Pointers

→ Advantages of pointers

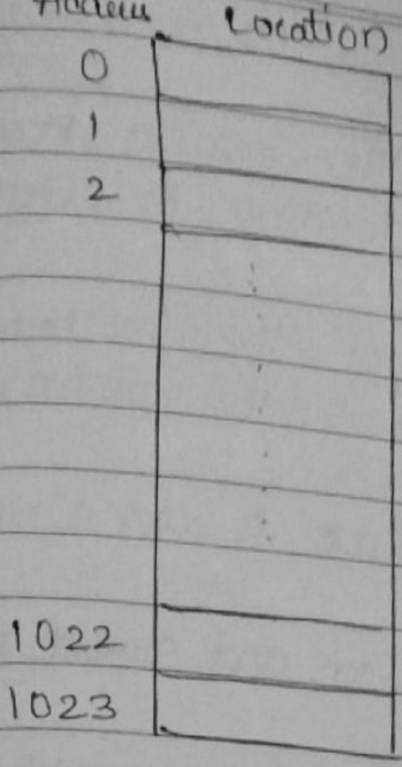
1. Enable us to write efficient and concise program
2. Enable us to establish inter-program data commun-
   - cation.
3. Enable us to dynamically allocate and de-allocate
   memory
4. Enable us to Optimize memory space usage.
5. Enable us to deal with hardware components
6. Enable us to pass variable numbers of arguments
   to functions.

→ let us learn overview of the organization of memory before stepping into the concept of pointers.

* Memory is organized as an array of bytes.
* A byte is a basic storage and accessible unit in memory.

- Each byte is identifiable by a unique number called address.

Eg: suppose we have 1KB of memory.
    since 1KB = 1024 Bytes, the memory can be viewed as an array of locations of size 1024 with the subscript range [0-1023].



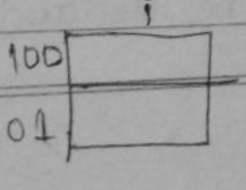
→ We know that variables are to be declared before they are used in a program.

→ Declaration of a variable tells the compiler to perform the following.

1. Allocate a location in memory. The number of bytes in the location depends on the data type of the variable.

2. Establish a mapping between the address of the location and the name of the variable.

Eg:- declaration int i; → the compiler reserve a location in memory.



∵ size of a variable of int type is two bytes.

Variable Name   Address of location
i   100

→ A mapping between the variable name and the address of the location is established.

→ Note that the address of the first byte of the location is the address of the variable.

→ The address of a variable is also a number, numeric data item.

→ It can also be retrieved and stored in another variable.

→ A variable which can store the address of another variable is called a pointer.

Pointer operators[&, *]

- C provides two special operators known as pointer operators. They are & and * stands for

& - 'Address of' → to retrieve the address of var.

* - 'Value at Address'.

↓

to access the value at a location by means of its address.

* Since pointer is also a variable, it also should be declared before it is used.

The syntax of declaring a pointer variable is

data-type *variable-name;

data-type ↓ is any valid datatype supported by C

* ↓ presence of * before variable-name indicates that it is a pointer variable.

variable-name → pointer variable

Eg: int *ip; → ip is declared to be a pointer variable of int type.

float *fp; → fp is declared to be a pointer variable of float type.

Eg: To illustrate the concept of pointer and pointer operators [&, *]

```
# include <stdio.h>
int main()
{
 int i=10, *ip;
 float f=3.4, *fp;
 char c='a', *cp;

 printf ("i= %d \n", i);
 printf ("f= %f \n", f);
 printf ("c= %c \n", c);
ip= &i;
printf ("\n Addrss of i= %u \n", ip);
printf ("value of i = %d \n", *ip);

fp= &f;
 printf ("\n Addrss of f = %u \n", fp);
 printf ("value of = %f \n", *fp);

 cp= &c;
 printf ("\n Addrss of c= %u \n", cp);
  printf (" value of c= %c \n", *cp);
 return 0;
}
```

Output:

```
    i=10
    f=3.400000
    c=a
 Addrss of i= 65524
 value of i=10

 Addrss of f= 65520
 value of f = 3.40000

 Addrss of c= 65519
  value of c = a
```

pointer Arithematic:

→ The following are the operations that can be performed over pointers.

1. we can add an integer value to a pointer.
2. we can subtract an integer value from a pointer.
3. we can compare two pointers if they point to the elements of the same array.
4. we can subtract one pointer from another pointer if both point to the same array.

