

Lecture 20

sizeof, size of structure, bit-field, enums

Assigning string value

```
struct student_database {  
    char name[10];  
    int roll;  
    int marks;  
}s;
```

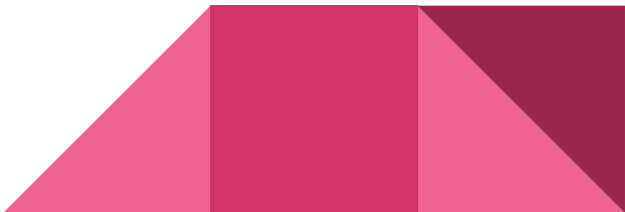
```
void main()  
{  
    s.name="Hello";  
    s.marks=500;  
    printf("Name : %s", s.name);  
    printf("Marks of Student : %d",  
        s.marks);  
    getch();  
}
```



Assigning string value

```
struct student_database {  
    char name[10];  
    int roll;  
    int marks;  
}s;
```

```
void main()  
{  
    strcpy(s.name,"Hello");  
    s.marks=500;  
    printf("Name : %s", s.name);  
    printf("Marks of Student : %d", s.marks);  
    getch();  
}
```



Why do we need to use strcpy in struct member

When you declare a string as a member of a structure, you're essentially declaring an array of characters within that structure. However, in C, you can't directly assign one array of characters to another using the assignment operator '='. So if you declare, `int a[50], b[50];` then you can't write `a = b;`

The reason for this is that arrays are actually pointers to their first element in many contexts, including when they're assigned to another array. This means that if you simply use the assignment operator to assign one string to another, you're actually just copying the pointer to the first character of the source string to the destination string, not the contents of the string itself.

The `strcpy` function, on the other hand, is specifically designed to copy the contents of one string to another. It takes care of copying each character from the source string to the destination string until it encounters the null terminator, ensuring that the entire string is copied properly.

So, to assign a string to a string member of a structure in C, you need to use `strcpy` or similar functions to ensure that the contents of the string are properly copied.

Sizeof

// C Program to illustrate that the 'sizeof' operator is a 'compile time operator'

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

```
int main(void)
```

```
{
```

```
    int y;
```

```
    int x = 11;
```

```
    // value of x doesn't change
```

```
    y = sizeof(x++);
```

```
    // prints 4 and 11
```

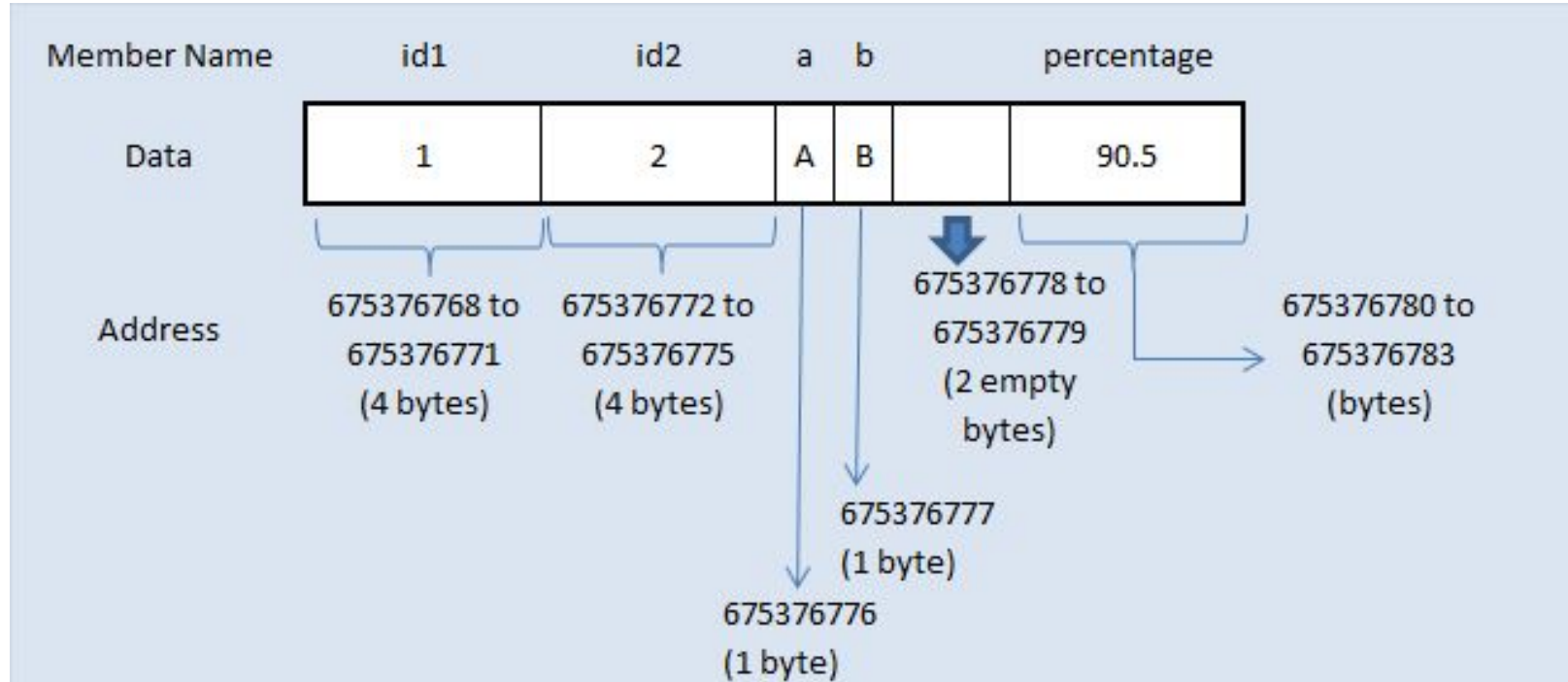
```
    printf("%i %i", y, x);
```

```
    return (0);
```

```
}
```



