Structure and Pointers

- → Structure
- → Pointer
- → Methods of memory allocation
- → Operations on pointers
- → Pointer and Array
- → Pointer and String
- → Pointer and Structure

Structure

- Array
  - Derived data type
  - Used to store group of similar type of data

A	10	5	4	0
	A[0]	A[1]	A[2]	A[3]

Roll	Name	DOB	Addre
int	char	int	char

- Structure
  - Derived data type
  - Used to store group of different types of data
  - Def : used to represent a collection of logically related data items

Roll	Name	DOB	Addre
int	char	int	char

Structure Definition

- Keyword : Struct
- Structure members variable : variable inside structure definition

```
Struct strcture_name
      {
            datatype variable;
            datatype variable;
            datatype variable;
      };
```

```
struct strdate
      {
        int roll;
        char name[10];
        int dob;
      };
```

Accessing elements of a Structure

- Accessing using dot / period (.) operator

(Objuct)(operator)(member variable)

S1.roll

S1.name

```
struct student
    {
        int roll;
        char name[10];
        int dob;
    }s1,s2,s3;
```

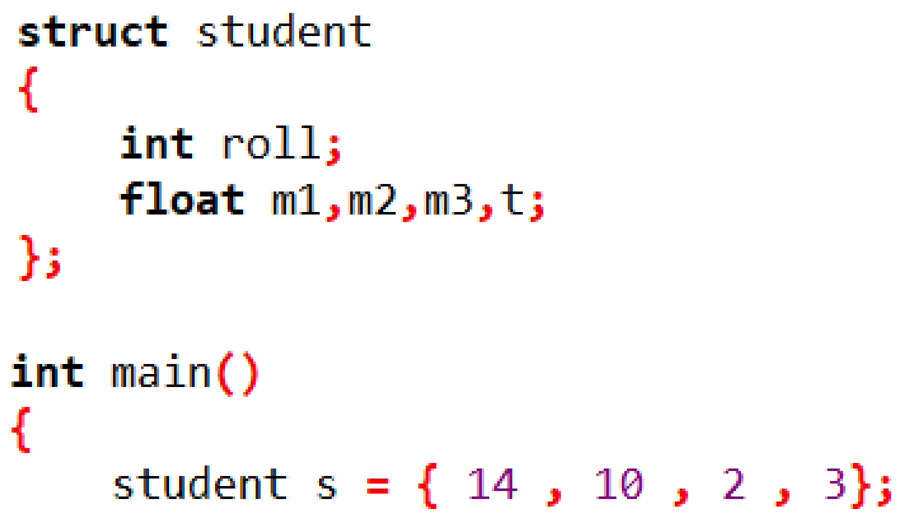
Example 1

- Write a program to add 3 mark of a student

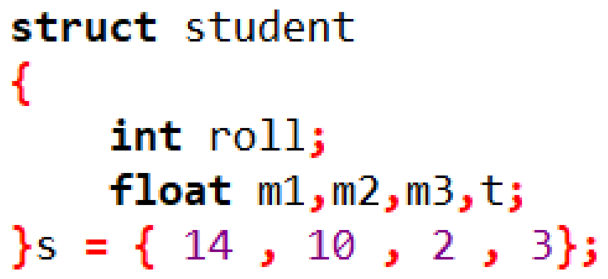
- Declaration : introduce to compiler
- Definition : make a meaning
- Accessing : use

```
struct student
{
	int roll;
	float m1,m2,m3,t;
};


int main()
{
	student s;
	cout<<"enter 3 marks";
	cin>>s.m1;
	cin>>s.m2;
	cin>>s.m3;
	s.t = s.m1 + s.m2 + s.m3;
	cout<<"total :" <<s.t;
}
```

- Variable initialisation
  - During the declaration of variable , we can set initial value
  - Int x = 10;
  - Int arr[4] = [10, 12, 30, 8]



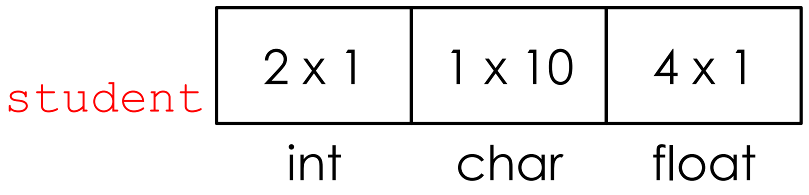
```
struct student
{
    int roll;
    float m1,m2,m3,t;
};

int main()
{
    student s = { 14 , 10 , 2 , 3};
```

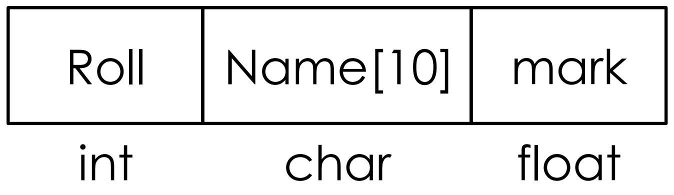


```
struct student
{
    int roll;
    float m1,m2,m3,t;
}s = { 14 , 10 , 2 , 3};
```

