Prep Smart. Score Better.

C Language- Pointers, Structure, and Command Line Arguments

ABOUT ME : NAVNEET GUPTA

- 8 years teaching experience.
- AIR 92 in GATE 2008
- Qualified UGC-NET 2012, Raj.-SET 2012, CSIR-Recruitment-Exam in 2011
- Achieved 3rd Rank in NPTEL-DBMS Course
- Achieved Silver Medal in CSIR on ERP Project in 2013
- Area of Expertise : DBMS, Programming, Algorithms, Discrete Maths, Computer Networks, Operating system

gradeup
Prep Smart. Score Better.

Pointers In 'C' Language

A pointer is a special variable which is different from traditional variables in the sense it stores the value i.e the address of another variable.

Syntax: How to declare a pointer

datatype *pointer-name ;

eg: int *ptr; // pointer variable declaration

ptr = &a;

int a=5;

gradeup
Prep Smart. Score Better.

1)
```
main()
{
    int a, *p;
    *p=2000;
    pf("%u", &a);
}
```

o/p :

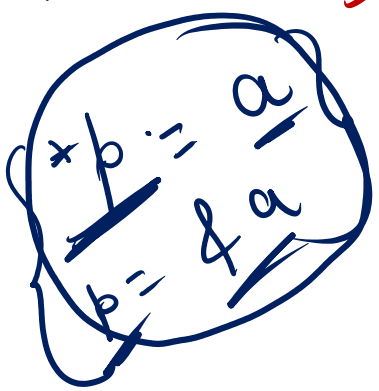
o/p
compile time Error

2)
```
main()
{
    int a, *p;
    *p = &(2000);
    pf("%u", p);
}
```

o/p :

Error

gradeup
Prep Smart. Score Better.



3) main() %u = unsigned integer

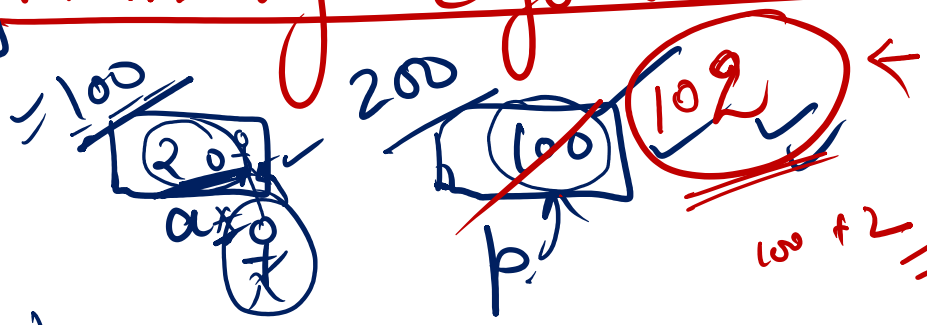
{

int a=20, *p = &a;

1. pf("%u", &a);
2. pf("%u", p);
3. pf("%u", &p);
4. pf("%u", *p);

int a=5;
int *p;
p = &a;
(or)
int *p = &a;
declaration

Memory Layout



1) 100
2) 100
3) 200
4) 20 ✓

100 + 1*2

5. p++; ⇒ p = p+1 ⇒ 102
6. pf("%u", p); ⇒ 102
7. ++*p; → 0
8. pf("%u", *p); → garbage

gradeup
Prep Smart. Score Better.

