1/9/2021

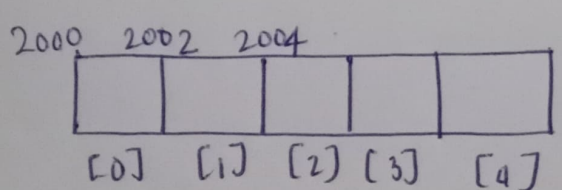
MODULE 1

ARRAY

- data structure ⇒ which (structure) is used for storing data.
- Array ⇒ data structure which is used to store homogenous (same type) types of elements stored in contiguous memory locations.
- Array is a linear data structure
  ⇓
  elements are stored one after another
- Array ⇒ linear data structure used to store homogenous types of elements at continuous memory locations.
- Eqn to find address in Array:

  starting address + index value x size of data type.

  int a[5];



a[0] = 8

SA + (index x size of dt)

= 2000 + (0 * 2)

= 2000

Interview Imp Storage Classes

1) Local Variable

- variable declared inside the function
- default value = garbage value
- int a ; | 100 |

memory location – hardware

a = 100; // 100 is meaningful to this pgm

once pgm execution complete, memory location is free

int b; // b gets memory location same as a, so that value 100 is garbage to b

- access → limited to block.

2) Global Variable

- default value - zero
- access - globally

3) Static Variable

- default value - zero
- access - live in that function (retain old value)

4) register variable

x default value - garbage value

x access - limited to block.

static eg:

```
void displ()
{
  static int s;
  printf("s=%d ", s);
     s++;
  }
int main()
 {
    displ();
    displ();
   displ();
   return 0;
 }
```

output

```
s=0
s=1
s=2
```

Assumption : $n \leq 0$

x responsibility of pgmmer that value of n will not cross the limit.

```
x #include<stdio.h>
  int s;                          // Since a global variable no error of undeclared variable
  void change()
  { int s; // local variable → only locally available
    s=20;
  }
  void overwrite()
  { s=30;
  }
  int main()
  { printf("s=%d",s); //s=0
    change();
    printf("s=%d",s); //s=20
    overwrite();
    printf("s=%d",s); //s=30
    return 0;
  }
```

so o/p

s=0
s=0
s=30

global variable ⇐ pgm

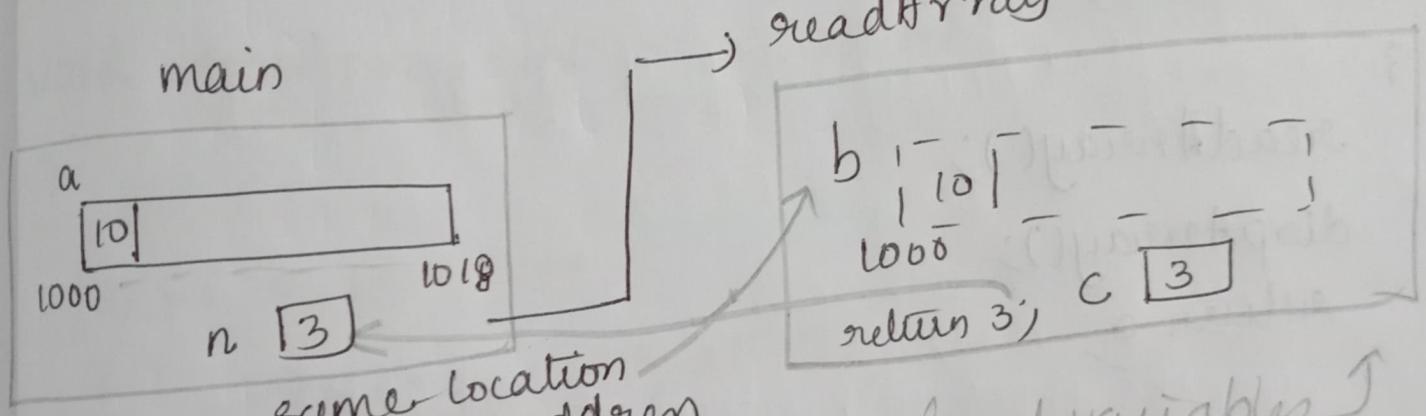
Output

s=0
s=20
s=30

× declaring global variable is not a good standard.