Memory allocation

- Total of individual datatypes used in the structure





```
struct student
    {
        int roll;
        char name[10];
        int dob;
    };
```

Example 2

- Write a program to store Students information
- Structure array

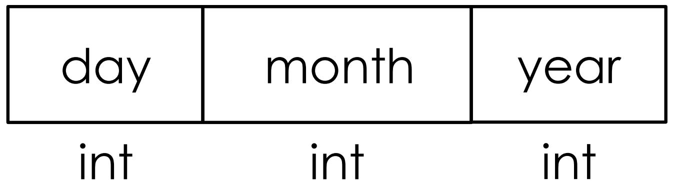
```cpp
#include <iostream>
using namespace std;

struct student
{
    char name[50];
    int roll;
    float mark;
};
```

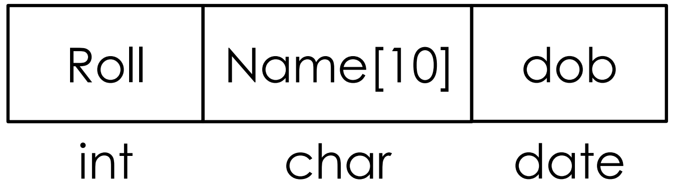
```cpp
int main()
{
    student s[5];
    cout<<"Enter information of student";
    for(int i=0;i<5;i++)
        {
            s[i].roll = i+1;
            cout<<"For "<<s[i].roll<<" : ";
            cout<<"Enter Name";
            cin>>s[i].name;
            cout<<"Enter Mark";
            cin>>s[i].mark;

        }

    cout<<"Students Information\n";
    cout<<"-----------------------------\n";
    for(int i=0;i<5;i++)
        {
            cout<<s[i].roll<<" : "; cout<<s[i].name;
            cout<<" : "; cout<<s[i].mark; cout<<"\n";

        }

}
```

Nested structure [def 1]

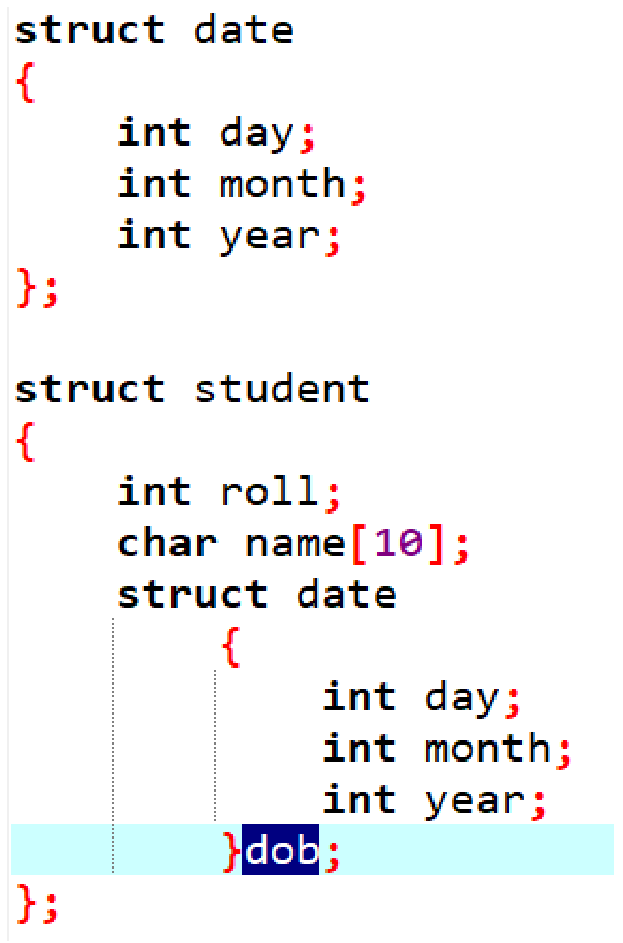
- A structure declare with the another structure datatype

```
struct date
{
    int day;
    int month;
    int year;
};

struct student
{
    int roll;
    char name[10];
    date dob;
};
```

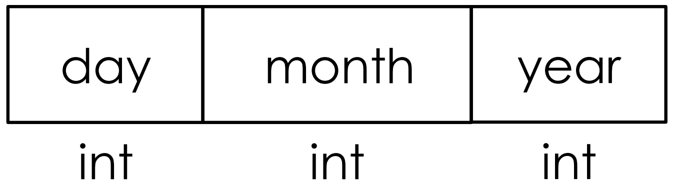


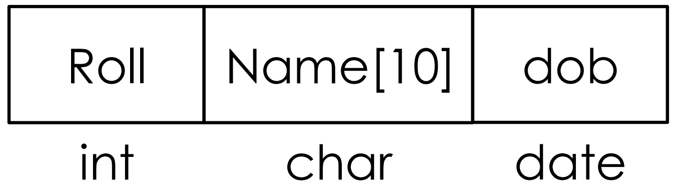


- Nested structure [def 2]



```
struct date
{
    int day;
    int month;
    int year;
};

struct student
{
    int roll;
    char name[10];
    struct date
        {
            int day;
            int month;
            int year;
        }dob;
};
```





