead of keeping it dangling.

```
int main() {
    int *p = (int*) malloc(20);
    // …
    free(p); // now p become dangling
    // …
    return 0;
}
```

Dangling Pointer

- A pointer pointing to a memory location that has been deleted (or freed) is called dangling pointer.
- It is pointer which keeps pointing to memory which is already deallocated.
- If we try to apply value at operator on such pointer it may result unpredictable value as it can be pointing to some invalid(dead) address or the one memory which is freed can be in alive state due to request by other programmer .

Steps for Dynamic Memory Allocation

1. REQUEST MEMORY DYNAMICALLY
2. USE DYNAMIC MEMORY - PROVIDING RECEIVED VALID ADDRESS AFTER CALL OF MALLOC
3. RELEASE DYNAMIC MEMORY

Structure

Structure

- It helps to collect members/fieldsof similar or disimilar type.
- It can be used to define data type.
- In case of structure every member receives memory separately.(i.e Each element has its own storage)0
- Structure is collection of non-similar data elements in contiguous memory locations.
- Structure is user defined data type.
- Members of structures can be accessed using "." operator with structure variable.
- Members of structures can be accessed using "→" operator with pointer to structure variable.
- Syntax of structure

```
struct <tagname>
{
            <data type> <identifier> ;
             ....
};
```

Structure Declaration

```
/*Structure declarations are generally done before main i.e. global declaration. We can also do it in a
   function.*/
struct student {
   int roll_no;
   char name[10];
   float avg;
};

/*initializing  structure variable at its declaration*/
void main() {
   struct student s1={1,”Akshita”,80};
   printf(“size=%d”,sizeof(s1));
   printf(“roll=%,name=%s,avg=%f”,s1.roll, s1.name, s1.avg);
}
```

Nested structures

- One can define a structure which in turn can contain another structure as one of its members.
- Example:

```
typedef struct
{   int dd;   int mm;    int yy;  }DATE;
typedef struct
{   int rollno;
    int marks;
    struct
    {           char fname[10];
                char mname[10];
                char lname[10];
    }name;
    DATE dob;
}STUDENT;
```

Arrays of structure

- C does not limit a programmer to storing simple data types inside an array. User defined structures too can be elements of an array.
- <keyword_struct> <struct_name> <struct_variablename>[sizeof_elements];
- struct student s[10];

Passing Structure to a Function by Value and by Reference

- A structure can be passed to any function from main function or from any sub function.
- Structure definition will be available within the function only.
- Example

```
  struct student
{ int rollno; int age; };
   void display(student s);  // passing structure by value in function argument
  //display(st); // calling function

   void show(student *s); // passing structure by reference in function argument
//show(&st); // calling function
```

Structures and Pointers

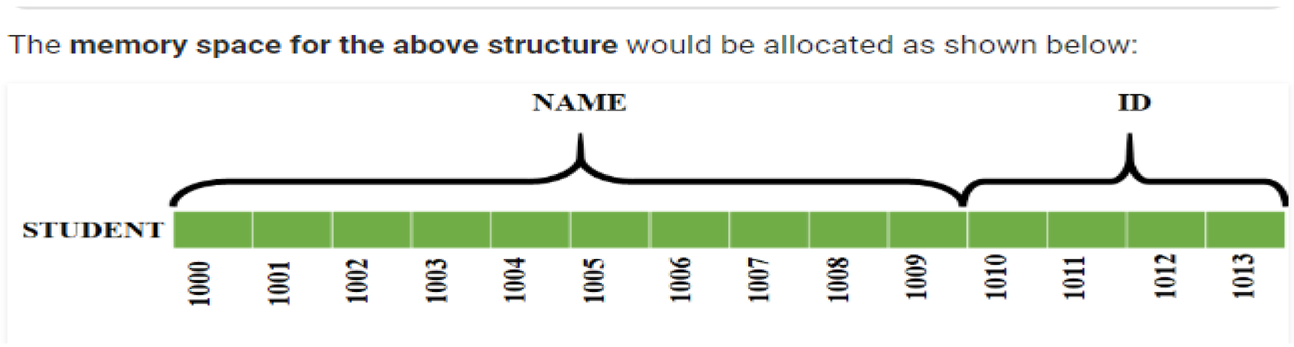
- Just like a variable, you can declare a pointer pointing to a structure and assign the beginning address of a structure to it.

Pointer to Structure

```
void main()
 {
		struct student s1={10,”Akshita”,78.67};
		struct student *ptr = &s1;
		printf(“size=%d”,sizeof(ptr));
		printf(“roll=%d,nm=%s,avg=%f”,ptr→roll,     ptr→name, ptr→avg);
}
```

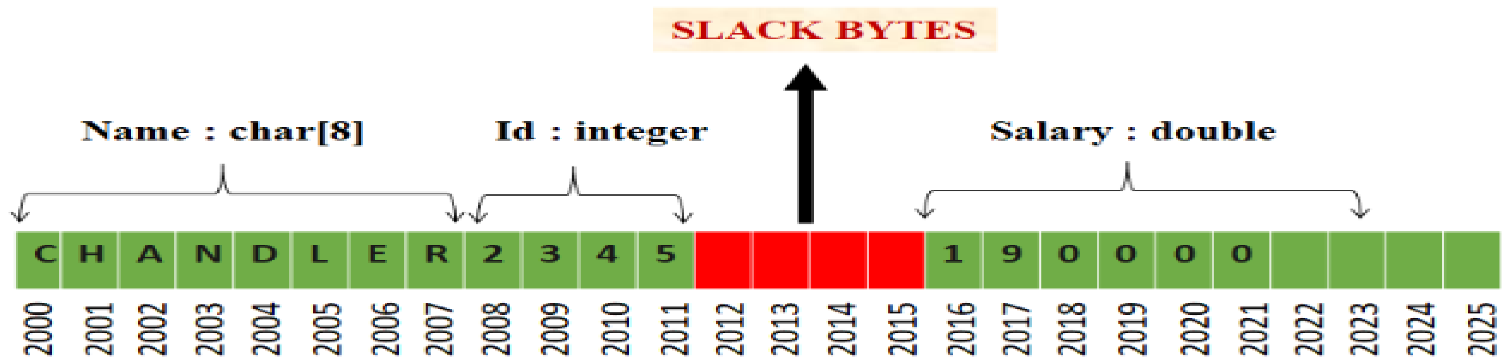
Slack Bytes

- Structures are used to store the data belonging to different data types under the same variable name.
- struct STUDENT { char[10] name; int id; };


The **memory space for the above structure** would be allocated as shown below:

•Here we see that there is no empty spaces in between the members of the structure. But in some cases **empty spaces** occur between the members of the structure in and these are known as ***slack bytes***.

Struct EMPLOYEE
{
char name[8];
int id;
double salary;
}



Bit field

- In programming terminology, *a bit field is a data structure that allows the programmer to allocate memory to structures and unions in bits in order to utilize computer memory in an efficient manner*.
- Limitations of bit-fields
  - Cannot take address of bit-field (&)
  - Cannot create array of bit-fields.
  - Cannot store floating point values.
- Example:

```
typedef struct
{
   char name[20];
   int rn:5;
   int marks:4;
}STUDENT;
```

Thank you!

Akshita Chanchlani <akshita.chanchlani@sunbeaminfo.com>