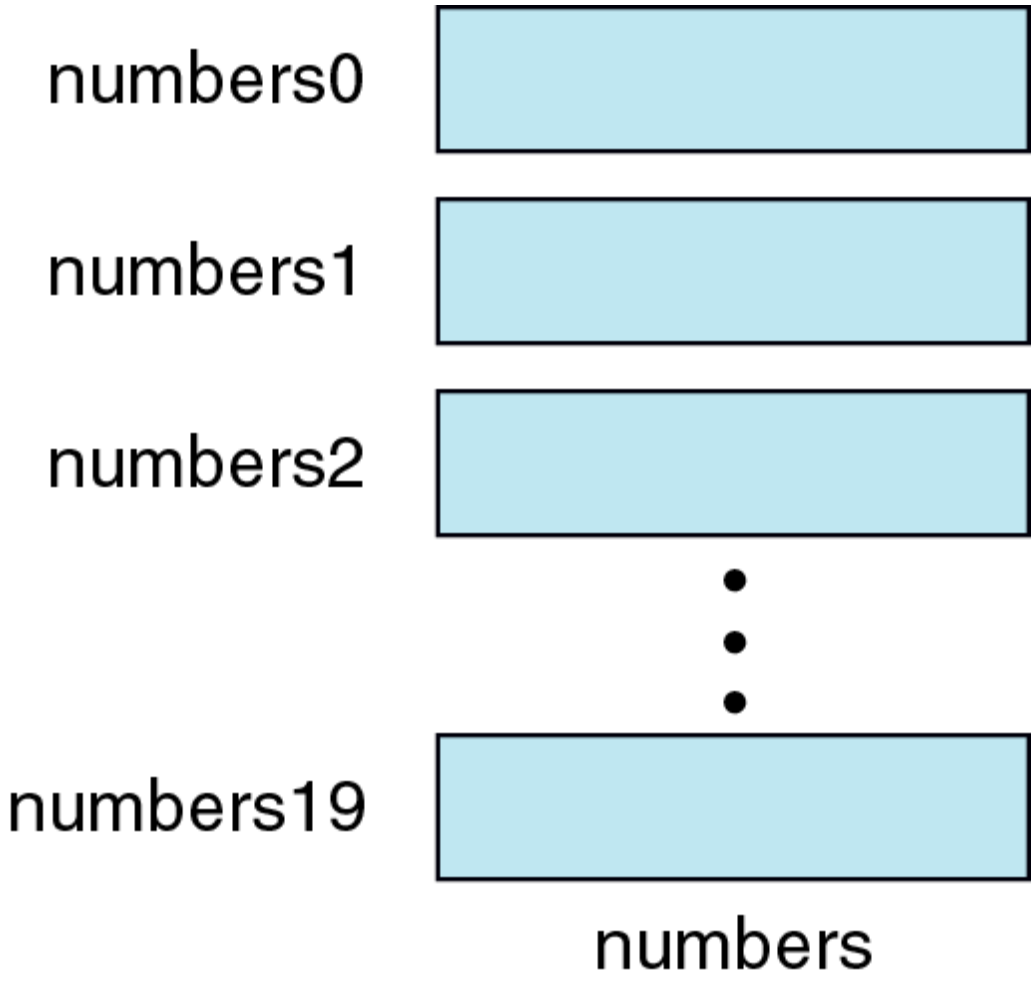
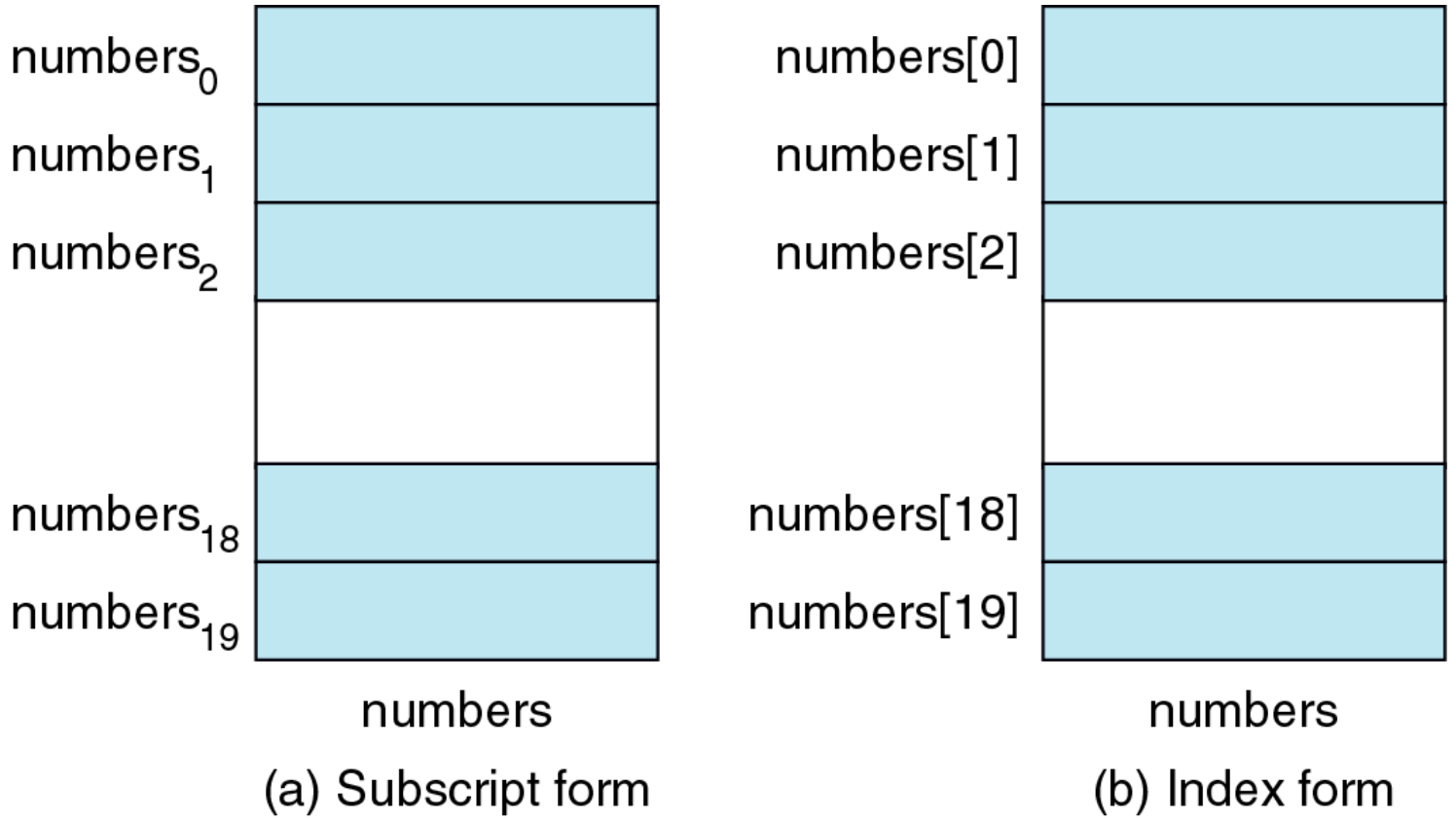
ARRAYS