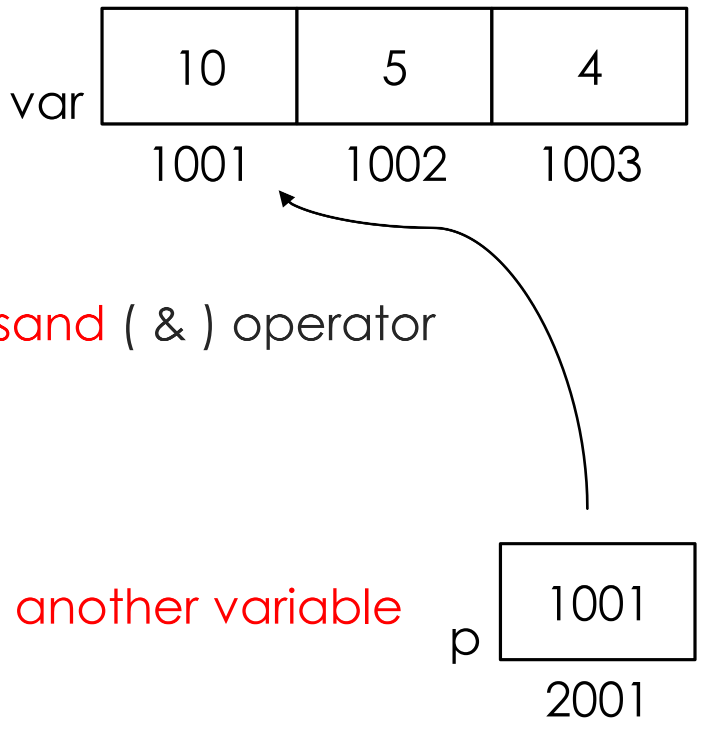
Array vs. Structure

- Derived data type
- Collection of same type of data
- Elements are accessed by using an integer index
- multidimensional array : when an array element contain another array

- User defend data type
- Collection of different type of data
- Elements are accessed by using dot(.) operator
- Nested structure : when a structure element contain another structure

Pointer

- Variable and address
  - Every variable is a memory location
  - Every memory location has unique address
  - The address can be accessed using ampersand (&) operator
- Pointer
  - Is a variable used to store the address of an another variable
  - Derived data type
  - Def 1 : Pointer is used to point the address of a memory location
  - Def 2 : Pointer is used to hold the address of a memory



- Declaration syntax

```
datatype *pointer_variable
```

- Pointer declaration

```
int *ip      // integer pointer
char *ch     // character pointer
```

- Pointer declaration and initialization

```
int *ptr = &data;
```

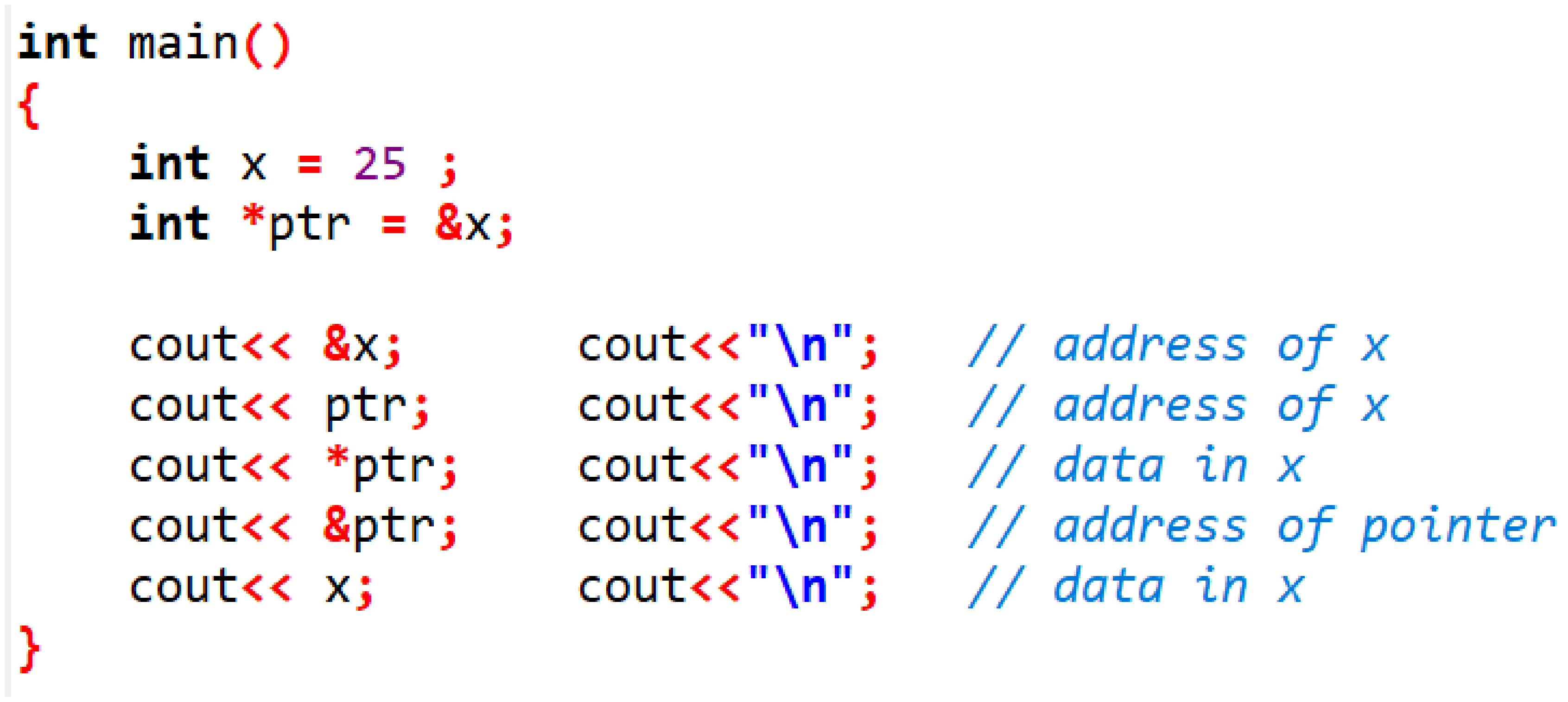
& - Address of operator
* - Dereference operators

