

DATA STRUCTURE

DATA

- a fact or figure from which conclusions may be drawn
- Basic unit of information is the BIT

BIT

- A smallest accessible unit of memory [Use Binary Number System]
- A switch representing two different states

Switch States are : ON = 1
OFF = 0

The Value stored in a bit are mutually exclusive and disjoint

No of Switches using two bits

0	0
0	1
1	0
1	1

Here we have four distinct states

No of Switches in 3 bits

0 0 0

0 0 1

0 1 0

0 1 1

1 0 0

1 0 1

1 1 0

1 1 1

We have mutually exclusive
distinct states

n

When n bits, it is possible to represent 2^n different states.



Types of data :

- 1) Numeric
- 2) Non- numeric

Numeric

- Binary
- Integer
- Real

Binary

One digit can represent utmost two values

TRUE / FALSE [ie YES or NO]

Only used to represent logical values

Example :

$A = 10; B = 02; C = 23;$

Is $A > B$?

Is A Equals To C ?

INTEGERS :

- Used to store precise information

E.g: No of heads in this class

 No of Cars crossing a specific place

- * Int. are stored in a series of consecutive bits.

- * Usually int. occupy 4 bytes [Depends OS]

How INT are represented in Computer Memory ?

Lets say 22 is stored as 10110

The largest integer that can be stored using n Binary digit is 2^{n-1}

Example : No of bit = 4

$$\text{Max Value} = 2^4 - 1$$

$$= 16 - 1$$

$$= 15$$

It's Eqvlnt. Binary No = 1111

PROCEDURE BINARY_TO_INT

Step 1: Read len_of_pattern, bit_array

Step 2: Set Index = 0

Step 3: Set Int_sum=0;

Step 4: Set string_index = len_of_pattern

Step 5: Set int_sum = int_sum + bit_array(string_index) * 2

Step 6: Set index = index + 1

Step 7: If index .eq. Len_of_pattern Then Goto Step 10

Step 8: Set string_index = string_index - 1

Step 9: Goto Step 5

Step 10: Print int_sum

Step 11: STOP

Example : $1111 = 15$



Representing -ve [Negative] Numbers

- Sufficient if we can represent -ve Binary numbers

Methods to represent -ve numbers

- * One's Complement
- * Two's Complement

ONE'S COMPLEMENT :

- Change all bits to it's complement

Eg. One's complement of 1111 is 0000

With this method, the MSB [Most significant Bit] is reserved as the sign bit.

ie with n bits, the range of numbers that can be represented is

$-2^{(n-1)} + 1$ 1 followed by n-1 zeroes

$2^{(n-1)} - 1$ 0 followed by n-1 ones

Eg. With 3 bits, -3 to +3 can be represented

Disadvantages in Ones Complement

There are two representation for the number zero:

- a +ve zero consisting of all 0's
- a -ve zero consisting of all 1's

Two's Complement

Step 1: Convert given binary value into One's comp.

Step2: Add 1 with the LSB [Least significant bit]

Eg.	0101	5
	1010	-5 in 1's complement
	1011	-5 in two's complement

0 has a unique representation

E.g2.	0000	+0 in one's complement
	1111	-0 in one's complement
	0000	0 in 2's complement
	10000	

With n bit, the range of numbers that can be represented using two's complement is from

$-2^{(n-1)}$	\implies	1 followed n-1 zeroes
to $2^{(n-1)}$	\implies	0 followed by n-1 ones

Example : with 3 bits, we have

100 -4

101 -3

110 -2

111 -1

000 0

001 1

010 2

011 3

Real Number Representation :

- Integers represent precise info
- Real represent approximation
- Used to represent continuous quantities

Example : Distance between two points

Qty of water remaining in a bottle

Floating Point / Scientific Notation :

- Mantissa
- Base - is fixed for all numbering system -10 for decimal
- Exponent

Example :

A real no with 24 - bit mantissa and 8 bit exponent is represented by a string of 32 bits

$$15.2 = 0.152 * 10^{-2} \text{ or it equals to } 152 * 10^{-1}$$

The 24 bit binary representation of 38753 is
00000000 10010111 01100001 and 8 bit two's
complement of -2 is 11111110;

the representation of 387.53 is

00000000 10010111 01100001 11111110

We chose the representation in which the mantissa is an integer with no zeros.

