```
void func1();
int main(void)
{
        printf("Address of function main() is  : %p\n",main);
        printf("Address of function func1() is : %p\n",func1);
        return 0;
}
void func1()
{
        printf("C in depth\n");
}
```

Output:

```
Address of function main() is  : 004113C0
Address of function func1() is : 00411104
```

9.23.1 Declaring a Pointer to a Function

We have seen that functions have addresses, so we can have pointers that can contain these addresses and hence point to them. The syntax for declaration of a pointer to a function is-

```
return_type (*ptr_name)(type1,type2,......);
```

For example-

```
float (*fp)(int);
char (*func_p)(float,char);
```

Here `fp` is a pointer that can point to any function that returns a `float` value and accepts an `int` value as argument. Similarly `func_p` is a pointer that can point to functions returning `char` and accepting `float` and `char` as arguments.

We can see that this declaration is similar to the declaration of a function, except that the pointer name is preceded by a * and is enclosed in parentheses. Use of * is obvious since we did this while declaring pointers to variables also, but why is the pointer name enclosed in parentheses. Let us remove the parentheses and see.

```
float *fp(int);
```

How would you declare a function returning a pointer to float and taking an int value? Well exactly in the same manner as above. So in this declaration `fp` is declared to be a function rather than a pointer, this happened because parentheses have higher precedence than * operator. This is the reason for enclosing the pointer name inside parentheses.

Now we have learnt how to declare a pointer to a function, the next step is to assign a function's address to it.

```
float (*fp)(int,int);          /*Declaring a function pointer*/
float func(int,int);           /*Declaring a function*/
fp=func;                       /*Assign address of function func() to pointer fp*/
```

After the above assignment `fp` contains the address of function `func()`. Declaring a function is necessary before using its address anywhere in the program because without declaration the compiler will not know about this function and will generate an error.

9.23.2 Calling a Function through Function Pointer

Now let us see how to invoke a function using a function pointer.

```
r=func(a,b);          /*Calling a function in usual way*/
r=(*fp)(a,b);         /*Calling a function via function pointer*/
r=fp(a,b);            /Calling a function via function pointer, indirection operator can be
                      omitted*/
```

The effect and result of calling a function by its name or by a function pointer is exactly the same. While using the function pointer we can omit the indirection operator as we have done in the third statement, but the second statement makes it clear to the user that a function pointer is used.

```
/*P9.35 Program to invoke a function using function pointer*/
#include<stdio.h>
```

Granny Schools

Scanned by CamScanner

```
float add(int,float),result;
int main(void)
{
        float (*fp)(int,float);
        float result;

        fp=add; /*Assign address of function add() to pointer fp*/

        /*Invoking a function directly using function's name*/
        result=add(5,6.6);
        printf("%f\n",result);

        /*Invoking a function indirectly by dereferencing function pointer*/
        result=(*fp)(5,6.6);
        printf("%f\n",result);
        return 0;
}
float add(int a,float b)
{
        return a+b;
}
```

Output:

```
11.600000
11.600000
```

Granny Schools

9.23.3 Passing a function's address as an argument to other function

We can send the function's address as an argument to other function in the same way as we send other arguments. To receive the function's address as an argument, a formal parameter of the appropriate type should be declared. We can then invoke the function sent as an argument by dereferencing the formal pointer variable. The following program will make this point clear.

```
/*P9.36 Program to send a function's address as an argument to other function*/
#include<stdio.h>
void func(char,void(*fp)(float));
void fun1(float);
int main(void)
{
        printf("Function main() called\n");
        func('a',fun1);
        return 0;
}
void func(char b,void(*fp)(float))
{
        printf("Function func() called\n");
        (*fp)(8.5);
}
void fun1(float f)
{
        printf("Function fun1() called\n");
}
```

Output:

```
Function main() called
Function func() called
Function fun1() called
```

Here `func()` is a function which accepts two arguments, a `char` and a function pointer. This function pointer can point to any function that accepts a `float` and returns nothing.

`fun1()` is a function that accepts a `float` and returns nothing so we can send its address as second argument to the function `func()`. The function `main()` calls function `func()` while the function `func()` calls function `fun1()` indirectly using function pointer. Now we will write the same program and this time we will send a pointer that contains the address of function `fun1()`.

Scanned by CamScanner

```
/*P9.37 Program to pass a pointer containing function's address as an argument*/
#include<stdio.h>
void func(char,void(*fp)(float));
void fun1(float);
int main(void)
{
        void (*p)(float);
        p=fun1;

        printf("Function main() called\n");
        func('a',p);
        return 0;
}
void func(char b,void(*fp)(float))              /*Value of p stored in fp*/
{
        printf("Function func() called\n");
        (*fp)(8.5);     /*Calling fun1 indirectly using pointer*/
}
void fun1(float f)
{
        printf("Function fun1() called \n");
}
```

Granny Schools

Output:

```
Function main() called
Function func() called
Function fun1() called
```

9.23.4 Arrays of function pointers

Function pointer can be used in applications where we don't know in advance which function will be called. In that case we can take the addresses of different functions in an array and then call the appropriate function depending on some index number.

Let us take a program and understand this concept. In this program we'll add, subtract, multiply or divide two numbers depending on user's choice. For this we will make four different functions and the addresses of these functions will be stored in an array of function pointers.

```
float add(float,int);          /*Declaration of functions*/
float sub(float,int);
float mul(float,int);
float div(float,int);

float (*fp[4])(float,int);     /*Declare an array of function pointers*/

fp[0]=add;    /*Assigning address to elements of the array of function pointer*/
fp[1]=sub;
fp[2]=mul;
fp[3]=div;
```

Instead of the above assignment statements, we could have initialized the array as-

```
float (*fp[])(float,int)={add,sub,mul,div};
```

Now we can see that-

`(*fp)[0](a,b);`	is equivalent to	`add(a,b);`
`(*fp)[1](a,b);`	is equivalent to	`sub(a,b);`
`(*fp)[2](a,b);`	is equivalent to	`mul(a,b);`
`(*fp)[3](a,b);`	is equivalent to	`div(a,b);`

```
/*P9.38 Array of function pointers*/
#include<stdio.h>
float add(float,int);
float sub(float,int);
float mul(float,int);
```

Scanned by CamScanner