× Linux error : segmentation fault, C does not check limit, of user's responsibility

× n = readArray (a) → call by reference
↓
starting address.



× Using Global Variables

× #include <stdio.h>
int a[10], n; //global variables
void readArray()
{
int i;
printf("Enter no: of elements :");
scanf("%od", &n); //no: of elements
for(i=0; i<n; i++)
{
printf("Enter a[%od] —>", i);
scanf("%od", &a[i]); //reading one by one

```
void dispArray()
{  int i; //local variable
   for(i=0; i<n; i++)
     {
       printf("%d\t", a[i]);
     }
}

int main()
{
  readArray();
  dispArray();
  return 0;
}
```

Output

```
no: of elements: 3
Enter a[0] -> 10
Enter a[i] -> 20
Enter a[2] -> 30
10  20  30
```

x Using Local Variables

```
#include <stdio.h>
int readArray (int b[10]) {
  int i, n;
printf("Enter no of elements: ");
```

```
    scanf("%d", &c);
    for(i=0; i<c; i++)
    {
        printf("Enter b[%d] -> ", i);
        scanf("%d", &b[i]);
    }
    return c;
}
void dispArray(int a[10], int n)
{
    int i;
    for(i=0; i<n; i++)
    {
        printf("%d\t", a[i]);
    }
}
int main()
{
    int a[10], n; //local variable
    n = readArray(a);
    dispArray(a, n);
    return 0;
}
```

a and b have same address

Output

```
Enter the no: of elements : 3
Enter a[0] -> 10
```

Enter a[1] → 20

Enter a[2] → 30

10 20 30

→
```
#include<stdio.h>
int readValue(int a[10], int n)
{
  if(n==9)
  {
   printf("Array is full");
  }
  else
  {
    n=n+1;
   printf("Enter the value at a[%od]:", n);
    scanf("%od", &a[n]);
  }
return n;
}

void dispArray(int a[10], int n)
{
  int i;
  if(n==-1)
  {
   printf("Array is Empty");
  }
  else
```

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

int main()
{
int a[10], n=-1, ch;
n=readArray(a,n);
dispArray(a,n);
return 0;
}

int menu()
{
  int ch;
  printf("\n Insert -1\n Display -2\n Exit -3\n
     Your Choice");
  scanf("%d", &ch);
  return ch;
}
```

```
for(ch=menu();ch!=3;
    ch=menu()){
switch(ch){
case 1: n=readValue(a,n);
        break;
case 2: dispArray(a,n);break;
default: printf("Wrong Choice");
}}
```

10/9/2021

Array

- ☆ linear data structure
- ☆ used for homogenous / same type of elements

★ stored at continuous memory locations.

⇓

this is to ~~store~~ resolve the address

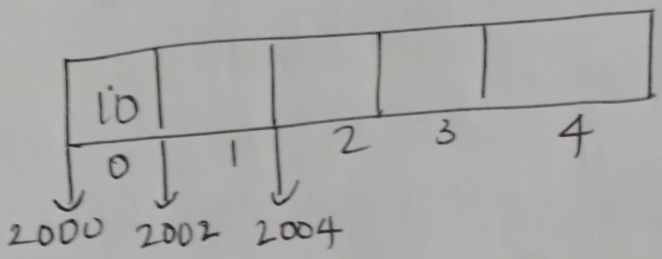
⇓

a[0] = 10; // how to identify that it should be stored at first location

★ ~~Asso~~ Address of i^{th} element:

(starting address) + (i x sizeof(datatype))

★ through starting address we can access other elements.

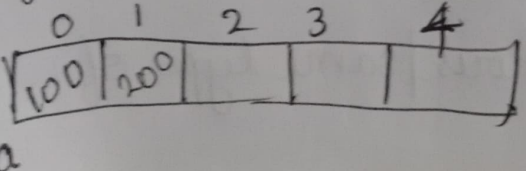


a[0] =
2000 + 0 * 2
= 2000

ways of inserting elements.

* variable called pos tells where the last element is inserted.

* initially value of pos = -1 ⇒ array is empty and first element should be zero.



- insert ⇒ 100
  - → pos = pos+1
  - → pos = -1+1 = 0
  - → a[pos] = 100;
- insert → 200
  - → pos = pos+1 ⇒ 0+1 = 1
  - → a[pos] = 200

* before increasing we have to check pos+1 == size

* S = 5;