Character / Strings :

Each Character is assigned an 8 bit representation

Note :

256 Characters can be represented with 8 bits

String means concatenation of more than one characters

Ex : “Asia Pacific Institute of Information Technology”

“LIM” is represented by 01001100 01001001 01001101

DATA TYPE

Kind of data stored using variable in a programming language.

DATA STRUCTURE

_____ Organization and manipulation of data inside a computer

Goals of the study on Data Structure

- * To identify and develop mathematical entity and their operations (Entity :- Something that exist alone)
- * To identify the class of problems those can be solved using these
- * To develop representations for these entities and to implement them

Efficiency of a data structure

- * SPACE

- * TIME

Space & Time tradeoff

DATA TYPE IN C

Integer :

- * Long Integer
- * Short Integer
- * Unsigned Integer
- * Signed Integer

Char, Float, Double [For Double precision float]

A Type Declaration specifies

- # bytes to be allocated to a variable of the type
- How the data represented by the bit-string is to be interpreted ? [int ? Float ? Char?]

Variables in C language

A variable preparation in C specifies two things

- * it specifies the amount of storage that must be set aside
- * Second, it specifies how data represented by string of bits are to be interpreted

POINTERS IN C

A pointer is a variable. If z of type Y then &z is a pointer and it points to a location where a variable of type Y is stored in memory.

Eg.

```
Int *ptr1;
```

```
float *ptr2;
```

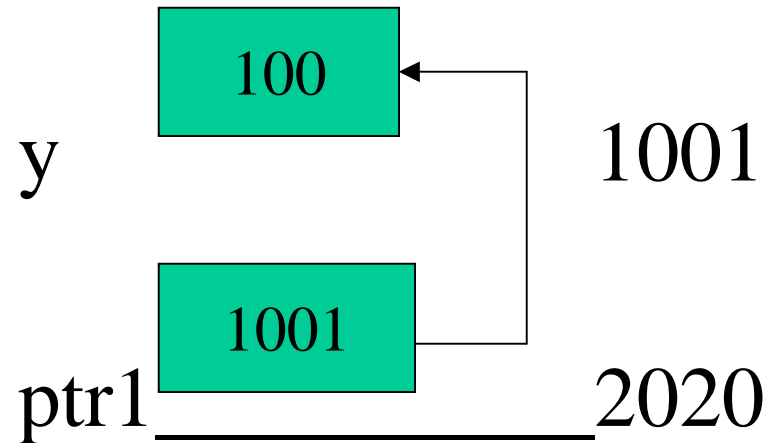
```
char *ptr3;
```

Value can be assigned to pointers similar to assigning values to variable

Eg. Ptr1 = &x;

Note : &x is an integer value

```
Int y=100;  
int *ptr1;  
ptr1 = &y;  
printf("\n Value of y = %d Ptr Value = %d",y,*ptr);
```



About Pointers

ADDRESS ARITHMETIC

You can use the following operations on pointers:

- add an integer to a pointer
- subtract an integer from a pointer
- get the difference of two pointers (must point to same object)
- compare pointers for equality

Actions NOT allowed:

- add, multiply, divide or shift pointer variables.
- E.g. `int_ptr a * int_ptr b` is not allowed.

Pointers



Pointers - Cont..

Exercise C24 POINTERS & ARRAYS

```
char s[]="Hello", **p,*t;
```

```
t = s;
```

```
p = &t;
```

```
printf("%s",*p);
```

Pointers Cont....

Exercise C24 POINTERS & ARRAYS

```
float a[5]={4,5,6,7,8};
```

```
float *p1, *p;
```

```
p1 = p2 = a;
```

```
p1++;
```

4 5 6 7 8

```
p1++;
```

Pointers Cont.....

Exercise C24 POINTERS & ARRAYS

```
main()
```

```
{ char blocks[4]={ 'a','b','c' };
```

```
char* ptr=&blocks[0];
```

```
char temp;
```

```
temp=block[0];                      a
```

```
temp=*(blocks+2);                  c
```

```
temp=*(ptr+1);                    b
```

```
temp=*ptr;                        a
```

```
}
```

Pointers

Pointers Cont....

```
#include<stdio.h>
```

```
main()
```

```
{
```

```
    char b[3]={'a','b','c'};
```

```
    char* ptr=&b[0];
```

```
    char temp;
```

```
    temp=b[0];
```

```
    printf("\n Value Temp=%c",temp);
```

Pointers

Pointers Cont...

```
temp=*(b+2);  
printf("\n Value Temp=%c",temp);  
temp=*(ptr+1);  
printf("\n Value Temp=%c",temp);  
temp=*ptr;  
printf("\n Value Temp=%c",temp);  
ptr=b+1;  
temp=*ptr;
```

Pointers Arithmetic

_____ *pi +a # *(pi+a)

- * Used extensively to pass parameters by reference. Here only one parameter need to be passed. Also resulting values need not passed back, since changes are reflected in the location.
- * By means of a pointer, a function can modify variables of another function.

