


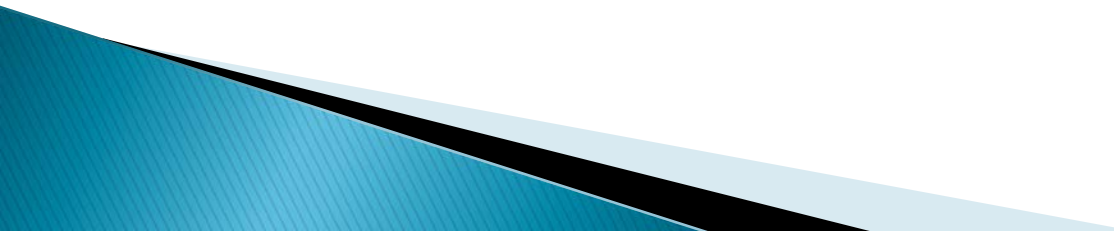
# Lecture-3

## Pointers to Functions

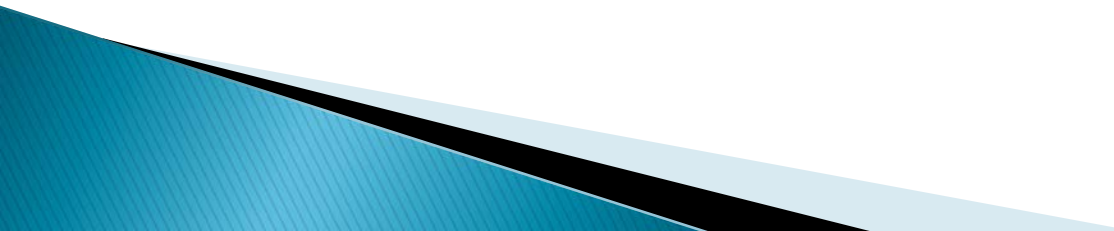
# Introduction

- ▶ While many programming languages support the concept of pointers to data, only a few enable you to define pointers to code -- that is, pointers that point to functions.
  - ▶ Originally introduced in C, pointers to functions are widely used in C++
  - ▶ Unfortunately, their cumbersome syntax baffles both novices and experienced programmers.
- 

# What are function Pointers?

- ▶ C does not require that pointers only point to data, it is possible to have pointers to functions
  - ▶ Functions occupy memory locations therefore every function has an address just like each variable
- 

# Why do we need function Pointers?

- ▶ Useful when alternative functions maybe used to perform similar tasks on data (eg sorting)
  - ▶ One common use is in passing a function as a parameter in a function call.
  - ▶ Can pass the data and the function to be used to some control function
  - ▶ Greater flexibility and better code reuse
- 

# Define a Function Pointer

- ▶ A function pointer is nothing else than a variable, it must be defined as usual.

Eg,

```
int (*funcPointer) (int, char, int);
```

funcPointer is a pointer to a function.

- ▶ The extra parentheses around (\*funcPointer) is needed because there are precedence relationships in declaration just as there are in expressions

# Assign an address to a Function Pointer

- ▶ It is optional to use the address operator & in front of the function's name
- ▶ When you mention the name of a function but are not calling it, there's nothing else you could possibly be trying to do except for generating a pointer to it
- ▶ Similar to the fact that a pointer to the first element of an array is generated automatically when an array appears in an expression

# Assign an address to a Function Pointer

```
//assign an address to the function pointer  
int (*funcPointer) (int, char, int);
```

```
int firstExample ( int a, char b, int c){  
    printf(" Welcome to the first example");  
    return a+b+c;  
}  
funcPointer= firstExample; //assignment  
funcPointer=&firstExample; //alternative  
using address operator
```

# Comparing Function Pointers

- ▶ Can use the (==) operator

//comparing function pointers

```
If (funcPointer == &firstExample)
```

```
    printf (“pointer points to firstExample”);
```



# Calling a function using a Function Pointer

- ▶ There are two alternatives
  - 1) Use the name of the function pointer
  - 2) Can explicitly dereference it

```
int (*funcPointer) (int, char, int);
```

```
// calling a function using function pointer
```

```
int answer= funcPointer (7, 'A' , 2 );
```

```
int answer=(* funcPointer) (7, 'A' , 2 );
```

# Arrays of Function Pointers

- ▶ C treats pointers to functions just like pointers to data therefore we can have arrays of pointers to functions
- ▶ This offers the possibility to select a function using an index

Eg.

suppose that we're writing a program that displays a menu of commands for the user to choose from. We can write functions that implement these commands, then store pointers to the functions in an array:

```
void (*file_cmd[]) (void) =  
{ new_cmd,  
  open_cmd,  
  close_cmd,  
  save_cmd ,  
  save_as_cmd,  
  print_cmd,  
  exit_cmd  
};
```

If the user selects a command between 0 and 6, then we can subscript the file\_cmd array to find out which function to call

```
file_cmd[n]();
```

# Trigonometric Functions

```
// prints tables showing the values of cos,sin
#include <math.h>
#include <stdio.h>
void tabulate(double (*f)(double), double first, double last, double incr);
main()
{
    double final, increment, initial;

    printf ("Enter initial value: ");
    scanf ("%lf", &initial);

    printf ("Enter final value: ");
    scanf ("%lf", &final);

    printf ("Enter increment : ");
    scanf ("%lf", &increment);

    Printf("\n  x  cos(x) \n"
           "  ----- \n");
    tabulate(cos, initial,final,increment);

    Printf("\n  x  sin (x) \n"
           "  ----- \n");
    tabulate(sin, initial,final,increment);

    return 0;
}
```

# Trigonometric Functions

```
// when passed a pointer f prints a table showing the value of f
void tabulate(double (*f) (double), double first, double last,
             double incr)
{
    double x;
    int i, num_intervals;
    num_intervals = ceil ( (last -first) /incr );
    for (i=0; i<=num_intervals; i++){
        x= first +i * incr;
        printf("%10.5f %10.5f\n", x , (*f) (x));
    }
}
```

Enter initial value: 0

Enter final value: .5

Enter increment: .1

X	cos(x)
0.00000	1.00000
0.10000	0.99500
0.20000	0.98007
0.30000	0.95534
0.40000	0.92106
0.50000	0.87758

X	sin(x)
0.00000	0.00000
0.10000	0.09983
0.20000	0.19867
0.30000	0.29552
0.40000	0.38942
0.50000	0.47943