- & Operator
- Get the address of a variable

```
int main()
{
    int data = 25;

    cout<< data ;
    cout<<"\n";
    cout<< &data ;
}
```

- * Operator
- Store the address of another variable
- Make a variable into a pointer

```
int main()
{
    int data = 25;

    int *p ;
    int *ptr = &data
}
```



```
int main()
{
    int x = 25 ;
    int *ptr = &x;

    cout<< &x;       cout<<"\n";    // address of x
    cout<< ptr;      cout<<"\n";    // address of x
    cout<< *ptr;     cout<<"\n";    // data in x
    cout<< &ptr;     cout<<"\n";    // address of pointer
    cout<< x;        cout<<"\n";    // data in x
}
```

- Write a program to add two numbers using pointer

```
int x, y;
int *p1, *p2;
p1 = &x;
p2 = &y;

cout<<"Enter 1st number :";
cin>>*p1;
cout<<"Enter 2st number :";
cin>>*p2;

cout<<"\nsum = " << *p1 + *p2;
```

- Write a program to swap two numbers using pointer

```
int x, y, temp;
int *p1, *p2;
p1 = &x;
p2 = &y;

cout<<"Enter x :";
cin>>*p1;
cout<<"Enter y :";
cin>>*p2;
```

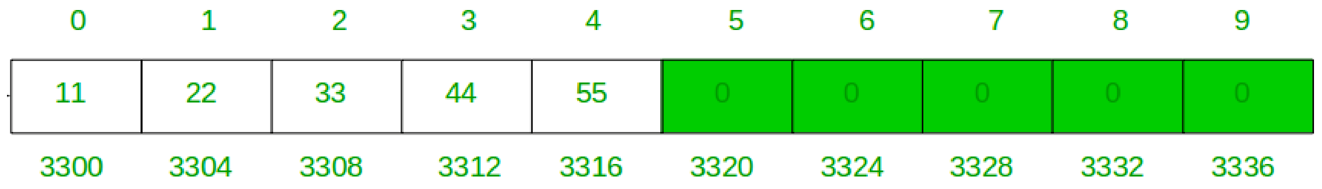
```
temp = *p1;
*p1 = *p2;
*p2 = temp;
```

Memory Allocation

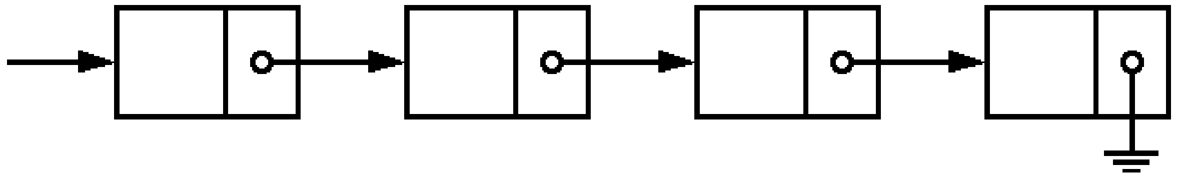
- → Static memory allocation
- → Dynamic memory allocation

Static memory allocation

- Variable memory allocation during compilation time is known as static memory allocation
- It is fixed memory : once memory allocated ,it is fixed
- Cannot expanded or reduced
- Ex : Arrays
- Int data[10] → 2 byte x 10 = 20 bytes



- Dynamic memory allocation
  - Variable memory allocation during execution or run time is known as dynamic memory allocation
  - It is not fixed memory : once memory allocated ,it is not fixed
  - It can be expanded or reduced
  - It is done using NEW and Delete operators
  - Ex : Pinter