5. we can assign one pointer to another pointer provided both are of same type.

But the following operations are not possible.

1. Addition of two pointers
2. subtraction of one pointer from another pointer when they do not point to the same array.
3. multiplication of two pointers
4. Division of one pointer by another pointer

Suppose p is a pointer variable to integer type, the pointer variable p can be subjected to the following operations.

1. We can increment it using increment operator ++

Eg:- p initially is storing addr 100 after p++ is executed, the content p gets incremented by 2, i.e., 102 the size of int type

2. We can decrement it using decrement operator--

p storing 100 initially
after decrement p-- is executed the content in p gets decremented by 2, so it becomes 98.

* In general, if a pointer variable is incremented using ++, it gets incremented by the size of its data type i.e.,
- In case of decrement operator, a pointer variable gets decremented by the size of its datatype.

3. An integer value can be added to it.
p = p + integer value

- The content of p will now get incremented by the product of integer value and the size of int type.

4. An integer value can be subtracted from it
Eg: p=p- Integer value

- The content of p will now get decremented by the product of integer value and the size of int type.

program to illustrate the concept of pointer arithematic.

```
# include <stdio.h>
int main()
{
  int i, *ip;
  ip=&i
  printf("ip= %u \n", ip);
  ip++;
  printf("After ip++  ip = %u \n",ip);
  ip--;
  printf("After ip--  ip = %u \n", ip);
  ip= ip+2;
  printf("After ip=ip+2  ip= %u \n", ip);
  ip=ip-2;
  printf("After ip=ip-2  ip= %u \n", ip);

  return 0;
}
```

output:

```
 ip= 65524
After ip++          ip=65526
After ip--          ip=65524
After ip=ip+2       ip = 65528
After ip=ip-2       ip =65524.
```

pointer Expressions:

- Once we assign the address of a variable to a pointer variable, the value of the variable pointed to can be made to participate in all manipula -tions by means of the pointer itself.

program to illustrate pointer expressions.

```
#include <stdio.h>
int main()
{
  int a=4,b=2, *ap, *bp, *sp;
  int s,d,p,q,r,t;
  ap=&a;
  bp=&b;

  s= *ap + *bp;
  d = *ap - *bp;
  p= *ap * *bp;
q= *ap / *bp;
r= * ap % *bp;

sp= &t;
*sp= *ap + *bp;

printf(" Sum= %d \n", s);
printf(" Difference = %d \n ", d);
printf(" product = %d \n", p);
printf (" Quotient = %d \n", q);
printf(" Remainder = %d \n", r);
printf(" Sum= %d \n", t);

return 0;

}
```

O/P:

```
sum= 6
Difference = 2
product = 8
Quotient = 2
Remainder = 0
sum= 6.
```

pointers and Arrays

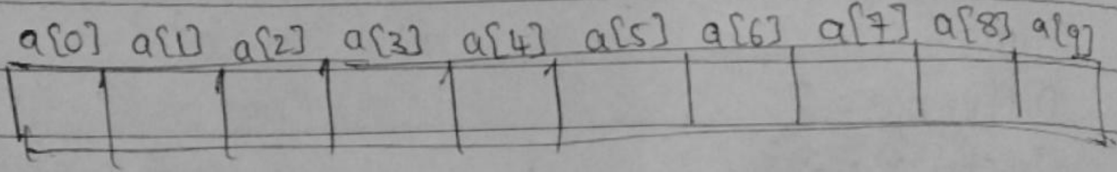
- Elements of an array are accessed through pointers internally.

- pointers and one-dimensional Arrays.

→ let us first consider relationship between one-dimensional arrays and pointers.
Suppose a is a one-dimensional array of int type and of size 10, which is declared as follows

```
int a[10];
```

- we know that a block of memory consisting of 10 contiguous locations gets allocated and all the locations share common name a and are distin-guishable by subscript values [0-9].



→ Here the array name a gives the base address of the array. i.e., the address of the first element a[0] of the array.

→ So a being equivalent to &a[0].