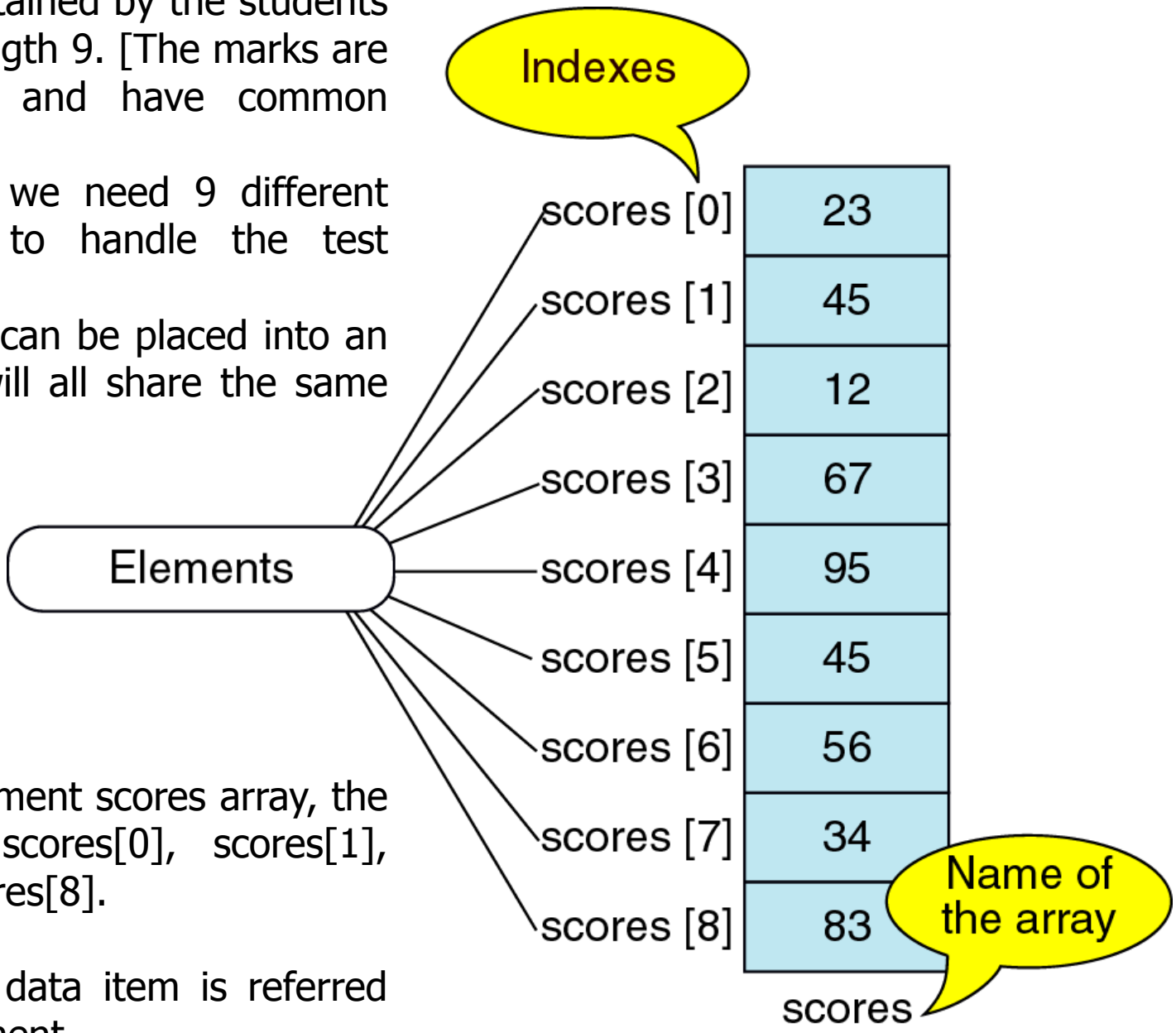
- Multiple data items that are of the same type and have common characteristics can be represented through a common name.

- The data items having the common properties can be allocated continuous memory locations and they can be referred by a single name.


numbers0
numbers1
numbers2
numbers19
numbers


numbers0
numbers1
numbers2
numbers18
numbers19
numbers
(a) Subscript form
numbers[0]
numbers[1]
numbers[2]
numbers[18]
numbers[19]
numbers
(b) Index form

- The test mark obtained by the students of a class of strength 9. [The marks are of type integer and have common characteristics.]
- In normal case, we need 9 different variable names to handle the test marks.
- Instead the data can be placed into an array and they will all share the same name 'scores'.



- Thus, in a 9 element scores array, the elements are scores[0], scores[1], scores[2],…,scores[8].

- Each individual data item is referred as an array element.

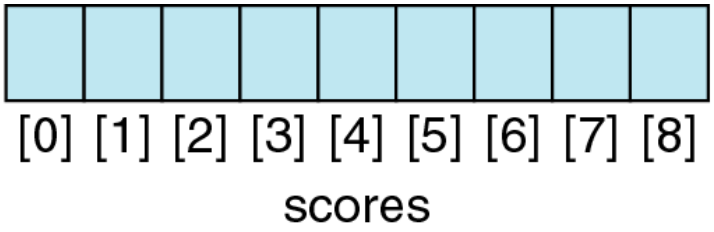
- Each array element is referred to by the array name followed by one or more subscripts with each subscript enclosed in square brackets.
- The elements of the array are individually addressed through their subscripts.
- Each subscript must be expressed as non-negative integer.
- The value of each subscript can be expressed as an integer constant, an integer variable, or a complex integer expression.
- The number of subscripts determines the dimensionality of the array.

- x[i] refers to an element in the one-dimensional array x.
- y[i][j] refers to an element in the two dimensional array y.
- z[i][j][k] refers to an element in the three dimensional array.
- Turbo C supports 7 dimensional array.

- An array must be declared and defined before it can be used.
- Declaration and definition tell the compiler the name of the array, the type of data and the number of elements.

