CS11001/CS11002 Programming and Data Structures (PDS)

(Theory: 3-1-0)

Pointers and 2-D arrays

Pointers and 2-D arrays

- Pointer to Array:

  data_type (*ptrvar) [col])

(Note the parenthesis, without which it refers to something else, as the [] has a greater precedence than *)

equivalent to

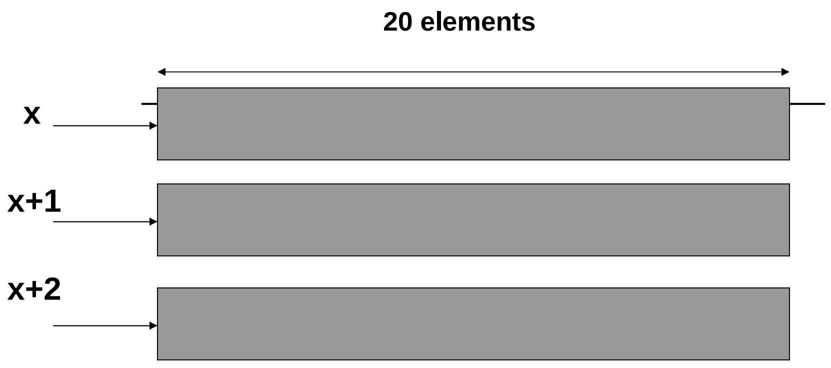
  data_type array[row][col]

Here, ptrvar is a ptrvar is pointer to a **contiguous,** one-dimensional array of col elements of type data_type

Thus, ptr+i refers to the pointer to the ith row.

Example

- int (*x)[20]; refers to a pointer to an array of 20 integers.
- Thus x points to the first element of the first row having 20 elements.



Extension to 3-D arrays

- float (*t)[20][30]
  - Thus the variable t is a pointer to a 2-D array of dimension [20][30].
  - (t+1) points to the second 20x30 array.

Accessing a 2-D array

- Consider int x[2][5]
  - this is the same as *(*(x+2)+5)
  - x is a pointer to the 0th row of a 1-D array
  - x+2 is a pointer to the 2nd row
  - *(x+2) is a pointer to the first element of the 2nd row
  - *(x+2)+5 is the pointer to the 5th element in the 2nd row
  - *(*(x+2)+5) refers to x[2][5]

Compatibility of [ ][MAXCOL] and (*)[ ]

```
#define MAXROW 4
 #define MAXCOL 5
int barsum ( int A[][MAXCOL] , int r , int c )
 { int i, j, s;
    int (*p)[MAXCOL];
    s = 0; p = A;
   for (i=0; i<r; ++i)
           for (j=0; j<c; ++j)
                s += p[i][j];
      return s;
   }
```

Compatibility of Pointers to arrays

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

#define MAXROW 4
#define MAXCOL 5

int barsum (int A[][MAXCOL] , int r , int c )
 { int i, j, s;
    int (*p)[MAXCOL];
    s = 0; p = A;
   for (i=0; i<r; ++i)
           for (j=0; j<c; ++j)
                s += p[i][j];
      return s;
   }
```

Compatibility of Pointers to arrays

```
main()
{
 int i, j;
 int r, c;
 //int A[4][5];
 int (*A)[5];

 printf("Enter the no of rows and cols\n");
 scanf("%d %d",&r, &c);

  //r pointers to arrays of size c integers are malloc-ed

 A = (int (*)[c])malloc(r*sizeof(int [c]));

 for(i=0;i<r;i++)
   for(j=0;j<c;j++)
     scanf("%d",&A[i][j]);

  printf("s=%d\n",barsum(A,r,c));
}
```

Compatibility of Pointers to arrays

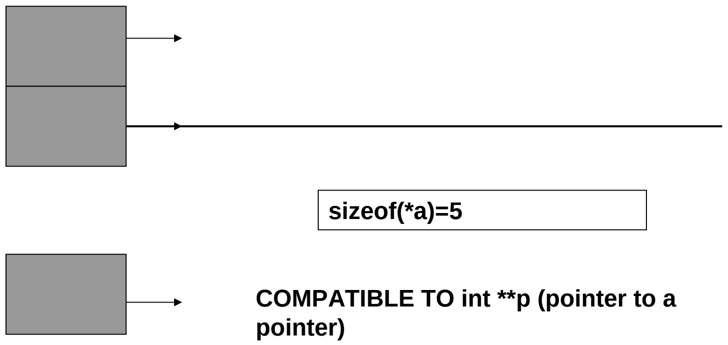
```
#include <stdio.h>
#include <malloc.h>