NEW and DELETE Operators

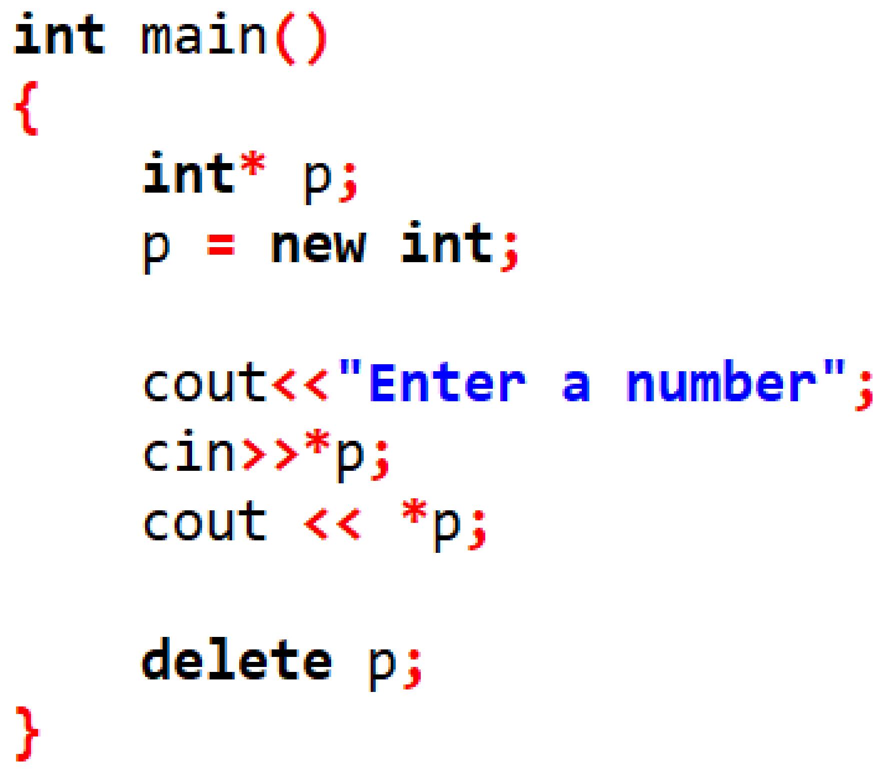
- New operator is used to allocate memory during runtime
- Unary operator

```
int* p;
p = new int;
```

- Delete operator is used to delete or de-allocate the memory during runtime
- Unary operator

```
delete p;
```

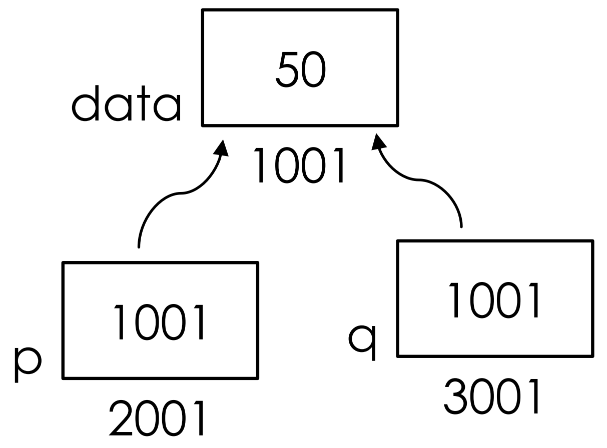
Example new & delete



```
int main()
{
    int* p;
    p = new int;

    cout<<"Enter a number";
    cin>>*p;
    cout << *p;

    delete p;
}
```

Memory leak

- Orphaned memory :
  - The memory allocated using the new operator forgot to de-allocate using delete operator, the memory is kept left unused.
  - Such memory blocks are called Orphaned memory
  - Each execution, the amount of orphaned blocks are increase
  - This situation is called memory leak
- Reason for memory leak:
  - Forget to delete allocated memory
  - Multiple allocation to a pointer variable



Operations on pointers

- Arithmetic operation
  - Integer addition
  - Integer subtraction

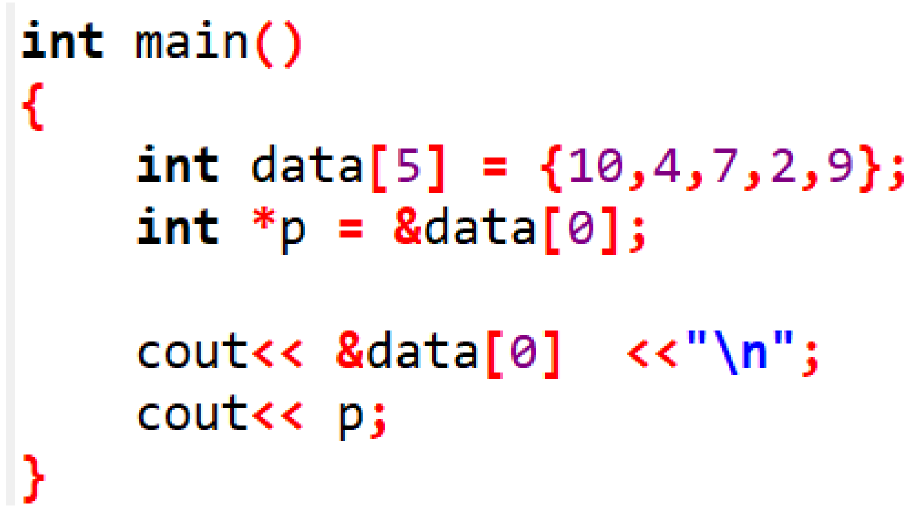
```
ptr++
ptr--

ptr = ptr + 1
ptr = ptr - 1

ptr += 1
ptr -= 1
```

