ARRAYS, POINTERS

prepared by Senem Kumova Metin
modified by İlker Korkmaz

What is an Array?

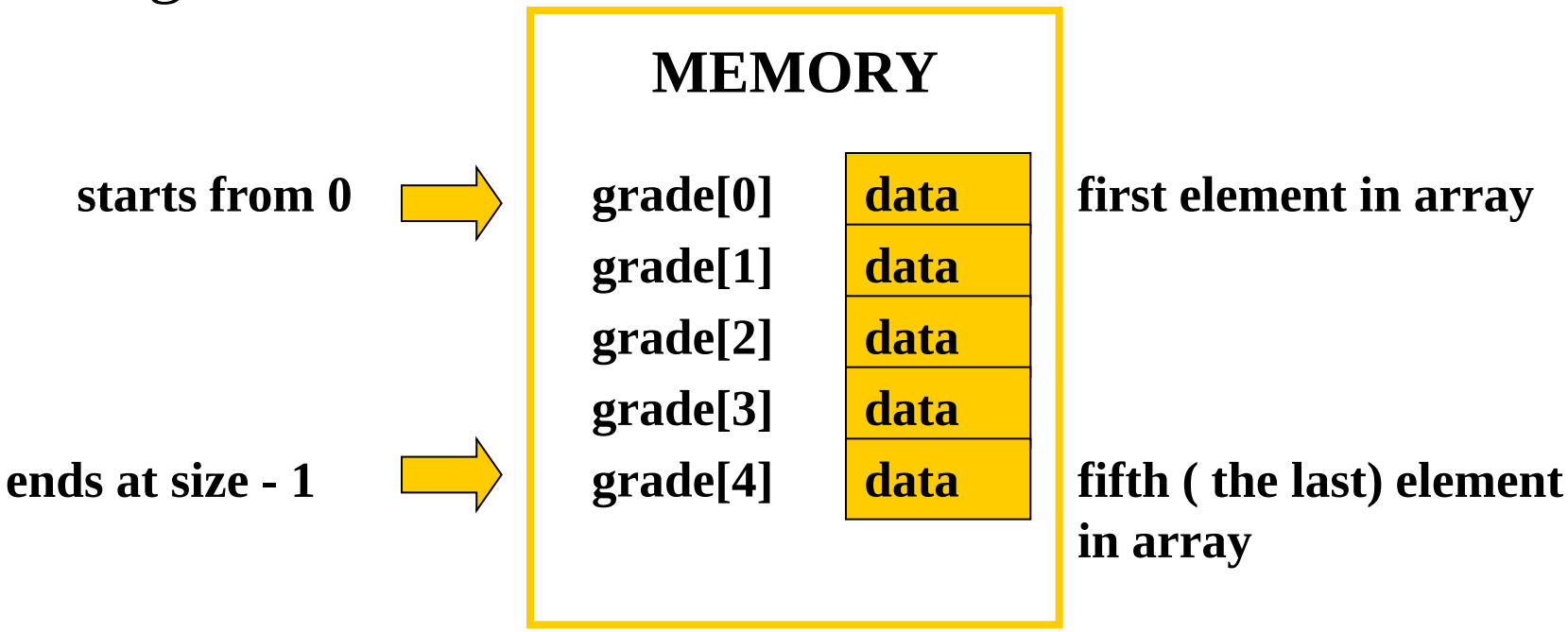
- int grade0, grade1, grade2;

- int grade [3];

- Arrays represent a group of homogenous typed values

One Dimensional Arrays (1)

- type name_of_array[size];
- int grade[5];



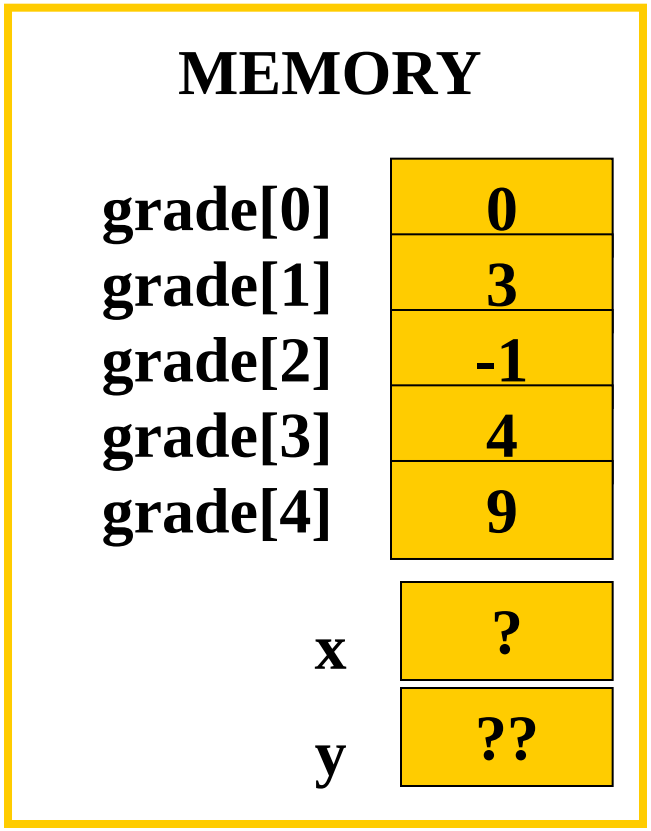
One Dimensional Arrays (2)

```
int grade[5], x=9, y=0;
grade[0]=0;
grade[3]=4;
grade[2]=-1;

grade[1]=grade[3]+grade[2];

grade[4]= x;

x= grade[2];  //the value of x ?
y= grade[grade[3]]; //the value of y ??
```