NOTE:

(i) *p++ ;

R-L

a) use / print / assign the value of *p

b) It increments the value of p

i.e ⇒ p = p+1 → data type

(ii) ++*p ;

(a) it will increment the value pointed by p first i.e. *p = *p + 1;

(b) it uses / print / assign value of *p.

gradeup
Prep Smart. Score Better.

p*

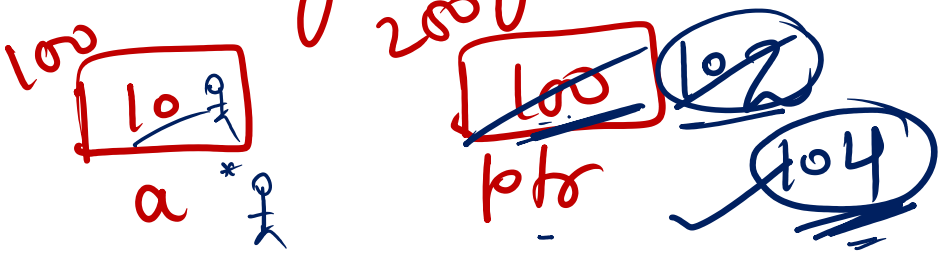
(iii) *++p;

(a) p = p+1

(b) use / assign / print the value of *p;

```
main()
{
    int a=10, *ptr=&a;
1   pf("%u", *ptr);
2   pf("%u", *ptr++);
3   pf("%u", *++ptr);
4   pf("%u", ++*ptr);
}
```