- Relational operation
  - Equal to
  - Not equal to

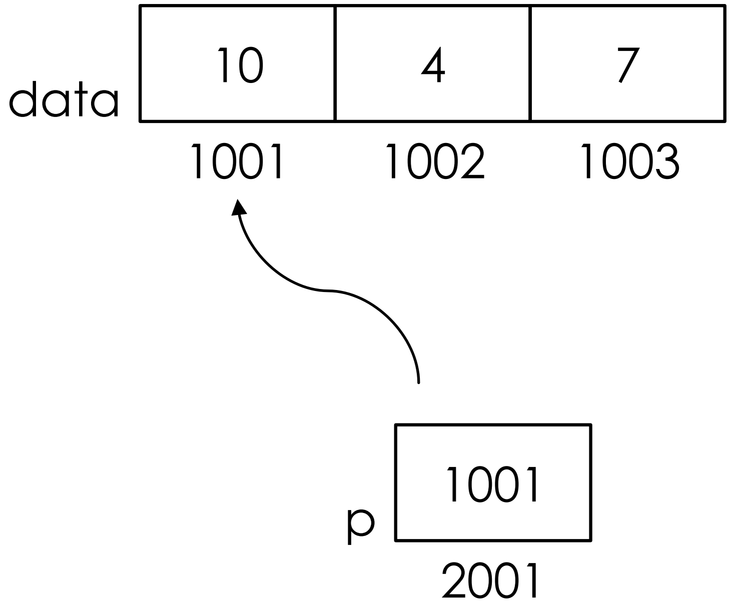
```
ptr1 == ptr2

ptr1 != ptr2
```

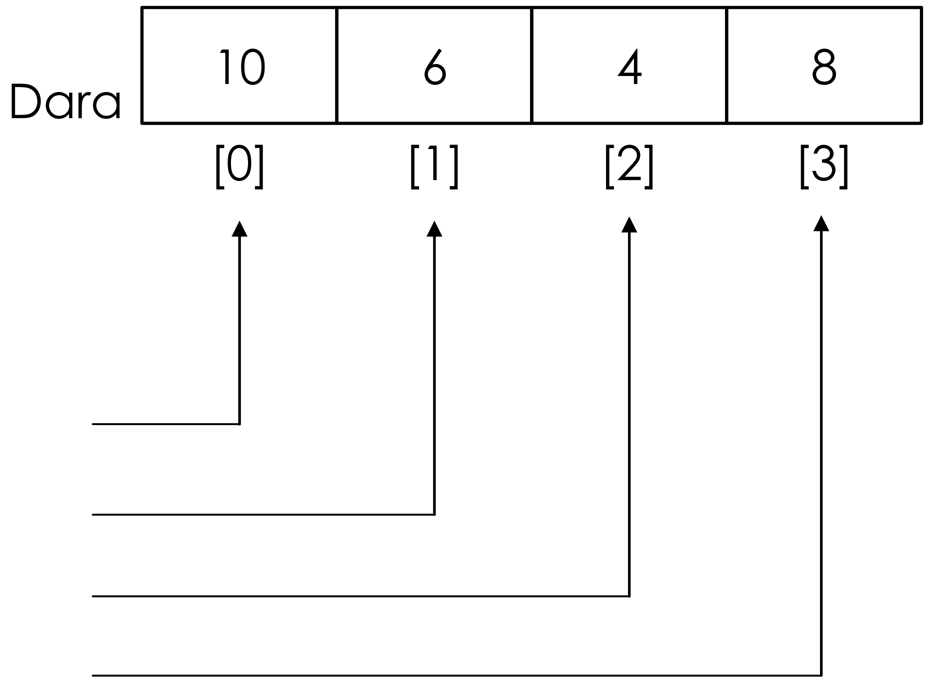
Pointer and Array

- Pointer array
  - Array name is treated as pointer
  - Pointer points the First element of the array

