Unit-4

19:27 PM
Mon-22-Jan
2024
Gitlab

Pointers & User-Defined Data Types

Pointers, Dereferencing and address operators, pointers and address arithmetic, array manipulation using pointers, User-defined data types - structures and Unions.

Pointers :-

A pointer is a derived data type in C. It is built from one of the fundamental data types available in C.

Pointers contain memory addresses as their values. These memory addresses are the locations in the computer memory where program instructions and data are stored.

Pointers are used to access and manipulate data stored in the memory.

Pointers offer number of benefits to the programmer. They include:

→ Pointers are more efficient in handling arrays and data tables.

→ Pointers can be used to return multiple values from a function via function arguments.

→ Pointers permit references to functions and thereby facilitating passing of functions as arguments to other functions.

→ The use of pointer arrays to character strings results in saving of data storage space in memory.

→ Pointers allow C to support dynamic memory management.

→ Pointers provide an efficient tool for manipulating dynamic data structures such as structures, linked lists, queues, stacks and trees.

→ Pointers reduce length and complexity of programs.

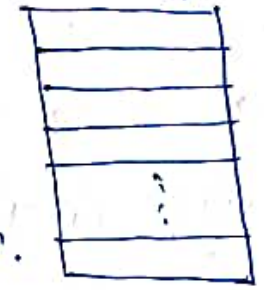
→ They increase the execution speed, reducing the program execution time.

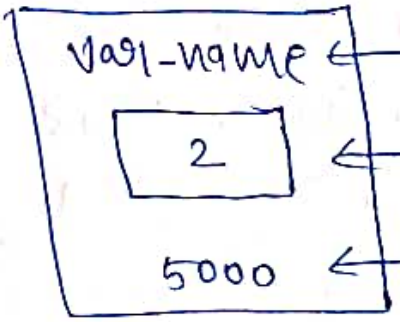
Understanding Pointers:

The computer memory is a sequential collection of storage cells. Each cell commonly known as a byte, has a number called address associated with it.

The addresses are numbered consecutively starting from zero. The last address depends on the memory size.

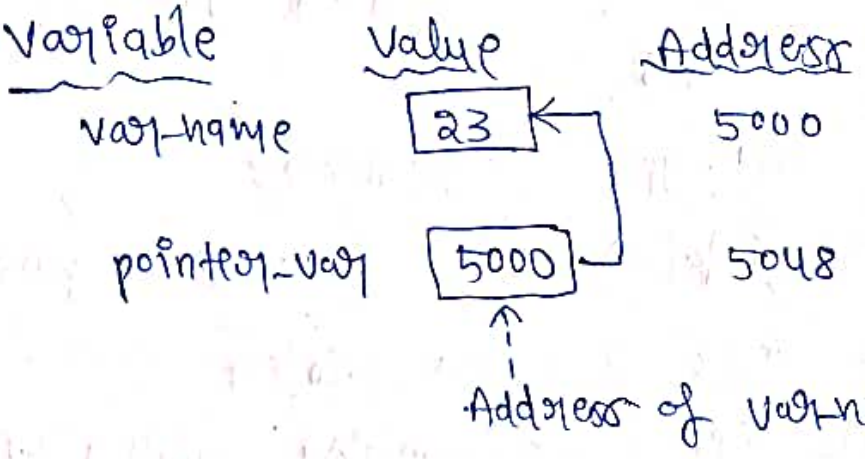
Whenever we declare a variable, the system allocates memory to hold the value of the variable. every byte has a unique address.



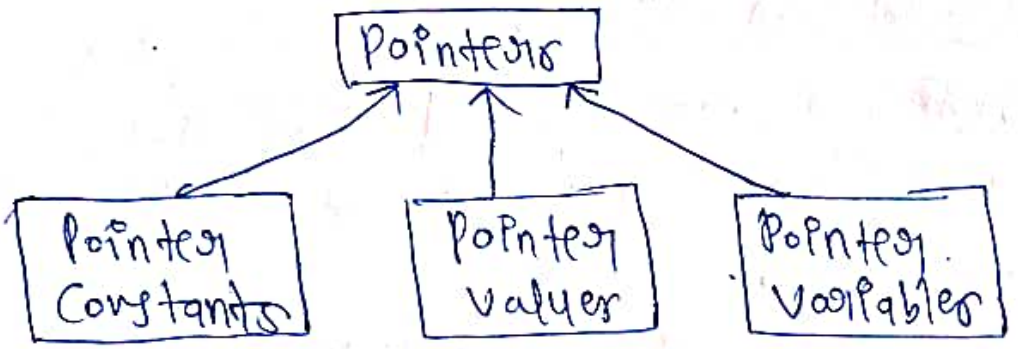


64K memory.