Ex :

```
x=5;
```

```
printf(“%d\n”,x);
```

```
funct(&x);
```

```
printf(“%d\n”,x);
```

```
..
```

```
funct(*py)
```

```
int *py;
```

```
{    int *py;
```

```
    ++(*py);
```

```
    printf(“%d\n”,*py);
```

```
}
```

ARRAYS

- * a finite ordered set of homogeneous elements arranged in a list

Eg. `Int num[10];`

- specifies an array of 10 integers used to store homogeneous data

Basic Operation

1) Extraction :- a function that takes an array `a` and index `i` and returns an element of the array. The result of this operations is denoted by `a[i]`;

result : Value of `a[I]` is returned

2) Storing

a fn that takes an array a , an index I and an element x . The operation is denoted by the stmt
 $a[I] = x$;

results : Value of x is set to $a[I]$

Initialization :

this should be the first operation on the array. Reference to an uninitialised array is illegal.

- a specific case of storing

Lower Bound : smallest element of an array's index

Lb always fixed to ϕ

Upper Bound : highest elements of an array's index.

range :- # elements in an array

$$= \text{Upper Bound} - \text{Lower Bound} + 1$$

```
int x[50];
```

```
lb = 0;
```

```
ub=49;
```

$$\text{range} = 49 - 0 + 1 = 50$$

Note : the lb and ub and range cannot be changed during program execution



Array Implementation in C

An array can be implemented by the declaration

```
int my_array[10];
```

- means, an my_array can have 10 consecutive memory locations reserved base address.
- address of first location, denoted by `base(my_array)`.

How to determine the address of an element in an array?

Let `int_size` be # bytes for integers, then address of I^{th} element is calculated by

$$\text{base}(\text{my_array}) + I * \text{int_size}$$

An array name/ variable stored as a pointer.

A string is stored as an array of chars terminated by a null char `\0`.

Passing arrays as parameters by reference saves time and space

Example : Using One Dimensional array:

```
#define NUMBER 100;
```

```
aver() {
```

```
    int num[NUMBER], I,total;
```

```
    float total,diff;
```

```
    total=0;
```

```
    for(I=0;I < NUMBER;I++) {
```

```
        scanf("%d",&nm[I]);
```

```
        total += num[I];    }
```

```
    avg = total / Number;
```

```
    printf("\n Number Difference ");
```

```
for(I=0;I < NUMBER;I++) {  
    diff = num[I] - avg;  
    printf(“\n %d  %d”,num[I],diff);  
}  
____printf(“\n Average is : %d “,avg);  
}
```

Arrays Application :-

Sorting :- Arranging data in ascending / descending order.

BUBBLE SORT

 32 51 27 85 66 23 13 57

Pass 1:

32 27 51 66 85 23 13 57

32 27 51 66 23 85 13 57

32 27 51 66 23 85 13 57

32 27 51 66 23 13 85 57

32 27 51 66 23 13 57 85

Pass 2:

<u>27</u>	<u>32</u>	51	66	23	13	57	85
27	32	51	<u>23</u>	<u>66</u>	13	57	85
27	32	51	23	<u>13</u>	<u>66</u>	57	85
27	32	51	23	13	<u>57</u>	<u>66</u>	85

Pass 3:

27	<u>32</u>	<u>51</u>	23	13	<u>57</u>	<u>66</u>	85
27	32	<u>23</u>	<u>13</u>	51	57	66	85

Pass 4:

27	23	32	13	51	57	66	85
27	23	13	32	51	57	66	85

Pass 5:

23 27 13 32 51 57 66 85

23 13 27 32 51 57 66 85

Pass 6:

13 23 27 32 51 57 66 85

Algorithm

Step1 : BUBBLE(DATA,N) /* DATA is an array
with N elements

Step2: Repeat Step3 &4 for K=1 to N-1

Step3: Set PTR=1

Step4: Repeat while PTR <= N-K [Execute Pass]

a) If DATA[PTR] >= DATA[PTR+1] then

Swap DATA[PTR] & DATA[PTR+1]