Memory layout

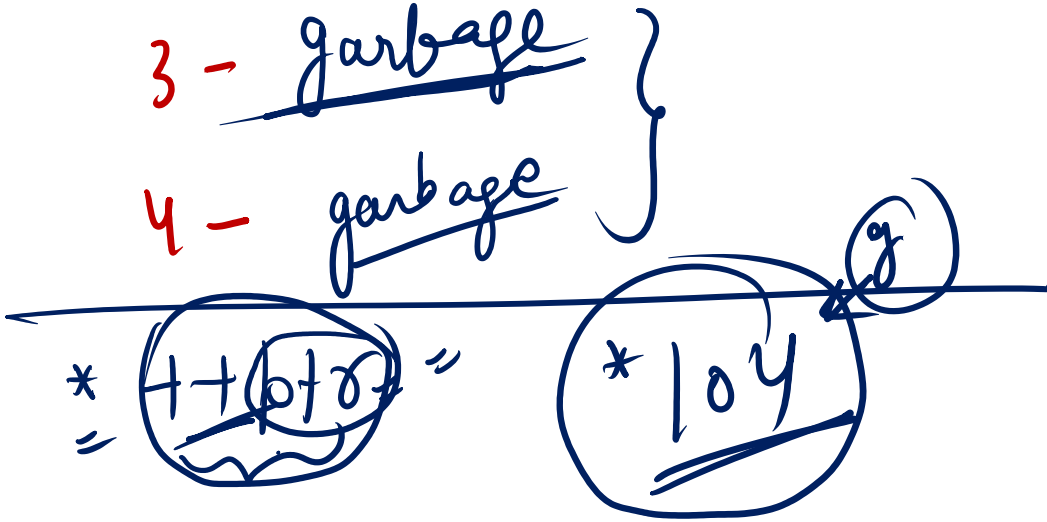


1 - 10

2 - 10

3 - garbage

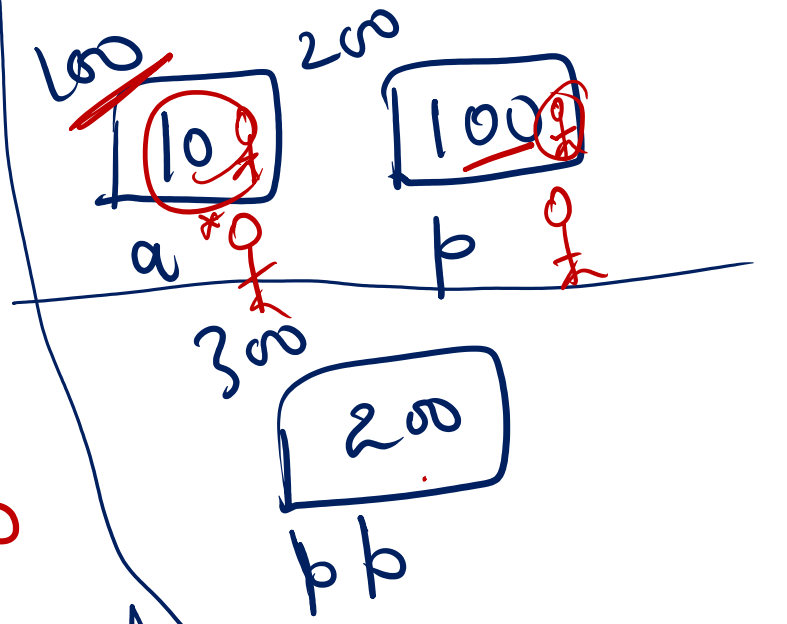
4 - garbage

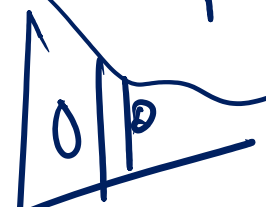


pointer to pointer (double pointer)

```
int a=10;
int *p;
int **pp;
p= &a;
pp= &p;
```

1. pf(p) 100
2. pf(*p) 10
3. pf(pp) 200
4. pf(*pp) 100
5. pf(**pp) 10



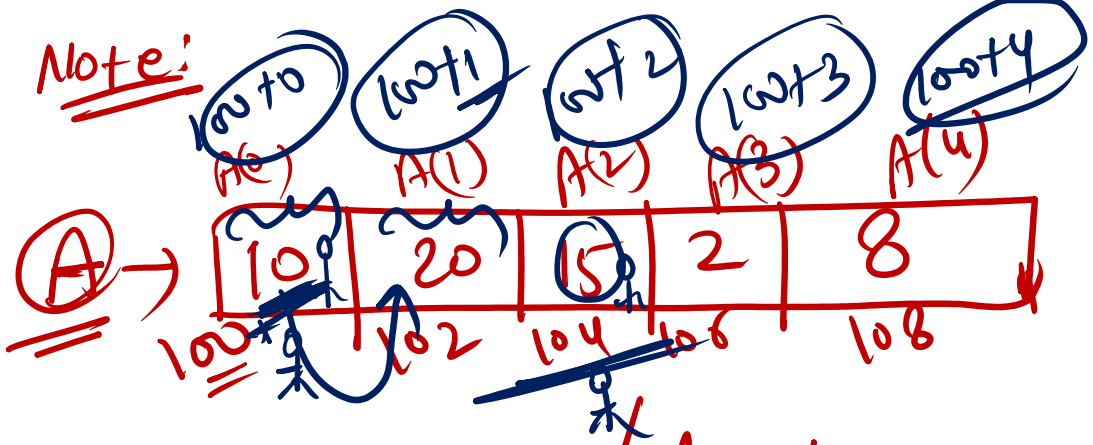


gradeup
Prep Smart. Score Better.

pointer and 1-D Array

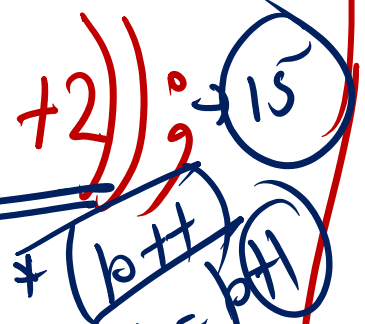
int A[5]; ≡ int *p; ≡ p = A;

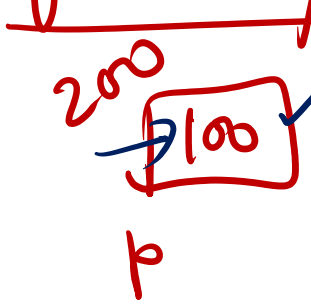
Note:



Writing array name / function name / structure variable name, it means base address of array / function / structure respectively.