Since a is a constant pointer it cannot be incremented to point to the next location.
But the expression (a+1) gives the address of the second element a[1] of the array.
*(a+1) gives the element itself stored at a[1]

Program to illustrate processing of one-dimensional arrays using pointers.

```
#include <stdio.h>
int main()
 {
int a[10], n, i;
Printf ("Enter the number of elements \n");
Scanf ("%d", &n);
Printf (" Enter %d number elements \n", n);
for (i=0 ; i<n ; i++)
 Scanf ("%d", *a+i);
 Printf (" The List of elements \n");
 for (i=0; i<n; i++)
 Printf (" %d", *(a+i));
 return 0;
  }
```

pointers and strings

* strings also being arrays enjoy the same kind of relation with pointers.

→ we know that a string is a sequence of characters terminated by a null character '\0'.

→ we know that the name of an array gives the base address of the array i.e., the address of the first element of the array. This true even in the case of strings

Eg: char s[20];

s → gives the base address of domain

∴ string can be denoted by a pointer to charac.

~~cha~~ char *cp;

cp=s;

Here s is declared to be an array of char and it can accomodate a string. To be precise, s represents a string.

cp is declared to be a pointer to char type

- cp also represents the string in s. Let us consider a string constant "abcd".
- we know that a string constant is a sequence of characters enclosed within a pair of double quotes.

- The string constant is stored in some part of memory and it requires five bytes of space.
- Here also the address of the first character represents the entire string
→ so, it can be assigned to a pointer to char type

If cp is a pointer to char type, cp = "abcd" is perfectly valid and cp now represents the entire string "abcd".

→ program to illustrate pointers to strings

```
#include <stdio.h>
int main()
{
  Char str[20], *cp;
  printf ("Enter a string \n");
  scanf (" %s", str);
  cp=str;
  printf (" string in str = %s \n", cp);
  cp="abcd";
  printf("%s \n", cp);
  return 0;
}
```

output :-

```
Enter a string
program
string in str= program
abcd
```

comparison between an array of char and a pointer to char

Array of char	Pointer to char
- we can intialize a string during declaration char str[20] = "abcd"	- we can initialize a string during a declaration char* cp = "abcd"
- A string variable or a string constant cannot be assigned to an array of char type. char str[20]; str= "abc" is invalid.	- A string variable or string constant can be assigned to a pointer to char type char *p; p="abcd";
- The number of bytes allocated for a string variable is determined by the size of the array	- The number of bytes allocated for a string is determined by the number of characters within string.
- An array of char refers to same string	- A pointer to char can represent different strings at different points of time.
- A string can be accepted through the console char str[20]; scanf("%s", str); is valid	- A string can not be accepted through the console char *cp; scanf("%s", cp) is not valid since cp contains garbage value.

Arrays of pointers to strings

→ If we are to deal with more than one strings, then we can use an array of pointers to char type to represent the strings.

Eg: Declaration of an array of pointer to char type.

```
        char *ptr[10];
```

Here ptr[0], ptr[1] ... ptr[9] are all pointers to char type and each of them can denote a string.

– program to illustrate an array of pointer to string

```
#include <stdio.h>
int main(void)
{
    char * names[5]= {"Nishu", "Harsha", "Shobha",
                       "Devaraj", "Asha"};


    int i;
    printf(" The list of five Names \n");
    for(i=0; i<5; i++)
        printf("%s \n", names[i]);
    return 0;
}
```

→ program to sort a list of names using array of pointers to strings

```
# include <stdio.h>
# include <string.h>
int main (void)
{
char *t, *names[5] = {"Nishu", "Harsha","Shobha","Devraj","Asha"};
int i,j ;
Printf (" The unsorted list of five names \n");
for (i=0 ; i<5 ; i++)
Printf (" %s \n", names[i]);

/* Sorting begins */
for (i=0 ; i<4 ; i++)
for (j=i+1 ; j<5 ; j++)
if (strcmp (names[i] , names [j]) >0)
{
 t= names [i];
names [i] = names [j] ;
names [j] = t ;
}
   /* Sorting ends */
Printf ("Sorted list of names \n");
for (i=0; i<5 ; i++ )
Printf (" %s \n", names[i]);
return 0;
}
```

pointer and functions

- passing pointers as arguments to function
- Returning a pointer from a function.
- pointer to a function.
- passing one function as an argument to another funtion.

⇒ passing pointers as Arguments to functions

- The mechanism of calling a function by passing pointer is called call by Reference.

- If a function is to be passed a pointer, it has to be reflected in the argument-list of function prototype and the header of the function definition.