End If

b) Set PTR = PTR+1

End of Inner Loop

End of Step4 Loop

Step5: End

SEARCHING

A) Linear Searching

B) Binary Searching

Algorithm

Step1 : LINEAR(DATA,N,ITEM,LOC) /* DATA is an
array with N elements

Set LOC=1;

Step2: Search for ITEM

Step3: Repeat while DATA[LOC] # ITEM

Set LOC = LOC+1

End Loop

Step 4: [If successful] Print LOC

Step5: Exit

Binary Searching : Algorithm

BINARY(DATA, LB, UB, LOC, ITEM)

Step1: Set $BEG = LB$; $END = UB$; $MID =$
 $INT(BEG + END) / 2$

Step2: Repeat step 3&4 while
 $BEG \leq END$ and $DATA[MID] \neq ITEM$

Step3 If $ITEM < DATA[MID]$ then

Set $END = MID - 1$

Else

Set $BEG = MID + 1$

End If

Step4 Set $MID = INT((BEG + END) / 2)$

End of Step2 Loop

Binary Search Cont.....

Step5 If DATA[MID] = ITEM then

Set LOC = MID

Else

Set LOC=NUL

End If

Step6 Exit

2 - D Arrays

- * a 1-d array with components type as another 1-D array
 - * each dimension has a range
 - * an element is referenced using two indices
 - * second dimension is only logical
(data is stored linearly)
- E.g. Chess Board, Matrix, Tables etc

ROW MAJOR REPRESENTATION

Eg. Int ar[r1][r2];

Address of ar[i1][i2] is

_____ $\text{base}(\text{ar}) + (\text{i1} * \text{r2} + \text{i2}) * \text{type_size}$

COLUMN MAJOR REPRESENTATION

Eg. Int ar[r1][r2];

Address of ar[i1][i2] is

_____ $\text{base}(\text{ar}) + (\text{i1} * \text{r1} + \text{i2}) * \text{type_size}$

MULTI-DIMENSIONAL ARRAYS

____3-D

- PLNE

-ROW

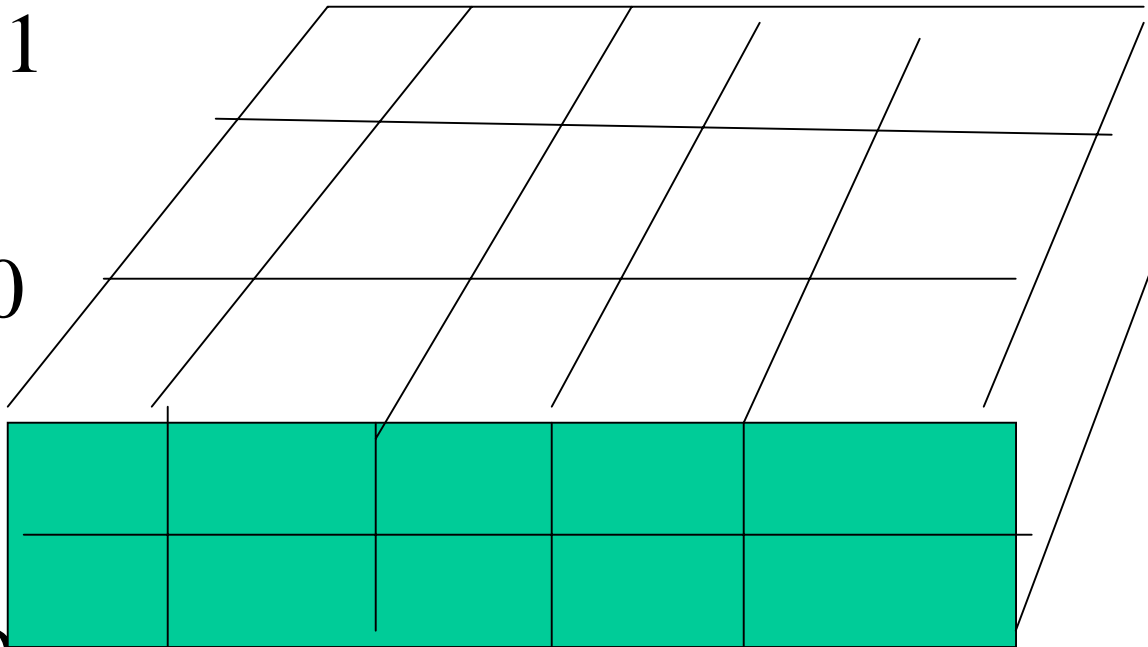
-COLUMN

PLANE 1

PLANE 0

ROW

COLUMN



In N Dimensional Array , the total no of elements can be determined by the following formula