#define MAXROW 4
#define MAXCOL 5

int barsum (int A[][MAXCOL] , int r , int c )
 { int i, j, s;
    int (*p)[MAXCOL];
    s = 0; p = A;
   for (i=0; i<r; ++i)
           for (j=0; j<c; ++j)
                s += p[i][j];
     printf("%x %x %d\n", p, *p, **p);
     return s;
   }
```

Array of Pointers and Pointers to Pointers

- int *a[5]
  - array of 5 integer pointers



sizeof(*a)=5

COMPATIBLE TO int **p (pointer to a pointer)

Array of Pointers and its compatibility

```
#include <stdio.h>
#include <malloc.h>
#define MAXROW 4
#define MAXCOL 5

int barsum (int **A, int r , int c )
 { int i, j, s;
    int *p[MAXROW];
    s = 0;
    for(i=0;i<r;i++)
        p[i] = A[i];
 printf("\n%x %x %d\n", p, *p, **p);

   for (i=0; i<r; ++i)
          for (j=0; j<c; ++j)
               s += p[i][j];
      return s;
   }
```

Array of Pointers and its compatibility

```
main()
{
 int i, j;
 int r, c;
  int *A[MAXROW];

 printf("Enter the no of rows and cols\n");
 scanf("%d %d",&r, &c);

 for(i=0;i<r;i++)
    A[i]=(int *)malloc(c*sizeof(int));

for(i=0;i<r;i++)
   for(j=0;j<c;j++)
     scanf("%d",&A[i][j]);

  printf("s=%d\n",barsum(A,r,c));

}
```

Multiplication of matrices

```
void  matread(int **a, int m, int n)
{
  int i, j;
  for(i=0;i<m;i++)
   for(j=0;j<n;j++)
     {
        scanf("%d",&a[i][j]);
     }

}
```

The mat-write function

```
void  matwrite(int **a, int m, int n)
{
  int i, j;
  printf("m=%d n=%d\n",m,n);

  for(i=0;i<m;i++){
   for(j=0;j<n;j++)
    printf("%d ",a[i][j]);
   printf("\n");
  }

}
```

```
int** matmul(int **a, int **b, int m1,
     int n1, int m2, int n2)
{
  int i, j, k;
  int **c;
  c=malloc(5*sizeof(int *));

printf("Dimension of A: row=%d,
     col=%d\n",m1, n1);
printf("Dimension of B: row=%d,
     col=%d\n",m2, n2);

printf("Matrix A:\n");
matwrite(a,m1,n1);
printf("Matrix B:\n");
matwrite(b,m2,n2);

for(i=0;i<n2;i++)
  c[i]=malloc(5*sizeof(int));
```

```
if(n1 != m2)
   printf("Mult not defined\n");
 else{
   for(i=0;i<m1;i++)
      for(j=0;j<n2;j++)
          c[i][j]=0;

   for(i=0;i<m1;i++)
     for(j=0;j<n2;j++)
         for(k=0;k<n1;k++)
              c[i][j]=c[i][j]+a[i][k]*b[k][j];
   }


return(c);
}
```

The create function

```
int** create(int m, int n)
{ int i;
  int **a;

  a=malloc(5*sizeof(int *));

 for(i=0;i<5;i++)
  a[i]=malloc(5*sizeof(int));
return(a);

}
```

The main function

```
#include<stdio.h>
#include<malloc.h>
main()
{
 int **a, **b;
 int **c;
 int m1, n1, m2, n2, i, j;
 void matread(int **,int ,int );
 void matwrite(int **,int ,int );
 int** create(int , int );

 int** matmul(int **, int **, int, int, int,
     int);

 scanf("%d %d",&m1,&n1);
 scanf("%d %d",&m2, &n2);
```

```
a=create(m1, n1);
 b=create(m2, n2);

 printf("Enter Matrix A\n");
 matread(a,m1,n1);
 matwrite(a,m1,n1);
 printf("Enter Matrix B\n");
 matread(b,m2,n2);
 matwrite(b,m2,n2);
 c=matmul(a,b,m1,n1,m2,n2);

  printf("result matrix\n");
  for(i=0;i<m1;i++){
   for(j=0;j<n2;j++)
    printf("%d ",c[i][j]);
   printf("\n");
  }
 matwrite(c,m1,n2);
}
```

Memory Organization

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

   #define ROWSIZE 4
   #define COLSIZE 5

   int A[ROWSIZE][COLSIZE];
   int (*B)[COLSIZE];
   int *C[ROWSIZE];
   int **D;
```