```
pos = -1;
insert(e)
{
   if(pos+1 == S)
   {
    announce("Array is full");
   }
   else
   { pos = pos+1;
     a[pos] = e;
   }
}
```

here not ↑ing the value. checking from the current position if the next position is out of bound

Deleting Elements from the Array.

```
deletion()
{ if (pos == -1)
    announce("Array is empty");
  else
```

```
}
    display the value at (pos)
    pos = pos - 1;
}
}
```

pos is decremented, element is there, next insertion will overwrite the element

* position is used to insert & delete element.
while inserting ↑ing position
while deleting ↓ing position.

Program to insert & delete elements from array.

Using Global Variables

```
#include <stdio.h>
#define size 5
int a[size];
int pos = -1;
~~int main~~ void insert(int e)
{
    if(pos+1 == size)
    {
        printf("Array is full");
    }
    else
    {
        pos = pos+1;
        a[pos] = e;
    }
}
```

```
void delete()
{
    if (pos == -1)
    {
        printf("Array is empty");
    }
    else
    {
        printf("Last inserted elements %d", a[pos]);
        pos = pos - 1;
    }
}
int main()
{
    delete(); //empty
    insert(100);
    insert(200);
    insert(300);
    insert(400);
    insert(500);
    insert(600); //full
    delete();
    delete();
    printf("Value of pos : %d", pos);
    return 0;
}
```

Output

Array is Empty
Array is full
Last Inserted element : 500
Last inserted element : 400
Value of pos : 2

can also do with menu

* Using Local Variables also this pgm can be done

~~#include<stdio.h>~~

Structure

* used for creating meaningful datatype using existing datatype.
* struct ⇒ keyword.
* student → entity.
  - ↳ sno → int
  - ↳ sname → char

  } attributes
* date → entity
  - ↳ date → int
  - ↳ month → char
  - ↳ year → int

  } attributes.
* struct emp name of structure.

```
{
    int eno; // 2 bytes
    char ename[20]; // 1×20 = 20 bytes
} float sal; //
```

```
void main()
{
  struct emp e; // memory allocated 26 bytes
}
```
2+20+4 = 26

* struct keyword has no size
* size of structure variable = sum of sizes allocated to individual variables inside the structure

Program using Structure

```
struct emp
{
  int eno;
  char ename[20];
  float esal;
};
int main()
{
  struct emp e;
  printf("Enter no, name and salary");
  scanf("%d %s %f", &e.no, &e.ename,
                    &e.esal);
  printf("No %d Name %s salary %f",
         e.eno, e.ename, e.esal);
  return 0;
}
```

e.no
↳ dot operator
member operator

Output

Enter no name and salary 1.

abcd

20000

No 1 Name abcd Salary 20000.000000

★ Inside of always writing struct emp, typedef keyword can be used.

★ typedef struct emp employee;

employee e;

↳ used to redefine ~~keywords~~

★ typedef int num;

num eno;

★ emp e[10];

&e[i].eno, &e[i].ename, &e[i]-esal

★ Total size of array of structure

= 26 × 10 = 260 bytes

Try to Do: Read & display polynomial using ~~stru~~