n

$$\prod_{k=1}^n i_k$$

elements

Example: `int b[3][2][4];`

3 = rows

2 = plane

4 = columns

STRUCTURES IN C

_____ A structures is a collection of heterogeneous items. Each item in this collection is termed as a member of the structure.

Eg.

```
Struct { char name[20];  
        int studno;  
        float tax;  
        }lim,sridnesh;
```

This creates two structure variable named lim and sridnesh.

- * First member is a char array.
- * Second member is an integer.
- * Third member is a float.

Tag : Declaring structure using tags.

A tag is attached to the structure.

This tag is used to declare variable of that type of structure

Eg.

```
Struct stud_info {  
    char stud_name[30];  
    int stud_no;  
    short rank;  
};  
:  
struct stud_info lim,sridnesh;
```

```
/* Program to illustrate a structure */  
#include <stdio.h>  
struct date { /* global definition of type date */  
    int month; int day; int year; };  
main() {  
    struct date today;  
    today.month = 10;  
    today.day = 14;  
    today.year = 1995;  
    printf("Todays date is %d/%d/%d.\n",  
        today.month, today.day,  
        today.year );  
}
```

```
/*  structure pointers */
#include <stdio.h>
main( )
{
    struct date { int month,
                  day, year;};
    struct date today, *date_ptr;
    date_ptr = &today;
    date_ptr->month = 9;
    date_ptr->day = 25;
    date_ptr->year = 1983;
    printf("Todays date is
    %d/%d/%d.\n", date_ptr->month, \
    date_ptr->day, date_ptr->year %
    100);}
```

Structures

Declaring Structure With *typedef*

E.g. typedef struct{
 stud_info; }
 sdud_info lim,sridnesh;

Accessing a member of a Structure

< structure name> . <member name>

E.g.

Int x,I;

:

x=sridenesh.stud_no;

lim.stud_name[I]=""Oliver";

Composite Structure :

A structure may contain one or more other structures.

E.g.

```
Structure add_type{  
    char straddr[30];  
    char city[19];  
    char zip[20];  
}  
:
```

```
Strcture nmodtype{
```

```
    struct nametype;
```

```
    struct addtype;
```

```
};
```

```
:
```

```
    struct nmodtype my_name_add,  
                    your_name_add;
```

```
my_name_add.addtype.city[4]=
```

```
your_name_add.addtype.city[4];
```

The amount of memory allocated vary from one machine to another.

A type indicates a specific amount of storage

E.g.

```
Struct test{  
    int x;  
    float y;  
    char c[10];  
};
```

bytes required = 4 + 8 + 10 bytes

But on some computers, int/float may begin only at address that are multiples of 4/8.

So we need few more bytes to store a structure.

E.g. Base address of test = 200

test.x begins at 200

test.y begins at 208

test.c begins at 216

test.ends at 225

Total memory 26 bytes

Wastage 4 bytes.

UNIONS

Used in structures

Allows a variable to take different types of data during program execution.

E.g

```
#define INTEGER 1  
#define REAL 2.0;  
struct stint{  
    int f3,f4;};  
struct stfloat{  
    float f5,f6;};
```

```
struct sample{
```

```
    int f1;
```

```
    float f2;
```

```
    int utype;
```

```
union {
```

```
    struct stint x;
```

```
    struct stfloat y;
```

```
} function;
```

```
}
```

Members in sample f1,f2 and utype requires
 $4+4+8=16$ bytes

The first member of UNION x requires 8 bytes.

The second member of UNION req. 16 bytes.

Note : The memory allocated for the UNION part of such a variable is the maximum of the space needed by any single member.

- * So, in this case, 16 bytes are allocated for the UNION part of sample.
- * Added to 16 bytes needed for the fixed part, 32 bytes are allocated to sample.

Conclusion :

A structure may be regarded as a road maps for the same area of memory is to be interpreted.

A UNION provides several road maps for the same area of memory and it is the responsible for the programmer to determine which road map in current use.

THE END OF FIRST LESSON