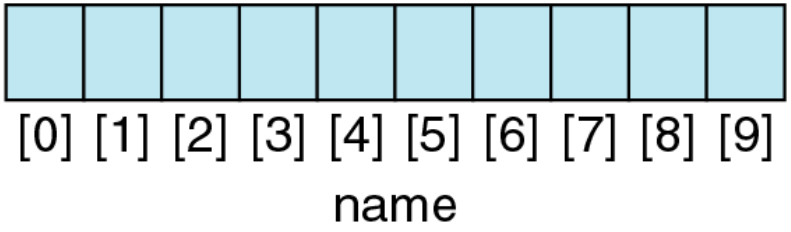
```
int scores [9];
```

type of each element



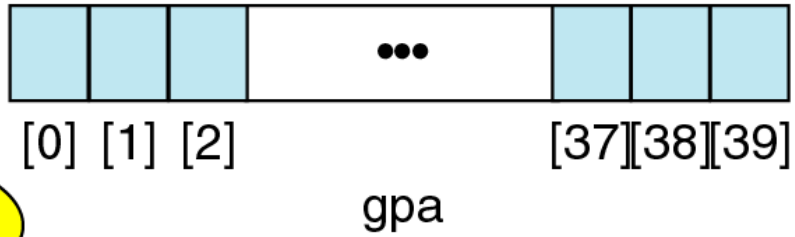

```
char name [10];
```

name of the array

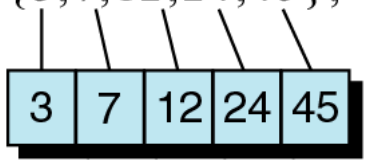



```
float gpa [40];
```

number of elements

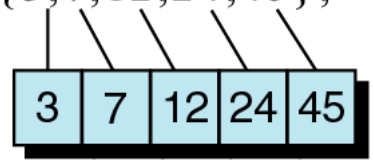


```
int numbers [5] = {3,7,12,24,45};
```



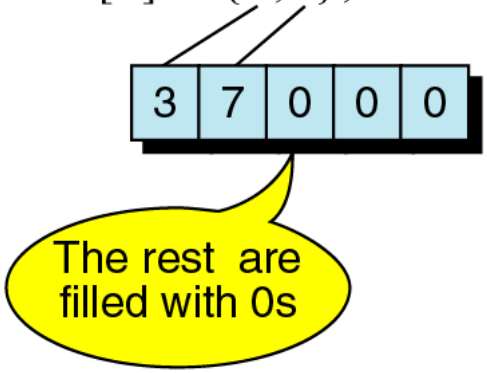
(a) Basic initialization

```
int numbers [ ] = {3,7,12,24,45};
```



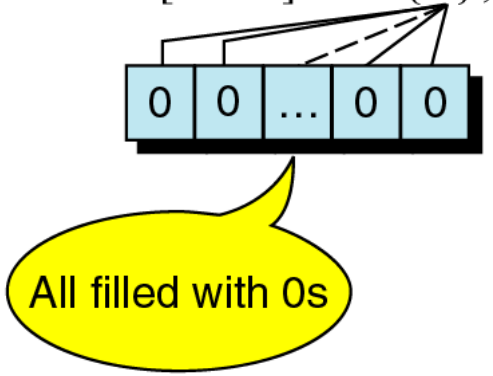
(b) Initialization without size

```
int numbers [5] = {3,7};
```