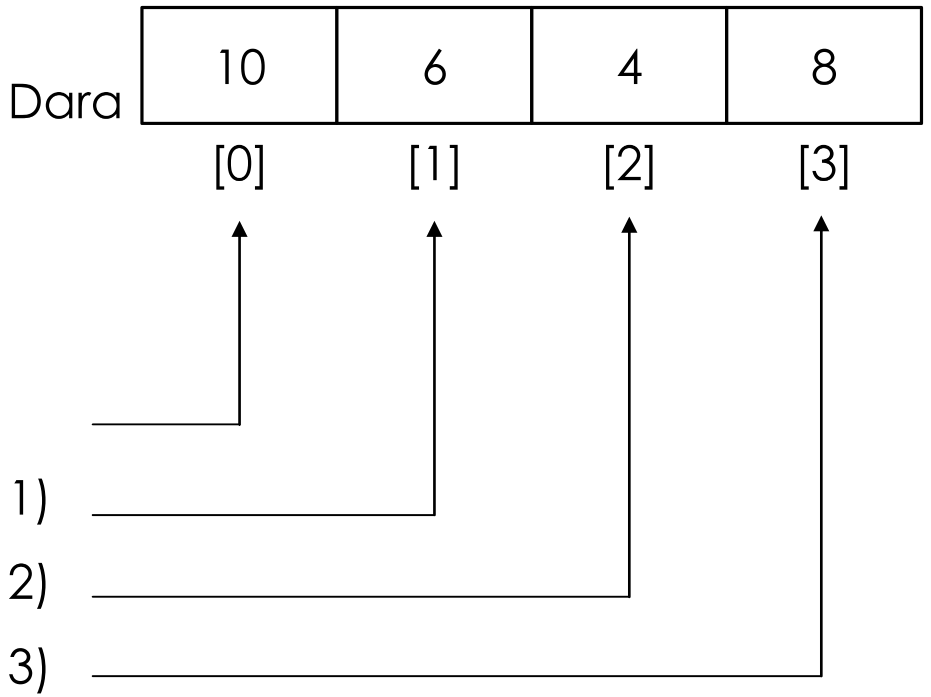


```
int main()
{
    int data[5] = {10,4,7,2,9};
    int *p = &data[0];

    cout<< &data[0]  <<"\n";
    cout<< p;
}
```



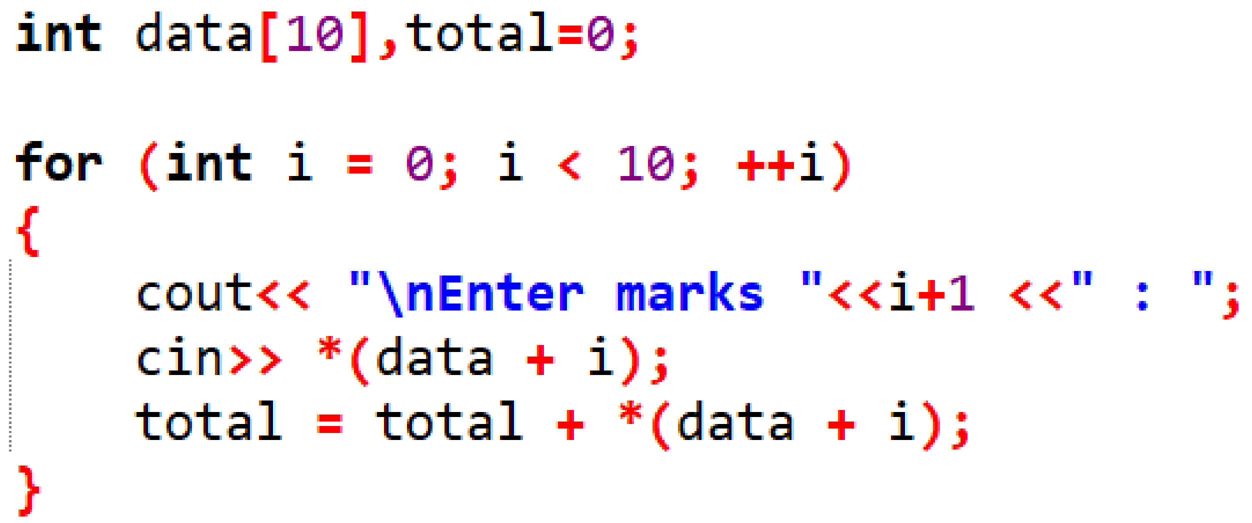


- Data[0]
- Data[1]
- Data[2]
- Data[3]



- *(Data)
- *(Data + 1)
- *(Data + 2)
- *(Data + 3)

- Write a program to add 10 students make using pointer array



```
int data[10],total=0;

for (int i = 0; i < 10; ++i)
{
    cout<< "\nEnter marks "<<i+1 <<" : ";
    cin>> *(data + i);
    total = total + *(data + i);
}
```

Pointer and String

- Pointer String
  - String is an array of character
  - Array name considered as string variable
  - No need to use & to assign pointer variable

```
char name[] = "appu";

char *p1 = name;

~~char *p2 = &name;~~
```

- Advantages of character pointer
  - No need to specify the size
  - Assignment operator (=) is used to copy a string (No need strcpy)

```
char name[] = "appu";

char *p1 = name;
```

```
char name[];
char *ptr ;

strcpy(name,"appu")        ptr = "appu"
```

- Array of strings
  - Declare a character array as pointer

```
char *name[4] = {"appu","raju","vishnu","vivek"};
```

