Two Dimensional Arrays in C

Two dimensional Arrays in C

- Static declaration:

  int a[5][7];

```
for(int i=0;i<r;i++){
    for (int j=0;j<c;j++)
        a[i][j]=5;
```

Two Dimensional Array as a Function Parameter

- A function that returns the summation of all the elements in a two dimensional array

```
int sum( int arr[][5], int r)
{
int s=0;
for(int i=0;i<r;i++){
   for (int j=0;j<5;j++)
    s+=arr[i][j];
}
return s;
}
```

- Number of columns needs to be mentioned in the function header

Two Dimensional Array as a Function Parameter

```
void main(){

int arr[4][5];
/*filling the array*/
 for (int i=0; i<4; i++)
    for (int j=0; j<5; j++)
        t[i][j]=25;

int result = sum(arr,4);

printf("%d \n", result);

}
```

Allocating a Two Dimensional array Dynamically

- How dynamically allocate memory for a two dimensional array with r rows and c columns?
- We should define an array of pointers (length: number of rows (r)) where each pointer is pointing to the array of elements in one row which includes c elements

Allocating a Two Dimensional array Dynamically

```
int **b;   //a pointers which points to a set of pointers
int r=2;  //number of rows
int c=3;  //number of colomns
b= (int**) malloc(r*sizeof(int*));
 for(i=0; i<r; i++)
   b[i] = (int*)malloc(c*sizeof(int));
  for(i=0; i<r; i++)
     for(j=0;j<c;j++)
        b[i][j]=5;
/*printing the array content*/
for(i=0; i<r; i++){
 for(j=0;j<c;j++)
     printf("%d ",b[i][j]);
      printf("\n");}
```

Struts as Function Parameters

Example:

- Implementing search for a binary search tree

Defining node

```
struct node
{
int key_value;
struct node *left;      //pointer to the left child
struct node *right;     //pointer to the right child
};
```

First defining the struct node

```
struct node
{
int key_value;
struct node *left;      //pointer to the left child
struct node *right;     //pointer to the right child
};
```

Search Implementation

```
struct node *search(int key, struct node *root)  //return type is a pointer
{
if( root != 0 )
{
if(key==root->key_value)
{
return root;
}
else if(key<root->key_value)
{
return search(key, root->left);
}
else
{
return search(key, root->right);
}
}
else return 0;
};
```