Alignment in Structures



Bit Field/ Bit Manipulation in C

- Bit field can be used to **reduce memory consumption** when it is known that only some bits would be used for a variable. Bit fields allow efficient packaging of data in the memory.
- As we know, integer takes two bytes(16-bits) in memory. Sometimes we need to store value that takes less than 2-bytes. In such cases, there is wastages of memory.
- `struct { type [member_name] : width ; };`

Bit Field in C

```
struct struct-name
{
    datatype var1 : size of bits;
    datatype var2 : size of bits;
    .....
    .....
    datatype varN : size of bits;
};
```


Bit Field in C Example

```
struct info1 {  
    int num;  
} f1;
```

```
struct info2 {  
    int num : 1;  
} f2;
```

```
void main() {  
    printf("\n\n\tSize of info1 is : %d",sizeof(f1));  
    printf("\n\n\tSize of info2 is : %d",sizeof(f2));  
}
```

Restrictions on Bit-fields

The following restrictions apply to bit fields. You cannot,

- Define an array of bit fields.
- Take the address of a bit field
- Have a pointer to a bit field.



Use of a bit-fields

```
union charToBinary{
```

```
    unsigned char n;
```

```
    struct {
```

```
        unsigned a:1;
```

```
        unsigned b:1;
```

```
        unsigned c:1;
```

```
        unsigned d:1;
```

```
        unsigned e:1;
```

```
        unsigned f:1;
```

```
        unsigned g:1;
```

```
        unsigned h:1;
```

```
};
```

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

```
int main()
```

```
{
```

```
    x.n = 'a';
```

```
    printf("%d%d%d%d%d%d%d%d", x.h, x.g,x.f,x.e,x.d,x.c,x.b,x.a);
```

```
    return 0;
```

```
}
```

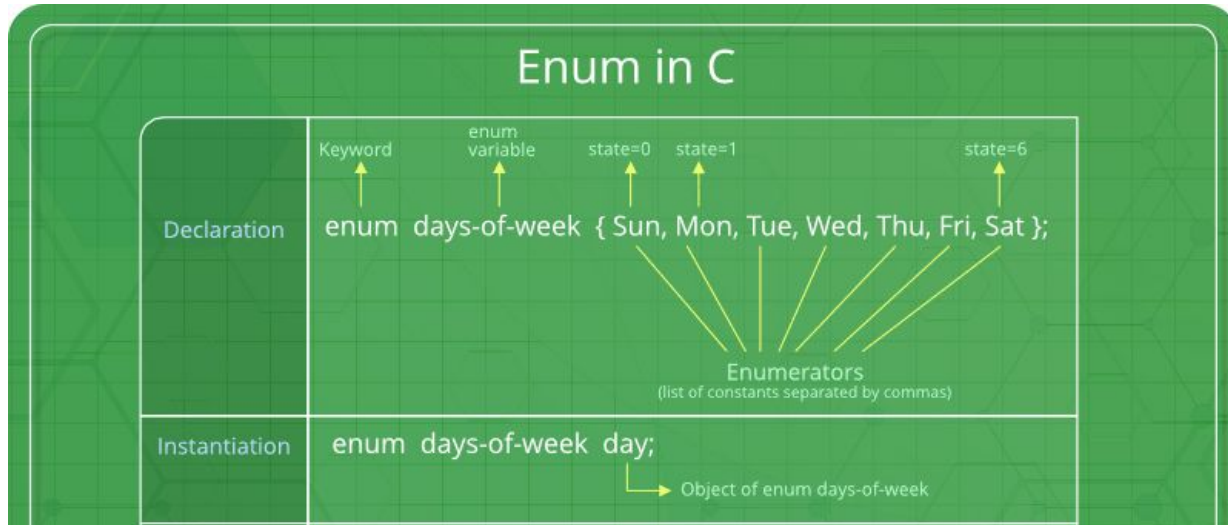
Enumeration in C

Enumeration (or enum) is a user defined data type in C. It is mainly used to assign names to integral constants, the names make a program easy to read and maintain.

The keyword 'enum' is used to declare new enumeration types in C and C++.



Enum in C



Enum in C

```
enum Bool {FALSE, TRUE};

enum Month {JAN=1,FEB,MAR,APR,MAY,JUN,JUL,AUG,SEP,OCT,NOV,DEC};

enum Day {THT = 30, THO = 31, TWN=28};

int days(int m){

    if(m == FEB) return TWN;

    else if(m == APR || m == JUN || m == SEP || m == NOV) return THT;

    else return THO;

}

int odd(int n){

    if (n%2 == 1)    return TRUE;

    else    return FALSE;

}
```

```
int main( ){

    enum Bool b;

    int m;


    b = odd(6);

    printf("%d ",b);

    scanf("%d",&m);

    printf("\n Number of Days in the Month: %d",days(m));

}
```



Resources

- <https://www.geeksforgeeks.org/structure-member-alignment-padding-and-data-packing/>
- <https://www.geeksforgeeks.org/sizeof-operator-c/>
- <https://www.geeksforgeeks.org/enumeration-enum-c/>