String Array V/s pointer string array

- Normal string array declaration and initialization

```
char name[10][4] = {"appu","raju","vishnu","vivek"};
```

- pointer string array declaration and initialization

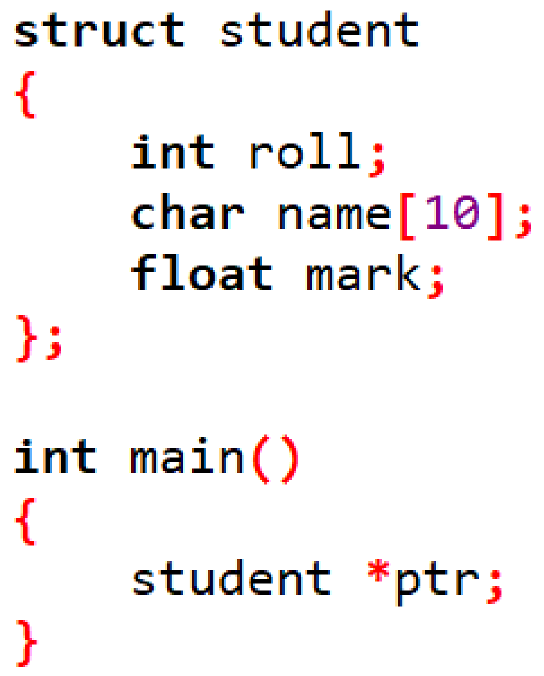
```
char *name[4] = {"appu","raju","vishnu","vivek"};
```

Example

```
int main()
{
    char *name[4] = {"appu","raju","vishnu","vivek"};

    for (int i=0; i<4; i++)
    {
        cout<<"\n" << name[i];
    }
}
```

Pointer and Structure

- Structure pointer
  - A pointer declare with a structure name is known as structure pointer



```
struct student
{
    int roll;
    char name[10];
    float mark;
};

int main()
{
    student *ptr;
}
```

- Using Struct variable and pointer

```
struct student
{
     int roll;
     char name[10];
     float mark;
};
```

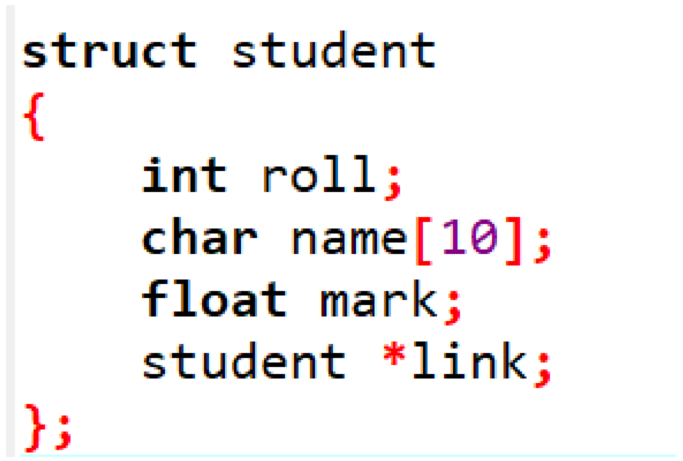
```
int main()
{
     student *ptr , d;
     ptr = &d;

     cout<<"\nenter roll : ";
     cin>>(*ptr).roll;
     cout<<"\nenter name : ";
     cin>>(*ptr).name;
     cout<<"\nenter mark : ";
     cin>>(*ptr).mark;
```

- Using dynamic memory

```
struct student
{
    int roll;
    char name[10];
    float mark;
};
```

```
int main()
{
    student *ptr;
    ptr = new student;

    cout<<"\nenter roll : ";
    cin>>(*ptr).roll;
    cout<<"\nenter name : ";
    cin>>(*ptr).name;
    cout<<"\nenter mark : ";
    cin>>(*ptr).mark;
```

- Self Referential Structure
  - Is a structure which one of the element is a pointer to the same structure



```
struct student
{
    int roll;
    char name[10];
    float mark;
    student *link;
};
```

Reference operator V/s Dot Operator

- Reference operator
  - Used to access data
  - Arrow like symbol (->)
  - syntax

```
ptr -> member
```

- Dot operator
  - Used to access address
  - dot symbol (.)
  - syntax

```
*ptr . member
```

```
struct student
{
    int roll;
    char name[10];
    float mark;
    student *link;
};

int main()
{
    student *ptr;
    ptr = new student;

    cout<<"\nenter roll : ";
    cin>> ptr -> roll;
    cout<<"\nenter name : ";
    cin>> ptr -> name;
    cout<<"\nenter mark : ";
    cin>> ptr -> mark;
```

```
struct student
{
    int roll;
    char name[10];
    float mark;
    student *link;
};

int main()
{
    student *ptr;
    ptr = new student;

    cout<<"\nenter roll : ";
    cin>> (*ptr).roll;
    cout<<"\nenter name : ";
    cin>> (*ptr).name;
    cout<<"\nenter mark : ";
    cin>> (*ptr).mark;
```