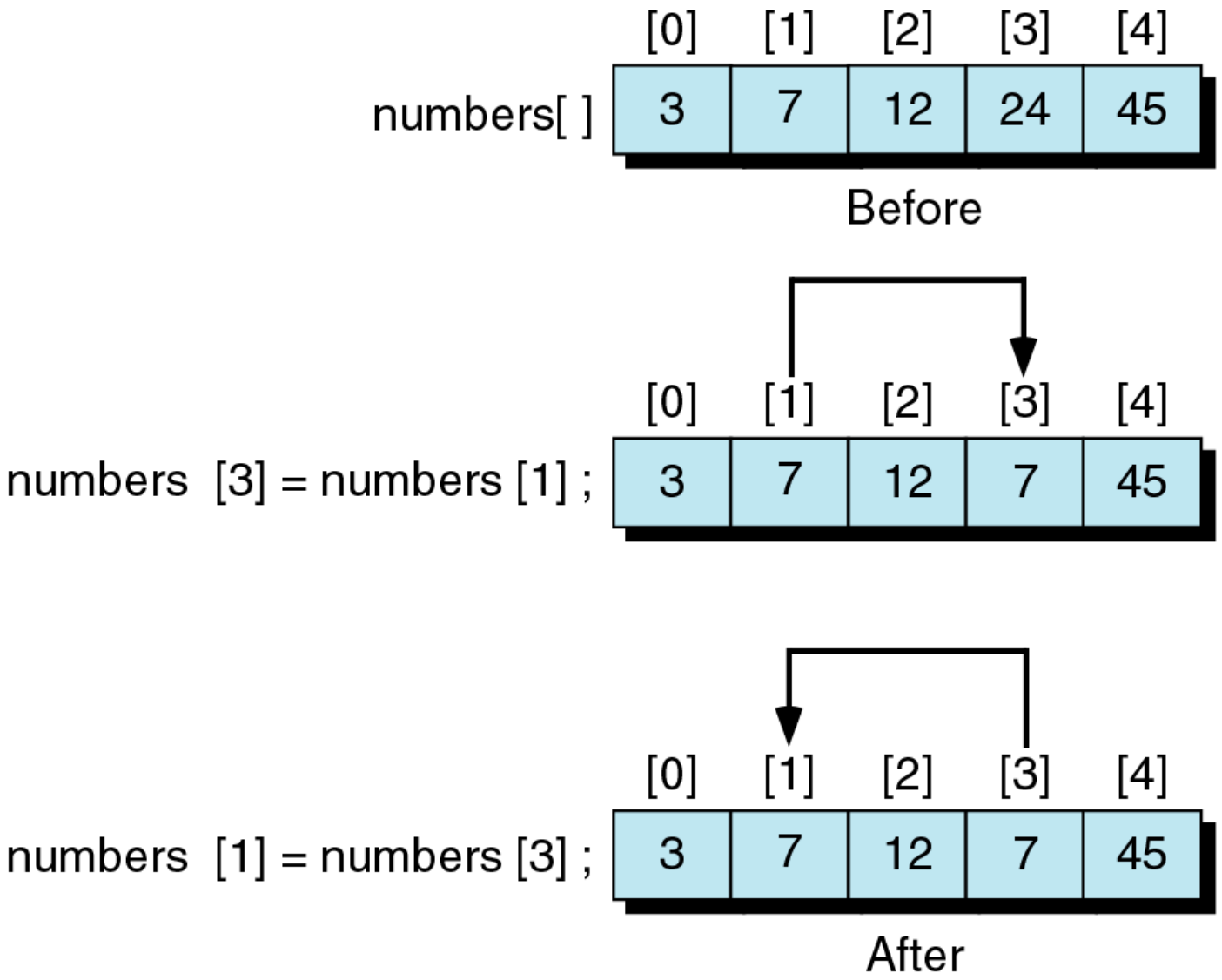


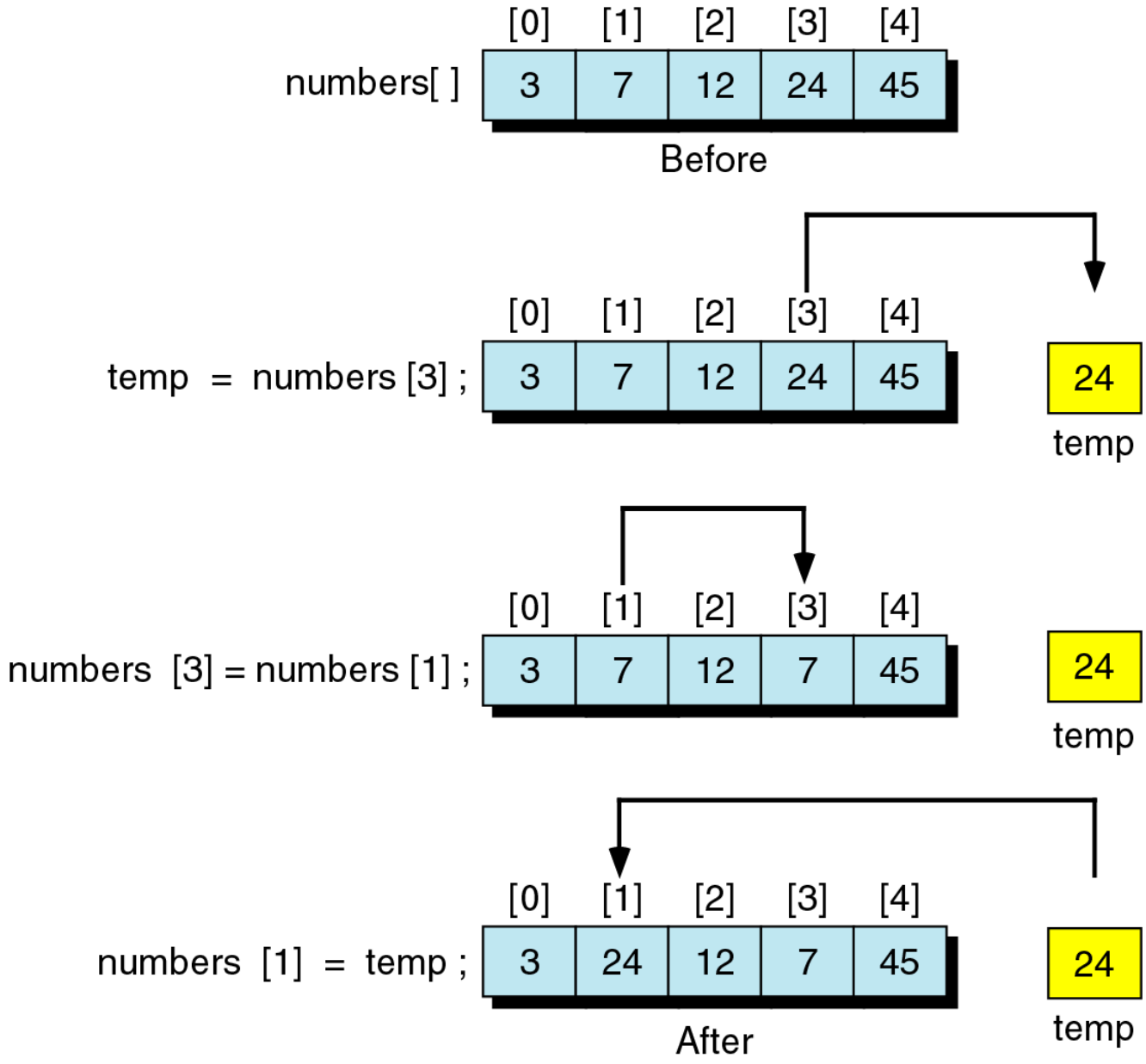
(c) Partial initialization

```
int lotsOfNumbers [1000] = n {0};
```



(d) Initialization to all zeros


[0] [1] [2] [3] [4]
numbers[ ]
3 7 12 24 45
Before
[0] [1] [2] [3] [4]
numbers [3] = numbers [1] ;
3 7 12 7 45
[0] [1] [2] [3] [4]
numbers [1] = numbers [3] ;
3 7 12 7 45
After


[0] [1] [2] [3] [4]
numbers[ ]
3 7 12 24 45
Before
[0] [1] [2] [3] [4]
temp = numbers [3] ;
3 7 12 24 45
24
temp
[0] [1] [2] [3] [4]
numbers [3] = numbers [1] ;
3 7 12 7 45
24
temp
[0] [1] [2] [3] [4]
numbers [1] = temp ;
3 24 12 7 45
24
temp
After

Example 1:

```
int mark[266];
float height[20];
float weight[35];
char name[40];
```

Example 2:

How values can be assigned to the array while declaring them?

```
int score[5] = {90, 96,78,69,97};
```

The result of the above assignment is :

```
score[0] = 90        score[1] = 96        score[2] = 78
score[3] = 69        score[4] = 97
```

Example 3:

float x[3] = {0.1,0.2,0.7};

The result of the above assignment is :

x[0] = 0.1 x[1] = 0.2 x[2] = 0.7

Example 4:

char color[3] = {'r','e','d'};

The result of the above assignment is :

color[0] = 'r' color[1] = 'e' color[2] ='d'

Note: All individual array elements that are not assigned explicit initial values will automatically be set to zero;

Example 5:

```
int num[4] = {1,2};
```

The result of the above assignment is :

num[0] = 1 num[1] = 2
num[2] = 0 num[3] = 0

Example 6:

```
char name[6] = {'r', 'o' ,' s' , 'e'};
```

The result of the above assignment is :

name[0] = 'r' name[1] = 'o' name[2] = 's'
name[3] = 'e' name[4] = 0 name[5] = 0

The array size need not be specified explicitly when initial values are included as a part of an array definition.

Example 7:

```
int mark[ ] = {67,78,88};
```

The individual elements will be assigned as follows:
mark[0] = 67 mark[1] = 78 mark[2] = 88

How array elements can be accessed?
Indexing can access individual array element.