A pointer is a variable, its value is also stored in the memory in another location.



Concepts of Pointers:



Many addresses within a computer are referred to as pointer constants.

We can not save the value of a memory address. We can only obtain the value through the variable stored there using the address operator (&). The value obtained is known as pointer value.

Once we have a pointer value, it can be stored in to another variable. The variable that contains a pointer value is called a pointer variable.

Declaring Pointer Variables:

Pointer variables contain addresses that belong to a separate data type.

The declaration of a pointer variable takes the following form:

```
data-type  *pt_name;
```

The asterisk(*) tells, it is a pointer variable.

```
eg: int *p;   // integer pointer
    float *f; /* floating pointer */
```

(Comments)

Pointer declaration Styles:

Pointer variables are declared similarly as normal variables except for the addition of the unary * operator.

```
int* p;
int *p;
```

The asterisk(*) symbol can appear anywhere b/w the type name and the pointer variable name.

Initialization:

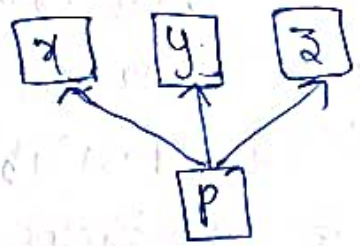
The process of assigning the address of a variable to a pointer variable is known as initialization.

```
int *p;          // declaration
p = &addr;       // initialization.

int *p = &addr;

int x, *p = &x;

int *p = NULL;

int *p = 0;
```

Pointer Flexibility:

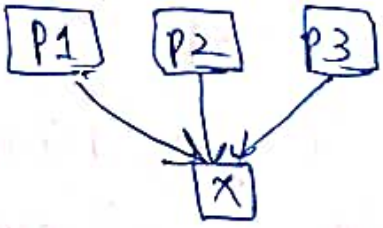
We can make the same pointer to point to different data variables in different statements.

eg:
```
int x,y,z,*p;
p = &x;
p = &y;
p = &z;
```



We can also use different pointers to point to the same data variable.

eg:
```
int x;
int *p1 = &x;
int *p2 = &x;
int *p3 = &x;
```



Accessing a variable through its pointer:

To access the value of the variable using the pointer

by using another unary operator asterisk (*), usually known as the Indirection operator.
Another name for the Indirection operator is the dereferencing operator.

```
int num, *p, min;
num = 17;
p = &num;
min = *p;
```

The * can be remembered as "value at address".

Rules of Pointer Operations:

The following rules apply when performing operations on pointer variable.

(i) A pointer variable can be assigned the address of another variable.
(ii) A pointer variable can be assigned the values of another pointer variable.
(iii) A pointer variable can be initialized with NULL (or zero value.
(iv) A pointer variable can be pre-fixed (or post-fixed with increment (or decrement operators.
(v) An integer value may be added (or subtracted from a pointer variable.
(vi) A pointer variable can not be multiplied by constant.
(vii) Two pointer variables can not be added.

Troubles with Pointers:

Some pointer errors that are more commonly committed by the programmers:

→ Assigning values to uninitiated pointers.

```
int *p, m = 100;
    *p = m;
```

→ Assigning value to a pointer variable.

```
int *p, m = 100;
    p = m;
```

→ No dereferencing a pointer when required.

```
int *p, x = 100;
    p = &x;
    printf("%d", p);
```

→ Assigning the address of an uninitialized variable.

```
int m, *p;
    p = &m;
```

→ Comparing pointers that point to different objects.

```
char name1[10], name2[10];
char *p1 = name1;
char *p2 = name2;
if (p1 > p2) ...
```

Dereferencing and address Operators:-

The '*' operator in 'C' is called the dereferencing operator.

* = Asterisk.

It is used to access the value stored at the address pointed to by a pointer.

If 'ptr' is a pointer, then '*ptr' refers to the value stored at the memory location pointed to by 'ptr'.

eg: int x=10;
int *ptr=&x; // ptr holds the add$ of x.
int value=*ptr; // value contains, The value stored at the add$ pointed by ptr. (which is 10).