Array Initialization

- float f1[]= { 1.0, 1.1, 1.2, 1.3 }; /* declaration without size */
- float f2[4]= { 1.0, 1.1, 1.2, 1.3 }; /* declaration with size */
- char z[] =“abc”; // special char arrays: strings
- char z[] = {‘a’, ‘b’, ‘c’, ‘\0’}

- int a[80]={0}; // initializes all the elements to zero

Example: One Dimensional Arrays

```
/* array1.c*/
main()
{
    int x[10], k;
    for(k=0;k<10;k++)
        {       x[k]=k;
                printf("x[%d] = %d\n", k, x[k]);
                //what will be the output ???
        }
}
```

Two Dimensional Arrays (1)

```
#define R 4
#define C 5
int x[R][C];
```

	column 0	column 1	column 2	….	column C-1
row 0	x[0][0]	x[0][1]	x[0][2]	….	x[0][C-1]
row 1	x[1][0]	x[1][1]	x[1][2]	….	x[1][C-1]
row 2	x[2][0]	x[2][1]	x[2][2]	….	x[2][C-1]
row 3	x[3][0]	x[3][1]	x[3][2]	….	x[3][C-1]
….	….	….	….	….	….
row R-1	x[R-1][0]	x[R-1][1]	x[R-1][2]	….	x[R-1][C-1]

Two Dimensional Arrays (2)

x[0][0]	data
x[0][1]	data
x[0][2]	data
x[0][3]	data
x[1][0]	data
x[1][1]	data
x[1][2]	data
x[1][3]	data
x[2][0]	data
x[2][1]	data
x[2][2]	data
x[2][3]	data

1. row , 1. column

1.row , 2. column

1.row , 3. column

1.row , 4. column

How can x be declared ?

→ int x[?][?];

3. row , 1. column

3.row , 2. column

3.row , 3. column

3.row , 4. column

Two Dimensional Array Initialization

- int y[2][3] = {1,2,3,4,5,6};
- int y[2][3] = {{1,2,3},{4,5,6}};
- int y[][3] = {{1,2,3},{4,5,6}};

POINTERS (1)

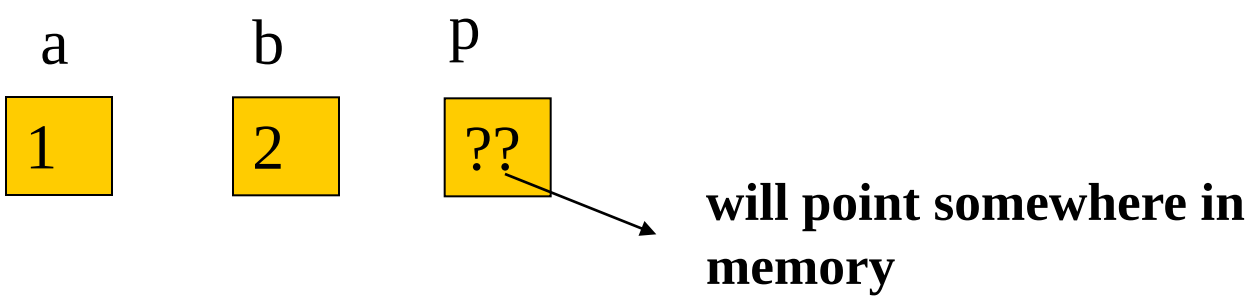
- int v = 5;
  - v is the identifier of an integer variable
  - 5 is the value of v
  - &v is the location or address of the v inside the memory
  - & means " the address of "

- Pointers are used in programs to access memory.