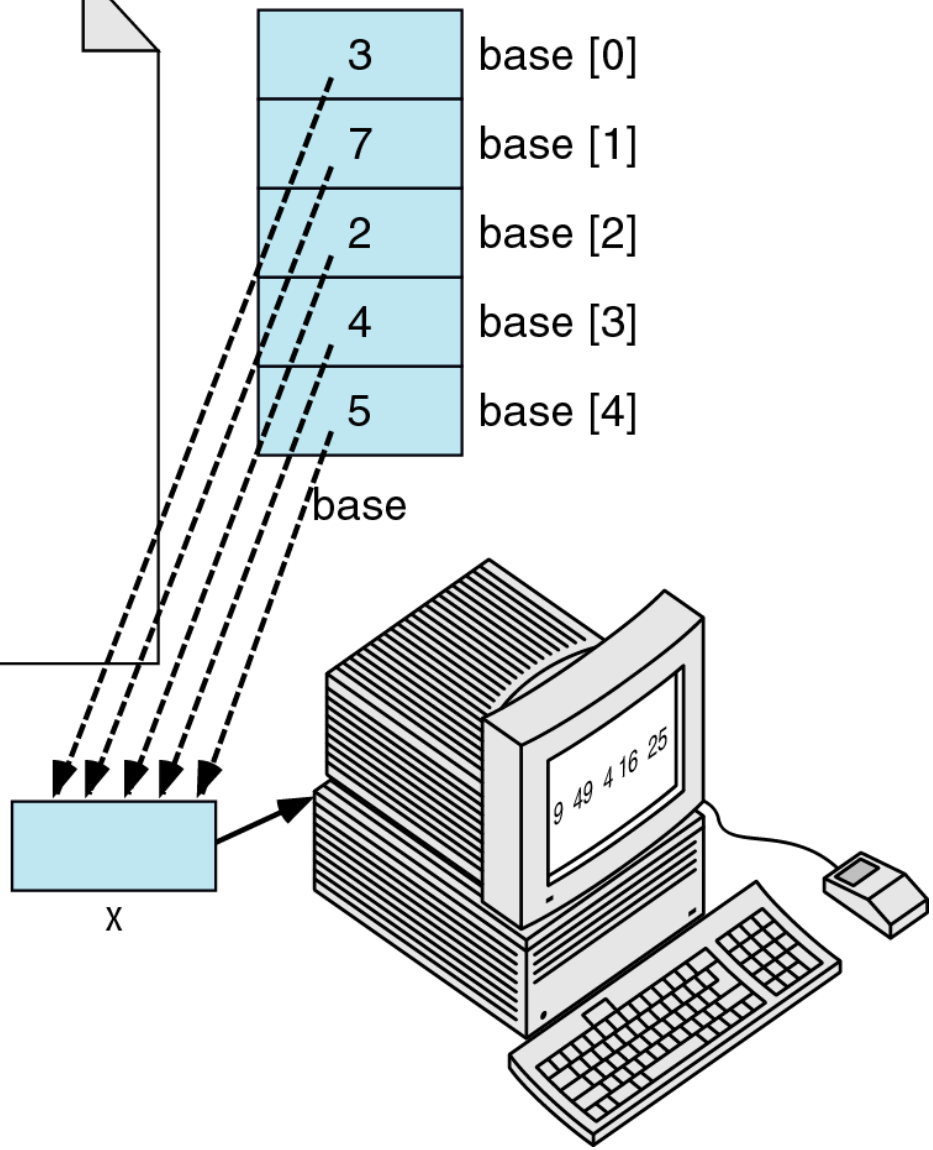
Example 8:

```
int score[3] = {20,12,88};
```

Then score[2] refers the value 88.

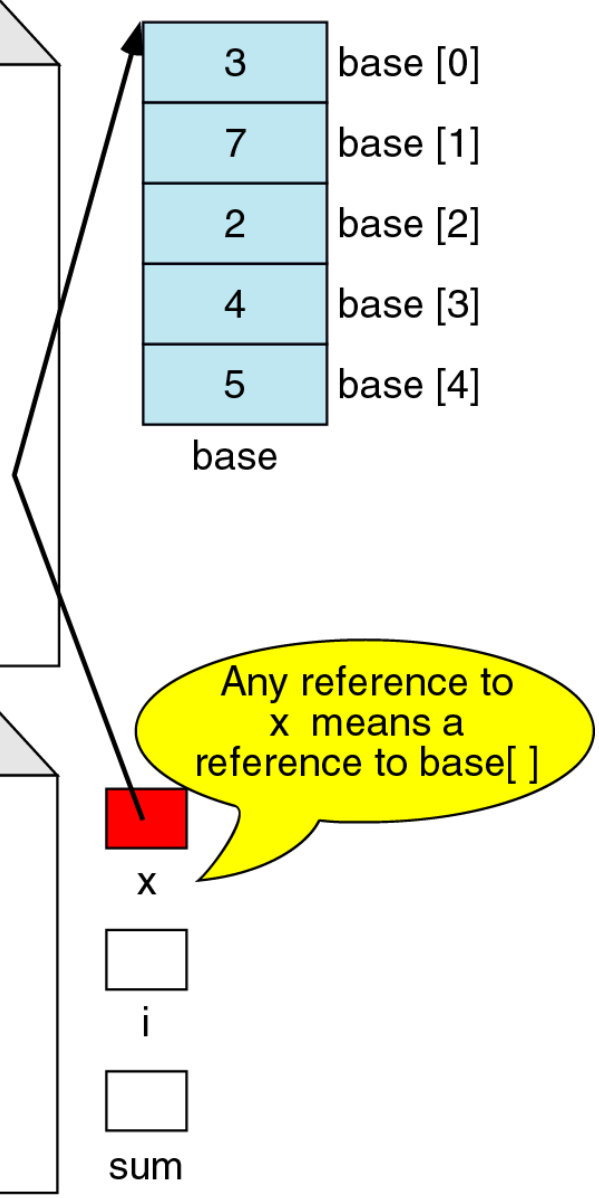
```c
#include <stdio.h>
/* Prototpye Declarations */
void print_square (int x);
int main (void)
{
   int i;
   int base[5] = {3, 7, 2, 4, 5};
   for (i = 0; i < 5; i++)
      print_square (base [i]);
   return  0;
} /* main */
```

```c
void print_square (int x)
{
   printf("%d ", x * x);
   return;
} /* print_square */
```