Eg: void display(int *);

- The function prototype indicates that the function display() requires a pointer to int as its argument.

```
void display(int *p)
{
  statements;
}
```

while calling the function display(), it needs to be passed the address of an integer variable
display(&a); where a is a var. of int type.

program to illustrate call by value and call by refeence.

```
# include <stdio.h>
void inc( int);
void incr(int *);

int main( )
{
int a=10 , b=10;
printf("call By value \n");
printf ("a= %d \n", a);
inc (a);
printf(" After calling inc() \n");
printf("a = %d \n", a);
printf(" call By refeence \n");
printf( "b= %d \n", b);
incr(&b);
 printf (" After calling incr() \n");
 printf(" b= %d \n", b);
 retuen 0;
}
void inc(int x)
{
 x++;
 printf(" x= %d \n", x);
 }
```

```
void incr (int *p)
{
    (*p)++
}
```

Output:

```
call By value
 a=10
 x=11
After calling inc()
 a=10
 call By Reference
 b=10
 After calling incr()
  b=11
```

② Returning a pointer from a Function.

- Under some circumstances, we require functions to return a pointer.

- The fact that a function returns a pointer should be indicated by its declaration and definition

syntax of declaration:

```
data-type *function_name (arguments);
```

* before function name indicate that the function returns a pointer to datatype

Eg: int *sum (int, int);

The funtion sum() is declared to indicate that it requires 2 parameters of int type and it returns a pointer to int type.

program to illustrate function returning a pointer

```
#include <stdio.h>
int * sum( int, int);
int main()
{
int a, b, *s;
printf(" Enter two numbers \n");
scanf ("%d %d", &a, &b);
printf(" a= %d  b= %d \n", a, b);
 s= sum(a, b);
 printf(" sum= %d \n", *s);
 return 0;
}
int * sum( int a, int b)
 {
   static int s;
    s= a+b;
   return (&s);
 }
```

output ;
```
Enter two numbers
        4 5
      a=4  b=5
      s=9.
```

Pointer to Functions

- In case of arrays, we know that name of an array gives its base address and thus acts as a pointer to the first location of the array.

- This is true even in the case of functions.
- Name of a function also gives its starting address.
- Thus we can use a pointer to invoke a function before which the pointer has to be declared appropriately and assigned the address of the function.

Syntax of declaring a pointer to a function

data-type (* Variable-name) (argument-types);

Variable-name is declared to be a pointer to a function, which returns a value of type data-type.

Eg: int (*fnp) (int, int);

=> fnp can now be assigned the address of a function accepting 2 arguments of int type and returning a value of int type.

```
int sum( int, int);
  fnp = sum;
```

fnp now points to the function sum().

- program to illustrate pointer to function.

```
#include<stdio.h>
int sum(int, int).
int main()
{
int a,b,s (*fnp)(int,int);
printf ("Enter 2 numbers \n");
scanf(" %d %d "; &a, &b);
 printf(" a= %d b= %d \n", a,b);
 fnp=sum;
  s= (*fnp)(a,b);
  printf("sum=%d \n",s);
  return0;
}
int sum (int a int b)
{
  int s;
  s=a+b;
  return s;
 }
```

output:-

```
Enter 2 numbers
 4 6
a=4  b=6
sum=10.
```

Pointers to pointers

- we now know that a pointer is a special type of variable and it can collect the address of another variable.

→ Once a pointer is assigned, the address of a plain variable, the value of the plain variable can then be accessed indirectly through the pointer with the help of * (value at address) ope

- Since the pointer is also a variable, it is also allocated memory space of size two bytes and its address also is retrievable.

- The address of the pointer variable itself can also be stored in another appropriately declared variable

- The variable which can store the address of a pointer variable itself is termed a pointer to pointer.

- Note that a pointer to pointer adds a further level of indirection.

The syntax of declaring a pointer to pointer

datatype **variable;

↙

Indicate that the variable is a pointer to a pointer to a variable of type data-type.

Eg:- int a=10, *ap, **app;

a - is a plain integer variable.
ap - is a pointer to int type.
app - pointer to pointer to int type.

ap can be assigned address of the variable a
ap= &a;
app= &ap;

app is assigned the address of ap.

. Program to illustrate a pointer to pointer.

```
#include <stdio.h>
int main(void)
{
int a=10, * ap, ** app;}
 ap= &a ;
 app = &ap ;
Printf ("Address of a=%u\n", ap);
Printf ("Address of ap=%u\n", app);
Printf ("value of a through ap=%d\n", *ap);
Printf ("value of a through app=%d\n", **app);
return 0;
 }
```