Statically allocated 2-D array

```
int main ()
  {
    int i, j;

    printf("\nArray A\n");
    printf("sizeof(*A) = %d\n",sizeof(*A));
    i=0;
    printf("A[i]=%4d, A=%4d\n",(int)A[i], (int)A);

    printf("                    j=0   j=1   j=2   j=3   j=4\n");
    printf("                 +-------------------------------+\n");
    for (i=0; i<ROWSIZE; ++i) {
       printf("A[%d] = %4d :  i=%d  |", i, (int)A[i]-(int)A, i);
       for (j=0; j<COLSIZE; ++j)
          printf("%6d", (int)(&A[i][j])-(int)A);
       printf(" |\n");
    }
    printf("                 +-------------------------------+\n");
```

Pointer to array

```
printf("\nArray B\n");
   B=(int (*)[COLSIZE])malloc(ROWSIZE * sizeof(int [COLSIZE]));

   printf("sizeof(*B) = %d\n",sizeof(*B));
   i=0;
   printf("B[i]=%4d, B=%4d\n",(int)B[i], (int)B);

    printf("                  j=0   j=1   j=2   j=3   j=4\n");
    printf("               +------------------------------+\n");
    for (i=0; i<ROWSIZE; ++i){
        printf("B[%d]=%4d ", i, (int)(&B[i]));
      for(j=0;j<COLSIZE;++j)
          printf("%6d ",(int)(&B[i][j])-(int)B);
      printf(" |\n");
      printf("\n");
    }
```

Array of pointers

```
printf("               +------------------------------+\n");

   printf("\nArray C\n");
   for (i=0; i<ROWSIZE; ++i)
      C[i] = (int *)malloc(COLSIZE * sizeof(int));
   printf("sizeof(*C) = %d\n",sizeof(*C));
   i=0;
   printf("C[i]=%4d, C=%4d\n",(int)C[i], (int)C);

    printf("                  j=0   j=1   j=2   j=3   j=4\n");
    printf("               +------------------------------+\n");
    for (i=0; i<ROWSIZE; ++i){
        printf("C[%d]=%4d ", i, (int)(&C[i]));

      for (j=0; j<COLSIZE; ++j)
          printf("%6d ",(int)(&C[i][j])-(int)C);
      printf(" |\n");
      printf("\n");
    }
```

Pointer to Pointer

```
printf("                   +-----------------------------+\n");

    printf("\nArray D\n");
    D=(int **)malloc(ROWSIZE * sizeof(int *));
    for(i=0;i<COLSIZE;i++)
       D[i]=(int *)malloc(COLSIZE*sizeof(int));

    printf("sizeof(*D) = %d\n",sizeof(*D));
    i=0;
    printf("D[i]=%4d, D=%4d\n",(int)D[i], (int)D);

     printf("                      j=0   j=1   j=2   j=3   j=4\n");
     printf("                   +-----------------------------+\n");
     for (i=0; i<ROWSIZE; ++i){
          printf("D[%d]=%4d ", i, (int)(&D[i]));
       for(j=0;j<COLSIZE;++j)
            printf("%6d ",(int)(&D[i][j])-(int)D);
       printf(" |\n");
       printf("\n");
     }
```

Statically allocated 2-D array

```
Array A
sizeof(*A) = 20
A[i]=134520128, A=134520128
                        j=0   j=1   j=2   j=3   j=4
                    +------------------------------+
A[0] =    0 :  i=0  |     0     4     8    12    16 |
A[1] =   20 :  i=1  |    20    24    28    32    36 |
A[2] =   40 :  i=2  |    40    44    48    52    56 |
A[3] =   60 :  i=3  |    60    64    68    72    76 |
                    +------------------------------+
```

Pointer to array

```
Array B
sizeof(*B) = 20
B[i]=160055304, B=160055304
                  j=0   j=1   j=2   j=3   j=4
               +-------------------------------+
B[0]=160055304     0     4     8    12    16  |

B[1]=160055324    20    24    28    32    36  |

B[2]=160055344    40    44    48    52    56  |

B[3]=160055364    60    64    68    72    76  |

               +-------------------------------+
```

Array of pointers

```
Array C
sizeof(*C) = 4
C[i]=160055392, C=134520212
                  j=0   j=1   j=2   j=3   j=4
               +-------------------------------+
C[0]=134520212 25535180 25535184 25535188 25535192 25535196  |

C[1]=134520216 25535204 25535208 25535212 25535216 25535220  |

C[2]=134520220 25535228 25535232 25535236 25535240 25535244  |

C[3]=134520224 25535252 25535256 25535260 25535264 25535268  |

               +-------------------------------+
```

Pointer to Pointer

```
Array D
sizeof(*D) = 4
D[i]=160055512, D=160055488
                    j=0   j=1   j=2   j=3   j=4
                +-------------------------------+
D[0]=160055488     24     28     32     36     40  |

D[1]=160055492     48     52     56     60     64  |

D[2]=160055496     72     76     80     84     88  |

D[3]=160055500     96    100    104    108    112  |

                +-------------------------------+
```