We can also use the dereferencing operator to modify the value at a specific memory location.

eg: *ptr=20;

Address of Operator: & = Ampersand.

The & operator in C is called the address of operator.

It is used to get the memory address of a variable.

If 'x' is a variable then &x gives the memory address where the value of 'x' is stored.

eg: int x=10;
int *ptr=&x; // ptr holds the add$ of x.

Address of operator is often used when passing variables by reference to functions.

```
void modifyValue (int *ptr) {
    *ptr = 20;
}
int main() {
    int x = 10;
    modifyValue (&x);
    // Now, the value of x is 20
    return 0;
}
```

These operators provide a way to manipulate data indirectly and are fundamental for tasks like dynamic memory allocation and passing variables by reference.

Pointer and Address Arithmetic:-

Pointer arithmetic is a powerful feature in C that allows you to manipulate pointers by performing arithmetic operations on them.

Increment and Decrement:
When you increment & decrement a pointer, it moves to the next & previous memory location based on the size of the data type it points to.

eg: `int arr[] = {10, 20, 30, 40};`
`int *ptr = arr; // ptr hold the first element of array.`
`ptr++; // moves to the next int-sized memory location`

Arithmetic with Indexing:
We can combine pointer arithmetic with array Indexing.

eg: int arr[] = {10,20,30,40};
int *ptr = arr;
int thirdElement = *(ptr+2); // access the 3rd element

Pointer Subtraction:
Subtracting two pointers gives the number of elements b/w them.

eg: int arr[] = {10,20,30,40};
int *ptr1 = arr;
int *ptr2 = &arr[3];
int numElements = ptr2 - ptr1; // will be 3

Address Arithmetic:
Address can be calculated using pointer arithmetic.

eg: int arr[] = {10,20,30,40};
int *ptr = arr;
printf("Address of arr[2]: %p\n",
(void *) (ptr+2)); // points the add of arr[2]

Size of data type:
pointer arithmetic is influenced by the size of the data type it points to.

eg: int *intPtr;
char *charPtr;
intPtr++; // Moves by sizeof(int).
charPtr++;

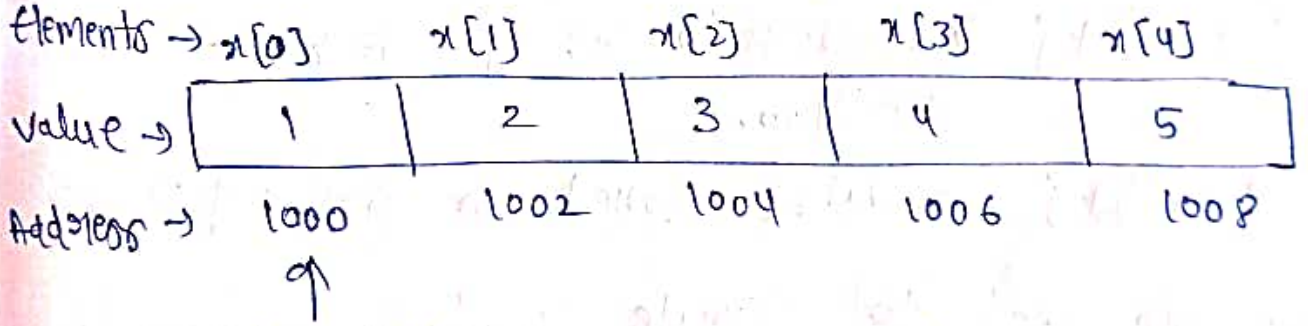
Pointers and address arithmetic are especially useful when working with arrays, dynamic memory allocation, and data structures.

Array manipulation using Pointers:-

When an array is declared, the compiler allocates a base address and sufficient amount of storage to contain all the elements of the array in contiguous memory locations.