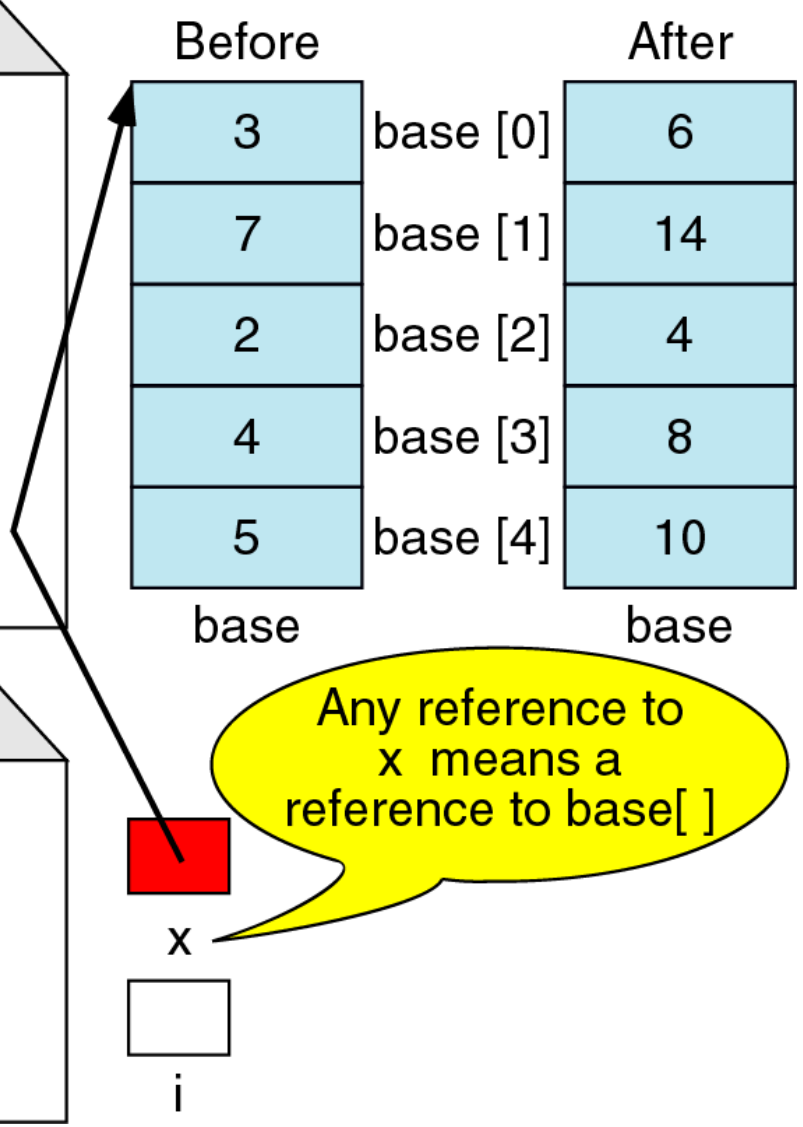
```c
#include <stdio.h>
/* Prototype Declarations */
double average (int x[]);

int main (void)
{
  double ave;
  int base[5] = {3, 7, 2, 4, 5};
  ...
  ave = average (base);
  ...
  return 0;
} /* main */
```

```c
double average (int x [])
{
  int i;
  int sum = 0 ;
  for (i = 0; i < 5; i++)
      sum += x [i];
  return (sum / 5.0);
} /* main */
```



```c
/* Prototype Declarations */
void multiply2 (int x[]);
#include <stdio.h>
int main (void)
{
  int base[5] = {3, 7, 2, 4, 5};
  multiply2 (base);
  return 0;
} /* main */
```

```c
void multiply2 (int x[])
{
  int i;
  for (i = 0; i < 5; i++)
      x[i] *= 2;
  return ;
} /* multiply2 */
```



```
/* Example 9 : */
/* program to find sum, average,*/
#include<stdio.h>
#define nsize 200
void main(void) {
float  num[nsize];
float sum,ave,sd;
int i,n;
sum  = 0.0;
printf("Number of data please");
scanf("%d",&n);
for(i=0;i<n;i++)     {
scanf("%f",&num[i]);
sum +=num[i];       }
ave = sum/ n;
printf("sum = %f, ave = %f\n",sum,ave);
return;
}
```

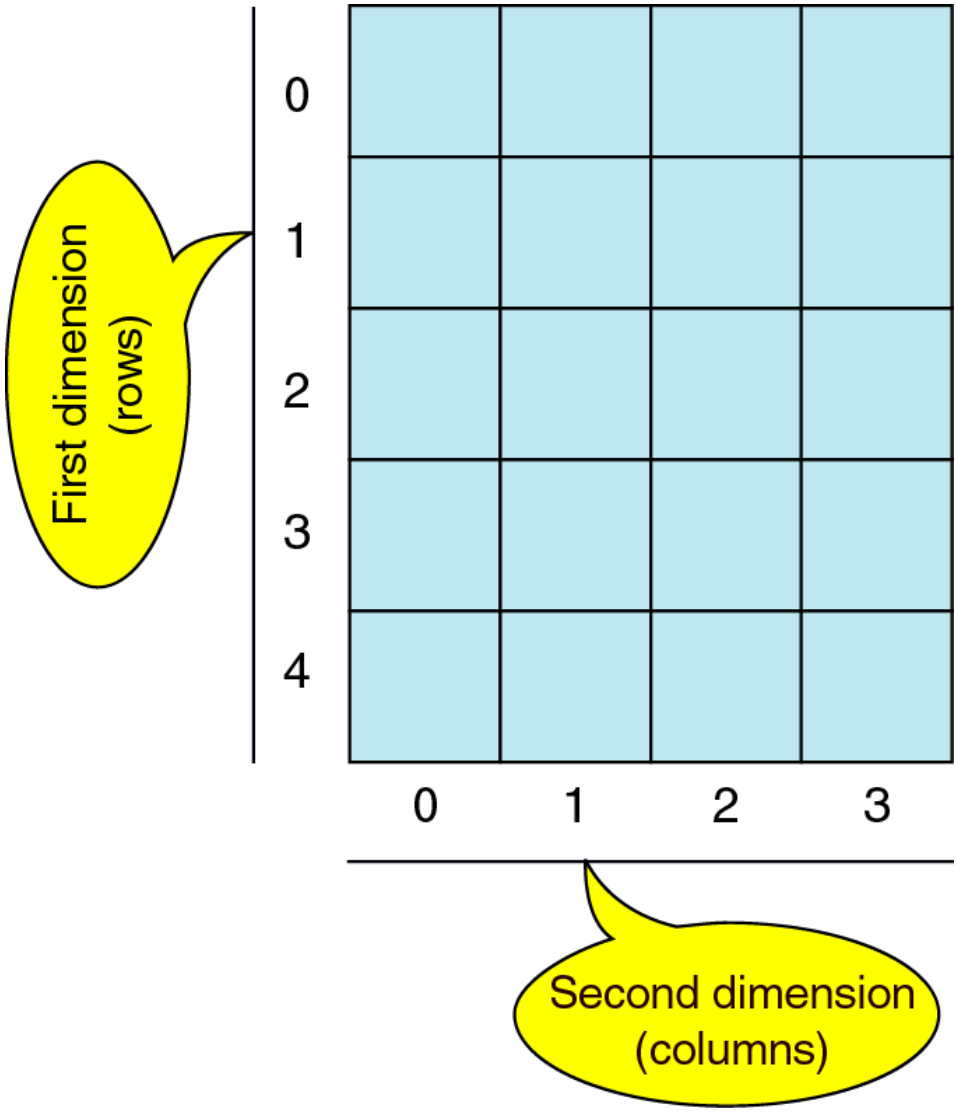
Example 12 :

```
/* program to read numbers and sort */

#include<stdio.h>
void readvec(float a[ ], int n);
void printvec(float a[ ],int n);
void sortnumber(float a[ ], int n);
void main(void) {
 int n;
 float a[10];
 n = 3;
 readvec(a,n);
 printvec(a,n);
 sortnumber(a,n);
 printvec(a,n);
 return;
}
```

```
void readvec(float a[ ], int n)
{
int i;
for(i=0;i<n;i++)
    scanf("%f",&a[i]);
return;
}
```

```
void printvec(float a[], int n)
{
int i;
for(i=0;i<n;i++)
  printf("%f\n",a[i]);
return;
}
```

```
void sortnumber(float x[ ], int n)
{
int item,i;
float temp;
for(item=0;item< n-1;item++)
   {
     for(i=item;i<n;i++)
          {
           if(x[i] < x[item])
             {
                    temp = x[item];
                   x[item] = x[i];
                   x[i] =temp;
              }
          }
   }
}
```


