**Function Pointer**

1. Function pointer points to code i.e stores address of function. Typically , function pointer stores the start of the executable code.

Void print(int a)

{

Printf(“%d”,a);

}

Int main()

{

Void (\*ptr)(int)=&print (// print);

Ptr(10);

Or

Void(\*ptr)(int);

Ptr=print;

\*Ptr(10);//ptr(10)

}

1. Why do we need extra bracket around pointer like

void (\*ptr)(int)?

Ans:- Now, function pointer without bracket will be like \*ptr(int) which means a function returning a pointer.

3)unlike normal pointers, we donot de-allocate memory using function pointers.

4)Like normal pointers , we can have array of function pointers and can be used in place of switch case:-

#include<stdio.h>

#include<stdlib.h>

int sum(int a,int b)

{ return a+b;}

//-----------------------------------------------------------------

int sub(int a,int b)

{ return a-b;}

//----------------------------------------------------------------

int multiply(int a,int b)

{ return a\*b;}

int main()

{

int (\*ptr[3])(int ,int); //={sum,sub,multiply};

ptr[0]=sum;

ptr[1]=sub;

ptr[2]=multiply;

int a=ptr[0](2,3);

int b=ptr[1](2,3);

int c=ptr[2](2,3);

printf("a:%d\nb:%d\nc:%d\n",a,b,c);}

5) A function pointer can be passed as an argument and can also be returned from function:-

#include<stdio.h>

#include<stdlib.h>

int sum(int a,int b)

{return a+b;}

int sub(int a,int b)

{return a-b;}

int multiply(int a,int b)

{return a\*b;}

int wrapper(int (\*fun)(int ,int ),int a,int b)

{return fun(a,b);}

int main()

{

int sum=wrapper(sum,2,3);

int sub=wrapper(sub,2,3);

int mul=wrapper(multiply,2,3);

printf("sum:%d\nsub:%d\nmul:%d\n",sum,sub,mul);

}