The base address is the location of the first element (index-0) of the array. The compiler also defines the array name as a constant pointer to the first element.

```
int x[5] = {1,2,3,4,5};
```



eg:- p = &x[0] = 1000

p = x

p = &x[0] = 1000

p+1 = &x[1] = 1002

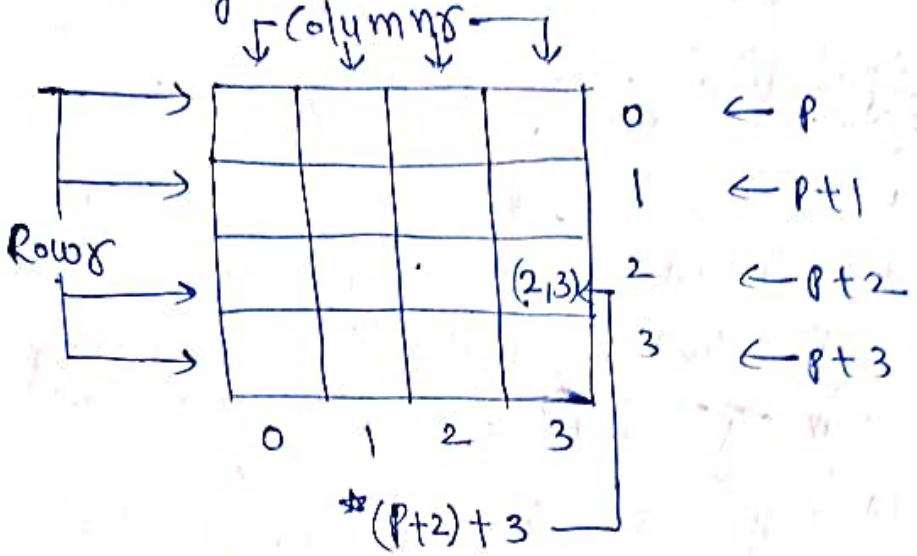
p+2 = &x[2] = 1004

address of x[3] = base address + 3 × scale factor of int

= 1000 + (3 × 2) = 1000 + 6

= 1006

Note that *(p+3) gives the value of x[3]. The pointer accessing method is much faster than array indexing.



p → pointer to the first row

p+i → pointer to the ith row

*(p+i) → Pointer to the first element in the ith row.

*(p+i)+j → pointer to the jth element in the ith row.

((p+i)+j) → value stored in the cell (i,j)

Pointers are used to iterate through arrays, access array elements, and perform various operations.

Iteration through an array:

```
#include<stdio.h>
int main() {
    int arr[] = {1,2,3,4,5};
    int *ptr = arr;
```

```
for (int i=0; i<5; i++) {
    printf("%d", *ptr);
    ptr++;
  }
return 0;
}
```

Modifying array elements:

```
#Include<stdio.h>
int main() {
    int arr[]={1,2,3,4,5};
    int *ptr = arr;
    for (int i=0; i<5; i++){
        (*ptr) += 10;
        ptr ++;
    }
  for(int i=0; i<5; i++) {
      printf ("%d", arr[i]);
  }
  return 0;
}
```

a += 10
a = a + 10
Both are same.

Finding the Sum of array elements:

```
#Include< stdio.h>
int main() {
    int arr[] = {1,2,3,4,5};
    int *ptr = arr;
    int sum=0;
```

```
for(int i=0; i<5; i++) {
    sum += *ptr;
    ptr++;
}
printf("Sum of array elements : %d\n", sum);
return 0;
}
```

Reversing an array :

```
#include<stdio.h>
int main() {
    int arr[] = {1,2,3,4,5};
    int *start = arr;
    int *end = arr+4;
    while(start<end) {
        int temp = *start;
        *start = *end;
        *end = temp;
        start++;
        end--;
    }
    for(int i=0; i<5; i++) {
        printf("%d", arr[i]);
    }
return 0;
}
```

06:02 AM Tue, 23 Jan 2024
Asked.

User-defined data types :-

Structures :

Arrays can be used to represent a group of data items that belong to the same type, such as int or float.

C supports a constructed data type known as structures, a mechanism for packing data of different types.

A structure is a convenient tool for handling a group of logically related data items.

eg: time : seconds, minutes, hours.
date : day, month, year
book : author, title, price, year.
city : name, country, population.

Structures help to organize complex data in a more meaningful way.

Defining a structure :

The keyword "struct" declares a structure.
The data fields inside the structure are called structure elements or members. Each member may belong to a different type of data.
Structure name is called the structure tag.
The general form is

```
struct tag_name
{
    data-type    member1;
    data_type    member2;
    ... :
```

In defining a structure, the following syntax will be needed,

(i) The template is terminated with a semicolon.