First dimension
(rows)
0
1
2
3
4
0
1
2
3
Second dimension
(columns)

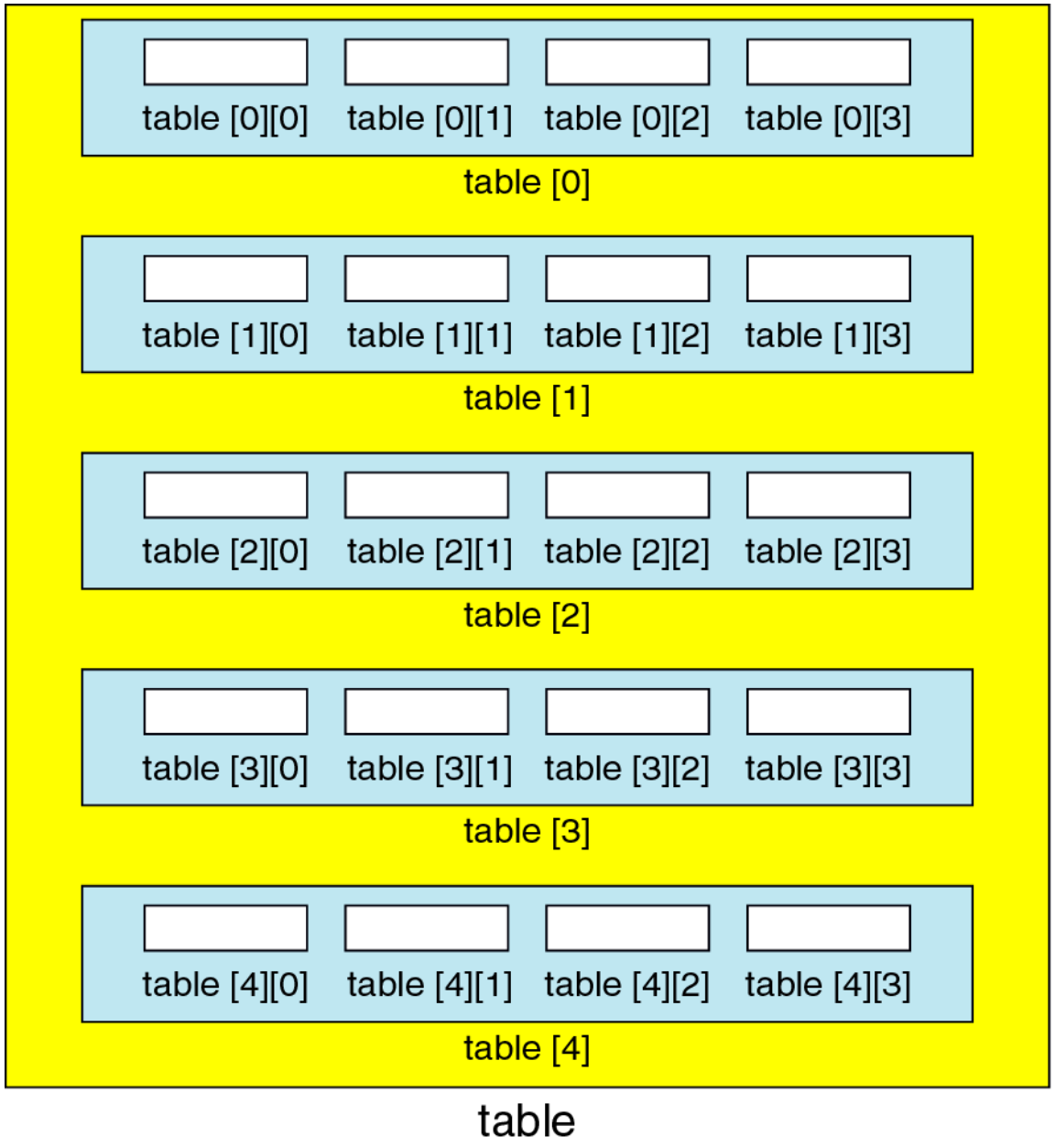
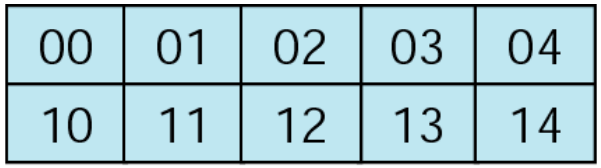
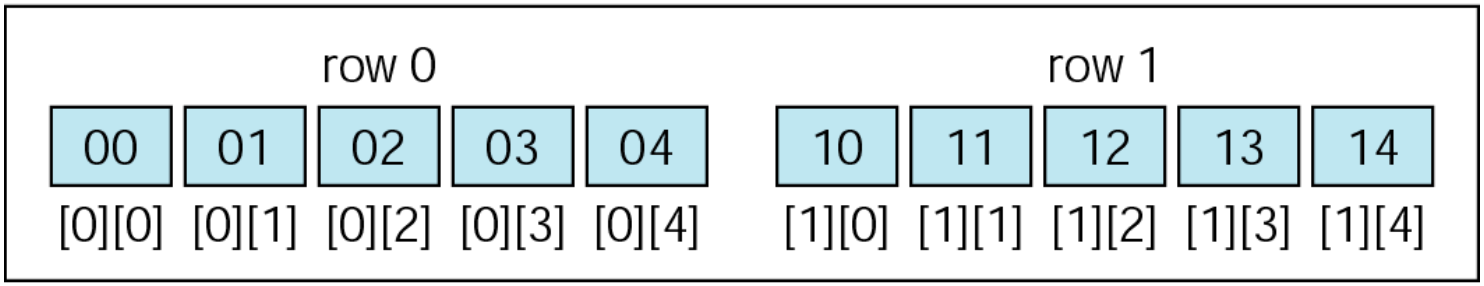


table [0][0]
table [0][1]
table [0][2]
table [0][3]
table [0]
table [1][0]
table [1][1]
table [1][2]
table [1][3]
table [1]
table [2][0]
table [2][1]
table [2][2]
table [2][3]
table [2]
table [3][0]
table [3][1]
table [3][2]
table [3][3]
table [3]
table [4][0]
table [4][1]
table [4][2]
table [4][3]
table [4]
table



User's view



Memory view

MULTI DIMENSIONAL ARRAY DEFINITIONS

The multi dimensional arrays are defined in much the same manner as one-dimensional arrays.