1. pf("%u", *p); → 10
2. pf("%u", p); → 100
3. pf("%u", *(p+2)); → 15





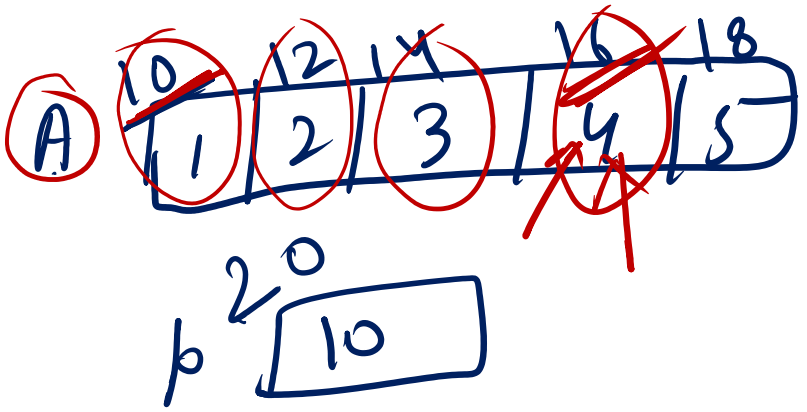
*(p+2)

100+2*2 → *(104)

← bcoz of datatype of pointer (int)

gradeup
Prep Smart. Score Better.

```
main()
{
    int A[5] = {1,2,3,4,5}
    int *p = A;
1.  pf("%u", A);   → 10
2.  pf("%u", A+3); → 16   10+3*2
                          10+6 = 16
3.  pf("%u", *(A+3)); → 4
}
```



gradeup
Prep Smart. Score Better.

Structure in C

*) It is used to define a new data type using primitive data type.

eg:

```
struct emp
{
    int salary;
    int age;
    char name[20];
};
```

member of structure

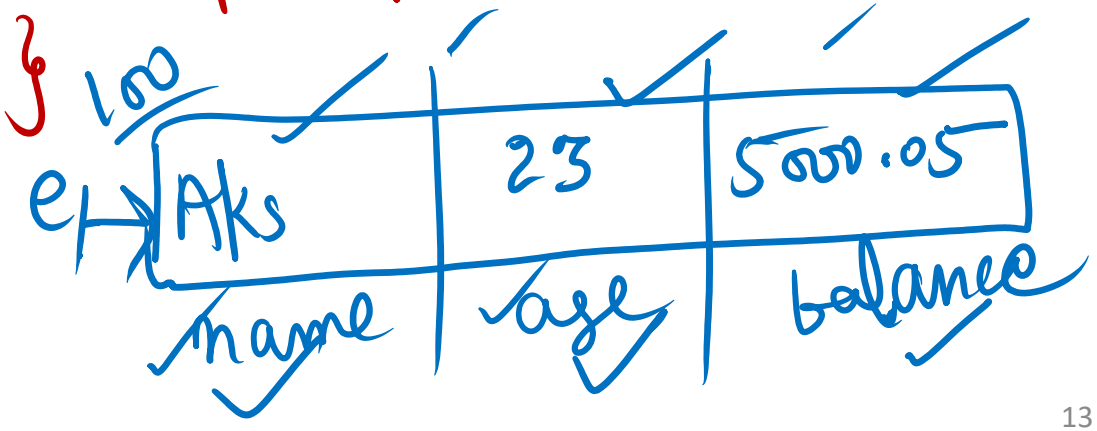
To access the member of structure we use . (dot) operator.

gradeup
Prep Smart. Score Better.

```
struct emp
{
    char name[20];
    unsigned int age;
    float balance;
};
```

public

```
main()
{
    struct emp e1;
    e1.name = "Akshaya";
    e1.age = 23;
    e1.balance = 5000.05;
    {
        pf("%s", e1.name);
        pf("%u", e1.age);
        pf("%f", e1.balance);
    }
}
```



typedef

typedef struct

{

int age;

char name[20];

} Customer;

eg: typedef int char;

⇒ char a = 9;

int

main()

{

Customer C1;

C1.age = 24;

C1.name = "Ram";

}

2

4