(ii) While the entire definition is considered as a statement, each member is declared independently for its name and type in a separate statement inside the template.

(iii) The tag name can be used to declare structure variables of its type.

Array Vs Structure:

The array and structure are classified as structured data types as they provide a mechanism that enable us to access and manipulate data in a relatively easy manner.

Array	Structure
→ Collection of related data items.	→ Elements of different types.
→ Derived data type	→ User-defined data type

Declaring Structure Variables:

A structure variable declaration is similar to the declaration of variables of any other data types.

It includes the following elements:

→ The keyword struct
→ The structure tag name.
→ List of variable names separated by commas

→ A terminating semicolon.

eg:
```
struct book_bank
{
    char title[20];
    char author[15];
    int pages;
    float price;
} book1, book2, book3;
```

eg:
```
struct book_bank
{
    char title[20];
    char author[15];
    int pages;
    float price;
}
struct book_bank, book1, book2, book3;
```

Type-Defined Structures:

We can use the keyword typedef to define a structure.

```
typedef struct
{
    type member1;
    type member2;
    ------
    ------
} type_name;
```

The type_name is the type definition name.

We cannot define a variable with typedef declaration

Accessing structure Members:

The members themselves are not variables. They should be linked to the structure variables in order to make them meaningful members.

The link between a member and a variable is established using the member operator '.' which is known as 'dot operator' or 'period operator'

eg: strcpy (book1.title, "BASIC");
book1.pages = 25;
scanf ("%s\n", book1.title);
scanf ("%d\n", &book1.pages);

Structure Initialization:

eg: struct student
{
int weight;
float height;
} stu1 = {60, 123.15};

main()
{
struct student stu2 = {53, 135.25};
.....
}

C language does not ~~specify~~ permit the

initialization of individual structure members within the template.
The initialization must be done only in the declaration of the actual variables.

Word Boundary and Slack Byte:

Computer stores structures using the concept of "word boundary". The size of a word boundary is machine dependent.

In a computer with two bytes word boundary, a character data takes one byte and an integer takes two bytes. One byte b/w them is left unoccupied. This unoccupied byte is known as the "slack byte".

Three ways to access structure members:

→ Using dot notation : v.x
→ Using indirection notation : (*ptr).x
→ Using selection notation : ptr → x.

Unions :-

A union is a variable that may hold objects of different types and sizes, with the compiler keeping track of size and alignment requirements.

Unions provide a way to manipulate different kinds of data in a single area of storage, without embedding any machine-dependent information in the program.

Syntax for based on structures:

```
union student {
    int num;
    float marks;
    char name[10];
} stu;
```

Unions may occur within structures and arrays. The members within a union all share the same storage area within the computers memory, where as each member within a structure is assigned its own unique storage area.
Unions are used to conserve memory.

- Union is a required keyword and the other terms have the same meaning as in structure definition.

eg:

```
union id {
    char color[12];
    int size;
};
struct clothes {
    char manufacturer[20];
    float cost;
    union id description;
} shirt, blouse;
```

An individual union members can be accessed in the same manner as an individual structure manner, using the operators '.' and '->'.

A union variable can be initialized provided its storage class is either external or static.

Definition:

A union is a user-defined data type that allows you to store different data types in the same memory location.

Example:

```
#include<stdio.h>
struct Point {
    int x;
    int y;
};
union Variant {
    int IntValue;
    float FloatValue;
    char stringValue[20];
};
int main() {
    struct Point P1 = {3,5};
    printf("Point : (%d, %d)\n", P1.x, P1.y);
```

```
union Variant var;
var.intValue = 10;
printf("Integer Value : %d\n", var.intValue);

var.floatValue = 3.14;
printf("Float Value : %f\n", var.floatValue);

return 0;
}
```

	Structure	Unions
Keyword:	Struct keyword is used to define a structure.	The keyword union is used to define a union.
Size:	The compiler allocates the memory for each member.	The compiler allocates the memory by considering the size of the largest memory.
Memory:	Each member within a structure is assigned unique storage area of location.	Memory allocated is shared by individual members of union.
Value Altering:	Altering the value of a member will not affect other members of the structure.	Altering the value of any of the members will alter other member values.
Accessing Members:	Individual member can be accessed at a time.	Only one member can be accessed at a time.

08:17 AM Tue - 23 - Jan - 2024