- A pair of square brackets is required for each dimension.
- A two dimensional array will require two pairs of square brackets.
- In general a multi-dimensional array can be written as:

  Data type array [exp 1][exp 2]…[exp n];

Example 13:

```
int val[3][4] = {1,2,3,4,5,6,7,8,9,10,11,12};
```

val[0][0] =1	val[0][1] =2
val[0][2] =3	val[0][3] =4
val[1][0] =5	val[1][1] =6
val[1][2] =7	val[1][3] =8
val[2][0] =9	val[2][1] =10
val[2][2] =11	val[2][3] =12

Example 14:

```
int val[3][4] = {     {1,2,3,4} ,
                      {5,6,7,8} ,
                      {9,10,11,12}          };
```

This definition results in the same initial assignments as in the last example

Example 15:

```
Int val[3][4] =       {       {1,2,3} ,
                              {4,5,6} ,
                              {7,8,9}       };
```

val[0][0] = 1 val[0][1] = 2
val[0][2] = 3 val[0][3] = 0

val[1][0] = 4 val[1][1] = 5
val[1][2] = 6 val[1][3] = 0

val[2][0] = 7 val[2][1] = 8
val[2][2] = 9 val[2][3] = 0

Example 16:

```
int val[3][4] = {1,2,3,4,5,6,7,8,9};
```

val[0][0] =1	val[0][1] = 2
val[0][2] =3	val[0][3] = 4
val[1][0] =5	val[1][1] = 6
val[1][2] =7	val[1][3] = 8
val[2][0] =9	val[2][1] = 0
val[2][2] =0	val[2][3] = 0

```c
#define MAX_ROWS  5
#define MAX_COLS  4
/* Prototype Declarations */
void print_square (int []);
int main (void)
{
  int row;
  int table [MAX_ROWS][MAX_COLS] =
          {
           { 0, 1, 2, 3 },
           { 10, 11, 12, 13 },
           { 20, 21, 22, 23 },
           { 30, 31, 32, 33 },
           { 40, 41, 42, 43 }
          }; /* table */
  ...
  for (row = 0; row < MAX_ROWS; row++)
     print_square (table [row]);
  ...
  return  0;
} /* main */
```

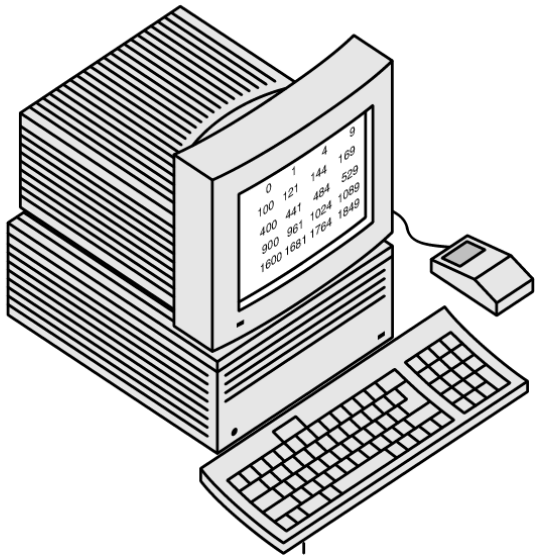
table

0	1	2	3
10	11	12	13
20	21	22	23
30	31	32	33
40	41	42	43

address of a row

x

```c
void print_square (int x[])
{
   int  col;
   for (col = 0; col < MAX_COLS; col++;)
        printf("%6d", x[col] * x[col]);
   printf ("\n");
   return;
} /* print_square */
```



```c
#define MAX_ROWS  5
#define MAX_COLS  4
/* Prototype Declarations */
double average (int [][MAX_COLS]);
int main (void)
{
  double ave;
  int    table[MAX_ROWS][MAX_COLS] =
         {
          { 0, 1, 2, 3 },
          { 10, 11, 12, 13 },
          { 20, 21, 22, 23 },
          { 30, 31, 32, 33 },
          { 40, 41, 42, 43 }
          }; /* table */
  ...
  ave  =  average (table);
  ...
  return 0;
} /* main */
```

```c
double average (int x[][MAX_COLS])
{
  int    i;
  int    j;
  double sum = 0;
  for (i = 0; i < MAX_ROWS; i++)
       for (j = 0; j < MAX_COLS; j++)
            sum += x [i] [j];
   return(sum / (MAX_ROWS *  MAX_COLS);
} /* average */
```

Example 17:

```
/* program in C to Add 2 matrices */
#include<stdio.h>
#define NROWA 3
#define NCOLA 3
#define NROWB 3
#define NCOLB 3
void main(void)
{
  float  a[NROWA][NCOLA];
  float b[NROWB][NCOLB], c[NROWA][NCOLB];
  int i,j,k;
```

continue

```
/* read the matrix a */

  for(i=0;i<NROWA;i++)
     {
      for(j=0;j<NCOLA;j++)
          scanf("%f",&a[i][j]);
     }

  /* read the matrix b */

  for(i=0;i<NROWB;i++)
     {
      for(j=0;j<NCOLB;j++)
      scanf("%f",&b[i][j]);
      }
```

continue

```
/* compute a+b */

 for(i=0;i<NROWA;i++)
    {
     for(j=0;j<NCOLB;j++)
         {
             c[i][j] = a[i][j] + b[i][j];
         }
     }
 /* print the matrix a */

   for(i=0;i<NROWA;i++)
    {
     for(j=0;j<NCOLA;j++)
     printf("%f\t",a[i][j]);
     printf("\n");
    }
```

continue

```
/* print the matrix b */

for(i=0;i<NROWB;i++)
   {
    for(j=0;j<NCOLB;j++)
        printf("%f\t",b[i][j]);
    printf("\n");
    }

/* print the matrix c */

for(i=0;i<NROWA;i++)
   {
    for(j=0;j<NCOLB;j++)
         printf("%f\t",c[i][j]);
    printf("\n");
    return;
    }
```

Example 18:

```
/* program in C to multiply 2 matrices */

#include<stdio.h>
#define NROWA 3
#define NCOLA 3
#define NROWB 3
#define NCOLB 4

void main(void)
{
 float a[NROWA][NCOLA],b[NROWB][NCOLB],c[NROWA][NCOLB];
  int i,j,k;
```

continue

```
/* read the matrix a */

 for(i=0;i<NROWA;i++)
    {
     for(j=0;j<NCOLA;j++)
          scanf("%f",&a[i][j]);
    }

/* read the matrix b */

 for(i=0;i<NROWB;i++)
    {
     for(j=0;j<NCOLB;j++)
          scanf("%f",&b[i][j]);
     }
```

continue

```
/* compute a*b */
 for(i=0;i<NROWA;i++)
    {
     for(j=0;j<NCOLB;j++)
          {
           c[i][j] = 0.0;
           for(k=0;k<NCOLA;k++)
               c[i][j] = c[i][j] + a[i][k]*b[k][j];
          }
     }

 /* print the matrix a */
   for(i=0;i<NROWA;i++)
    {
     for(j=0;j<NCOLA;j++)
           printf("%f\t",a[i][j]);
     printf("\n");
    }
```

continue

```
/* print the matrix b */
 for(i=0;i<NROWB;i++)
    {
     for(j=0;j<NCOLB;j++)
          printf("%f\t",b[i][j]);
     printf("\n");
     }

/* print the matrix c */
 for(i=0;i<NROWA;i++)
    {
     for(j=0;j<NCOLB;j++)
           printf("%f\t",c[i][j]);
     printf("\n");
     return;
    }
```

THANK YOU