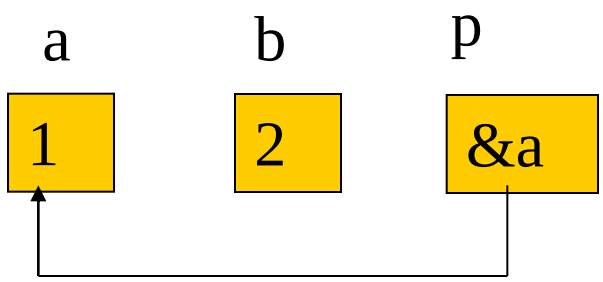
```
int v = 5;
int * p;  //  p is the identifier of a "pointer to an integer"
p=&v;  // p is assigned with the address of v
p=0;     // OR   p = NULL;
```

POINTERS (2)

int a=1, b=2, *p;

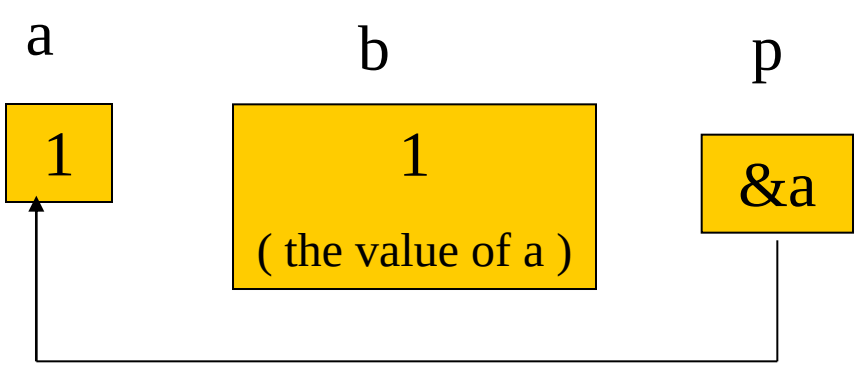


p=&a;



POINTERS (3)

b=*p;



// equivalent to b=a

Example: Pointers

EXAMPLE:

```
#include <stdio.h>
int main(void)
{
   int   i = 7, j , *k;

   k = &i;
   printf("%s%d\n%s%p\n", "   Value of i: ", *k, "Location of i: ",  k);
   j=*k;
   return 0;
}
```

CALL BY VALUE

/* Whenever variables are passed as arguments to a function, their values are copied to the function parameters , and the variables themselves are not changed in the calling environment.*/

```
int main()
{   int a=20; int b=30;
    swap (a, b)
    printf("%d  %d: ", a, b);
    return 0;
}
void swap(int x, int y)
{
    int tmp;
    tmp=x;
    x=y;
    y=tmp;
    return;
}
```

CALL BY REFERENCE

/* Whenever addresses of variables are passed as arguments to a function, their values shall be changed in the calling environment.*/

```
void main()
{   int a=20; int b=30;
    swap (&a, &b)
    printf("%d  %d: ", a, b);
}

void swap(int *x, int *y)
{   int tmp;
    tmp = *x; // get value pointed by x.
    *x = *y; // assign value pointed by y to x
    *y = tmp;
    return;  }
```

Relationship between “pointers” and “arrays” (1)

/* The name of an array is the adress or the pointer to the first element of the array. */

```
int a[5] , *p;
p=a; // OR   p=&a[0];
```

Relationship between “pointers” and “arrays” (2)

```
&a[0] equals to a    then  a[0] equals to *a
&a[1] equals to a+1  then  a[1] equals to *(a+1)
&a[2] equals to a+2  then  a[2] equals to *(a+2)
……
&a[i] equals to a+i  then  a[i] equals to *(a+i)
```

- **EXAMPLE:**

```
int a [5]={1,2,3,4,5};
int *p;
printf(“%d”,a[0]); printf(“%d”,*a);
printf(“%d”,a[2]); printf(“%d”,*(a+2));
p=&a[4];
p=a+4;
```

Storage mapping

- the mapping b/w pointer values and array indices
- EXAMPLE:

  int d [3];

  d[i] → *(&d[0]+i)
- EXAMPLE:

  int a [3] [4];

  a[i][j] → *(&a[0][0]+4*i+j)

Arrays as Function Arguments

```
double sum(int [] , int);
main()
{    int x[9];
     double r;
     r=sum(x,9); //sum(&x[0],9)
}
double sum( int a[], int n)
{    int i;
     double result =0.0;
     for (i=0; i<n; i++)
          result=result+a[i];
     return result;
}
```

```
double sum(int* , int);
main()
{    int x[9];
     double r;
     r=sum(x,9); //sum(&x[0],9)
}
double sum( int *a, int n)
{    int i;
     double result =0.0;
     for (i=0; i<n; i++)
          result=result + *(a+i);
     return result;
}
```

Dynamic Memory Allocation

- calloc : **C**ontiguous memory **ALLOC**ation
- malloc : **M**emory **ALLOC**ation

calloc

- calloc(n, el_size)
  - an array of n elements, each element having el_size bytes

```
void main()
{    int *a;  //will be used as an array
     int  n; // size of array
     ....
     a=calloc(n, sizeof(int)); /* get space for a , and
                                  initialize each bit to zero */
     ....
     free(a); /* each space allocated dynamically should
                 be returned */
}
```

malloc

```
/* malloc does not initialize memory */
void main()
{    int *a;  //will be used as an array
     int  n; // size of array
     printf("give a value for n:");
     scanf("%d",&n);
     a=malloc(n*sizeof(int)); /* get space for a (allocate a)*/
     ....
     free(a);
}
```