C-Refresher: Session 08
Function Pointers

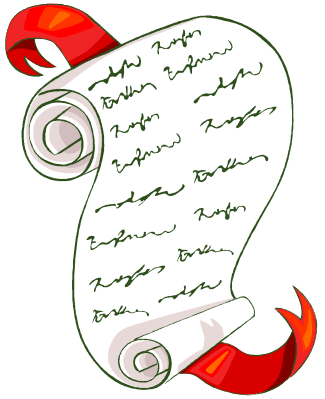
Arif Butt

Summer 2017

I am Thankful to my student Muhammad Zubair bcsf14m029@pucit.edu.pk for preparation of these slides in accordance with my video lectures at

http://www.arifbutt.me/category/c-behind-the-curtain/

Today's Agenda



- Data Pointers vs Function Pointers
- Declaring Function Pointers
- Calling a Function using Function Pointer
- Passing Function Pointer as Parameter to Function
- Array of Function Pointers
- C qsort() Library Function

Data Pointers vs Function Pointers

❑Data Pointers/Pointers to Data

- Till now, we have studied data pointers
- Data Pointers contain the address of the place in memory
- And that memory location may contain
  - a variable
  - an array of variables
  - a structure
  - an array of structures
- You may say that Data Pointers is the vanilla flavour of pointers

Data Pointers vs Function Pointers(cont…)

❑Function Pointers/Pointers to Functions

- Unlike data pointers, a function pointer points to some piece of code not data
- Typically, it stores the address of start of some executable code
- We can define it as
- "A Pointer that stores the address of a function"

Declaring Function Pointers

❑Declaring a function pointer

- **Syntax**
  - `return-type(*name)(arg-list);`
- e.g. let's suppose a function
- `void f1();`
- Its function pointer can be declared as
- `void(*fptr)();` //as `f1()` has no argument list
- Now, let's suppose another function
- `double f2(double,char);`
- Its function pointer will be like
- `double (*fptr2)(double,char);`

Declaring Function Pointers(cont…)

- We can also declare an array of function pointers
- Suppose we have the following functions
- `double* f1(int,int);`
- `f2(), f3(), f4()` are functions with the same return type and argument list as `f1()`
- Now, array of function pointers for these four functions can be declared like
- `double*(*fptr[4])(int,int);` /*so we have an array of function pointer of size `4`*/

Declaring Function Pointers(cont…)

/*It's a simple program, that passes value to a function and the function prints that value*/

```
#include<stdio.h>
void f1(int);
int main(){
  f1(56);
  return 0;
}
void f1(int n){
  printf("Value passed to function is: %d\n",n);
}
```

Declaring Function Pointers(cont…)

- Output of the above program is:

```
Value passed to function is: 56
```

- Now we will try to use function pointer for calling the function in the above program

Declaring Function Pointers(cont…)

/*Here a function pointer has been declared and then that pointer is used to call the function*/

```
#include<stdio.h>
void f1(int);
int main(){
  void(*fptr)(int)=NULL;
  fptr=f1;
  (*fptr)(56);
  return 0;
}
void f1(int n){
  printf("Value passed to function is: %d\n",n);
}
```

Declaring Function Pointers(cont…)

- Output of the above program is:

```
Value passed to function is: 56
```

- Let's discuss another example in which we will declare a function pointer that will return the value from the function

Declaring Function Pointers(cont…)

/*The program shows using function pointer returning value from the function*/

```
#include<stdio.h>
int sum(int,int);
int main(){
  int (*fptr)(int,int)=NULL;
  fptr=&sum;
  int rv=(*fptr)(5,4);
  printf("Sum is: %d\n",rv);
  return 0;
}
int sum(int a,int b){
  return a+b;}
```

Declaring Function Pointers(cont…)

- Output of the above program is:

```
Sum is: 9
```

- The value returned by the function is displayed

Passing F-Pointers to Functions

- We can pass function pointers as parameters to functions as well, which is the main power of function pointers
- Let's start with an example
- Suppose we have a function, in which three arguments are passed, first one is a function pointer and the next two arguments are integer numbers
  - `int rv=compute(add,int,int);` or
  - `int rv=compute(sub,int,int);` or
  - `int rv=compute(mul,int,int);` `/*add, sub, mul` are function pointers*/

Passing F-Pointers to Functions(cont…)

- `add(), sub() and mul()` functions have been declared which compute the sum, difference and multiplication of two numbers passed as parameters respectively
- `int add(int a,int b){return a+b;}`
- `int sub(int a,int b){return a-b;}`
- `int mul(int a,int b){return a*b;}`
- And Prototype of `compute()` is
- `int compute(int(*fptr)(int,int),int,int);`
- Now let's start with a program example to explain this

Passing F-Pointers to Functions(cont…)

/*The program calls three functions `add()`, `sub()` and `mul()` and shows the values*/

//no use of function pointers in this program

```
#include<stdio.h>
int add(int,int);
int sub(int,int);
int mul(int,int);
int main(){
  int a=15, b=10;
  int rv1=add(a,b);
  printf("%d+%d=%d\n",a,b,rv1);
  int rv2=sub(a,b);
  printf("%d-%d=%d\n",a,b,rv2);
```

Passing F-Pointers to Functions(cont…)

```
int rv3=mul(a,b);
  printf("%d*%d=%d\n",a,b,rv3);
  return 0;}
int add(int a,int b){
  return a+b;
}
int sub(int a,int b){
  return a-b;
}
int mul(int a,int b){
  return a*b;
}
```

Passing F-Pointers to Functions(cont…)

- Output of the above program is:

```
15+10=25
15-10=5
15*10=150
```

- Now let's write the above program using function pointers
- Program declares another function `compute()` and calls it by passing it function pointers of `add()`, `sub()` and `mul()`, which returns respective result

Passing F-Pointers to Functions(cont…)

```
#include<stdio.h>
int add(int,int);
int sub(int,int);
int mul(int,int);
int compute(int(*)(int,int),int,int);
int main(){
  int a=15, b=10;
  int rv1=compute(add,a,b);
  printf("%d+%d=%d\n",a,b,rv1);
  int rv2=compute(sub,a,b);
  printf("%d-%d=%d\n",a,b,rv2);
  int rv3=compute(mul,a,b);
  printf("%d*%d=%d\n",a,b,rv3);
  return 0;}
```

Passing F-Pointers to Functions(cont…)

```
int compute(int(*fptr)(int a,int b),int a,int b){
  int result=(*fptr)(a,b);
  return result;
}
int add(int a,int b){
  return a+b;
}
int sub(int a,int b){
  return a-b;
}
int mul(int a,int b){
  return a*b;
}
```

Passing F-Pointers to Functions(cont…)

- Output of the above program is:

```
15+10=25
15-10=5
15*10=150
```

Passing F-Pointers to Functions(cont…)

- These functions `add()`, `sub()` and `mul()` are called call back functions because these are the functions that are called through a function pointer
- This call back function is one of the biggest power of function pointers, and we achieve this by passing a function pointer as parameter to the function
- Function pointers can also be returned from a function, just as they can be passed to a function as parameter

Array of Function Pointers

- Array of function pointers can be used to evaluate the function on the basis of some criteria
- An array of function pointers can be declared like
- `int (*fptr_arr[3])(int,int);` //array of 3 elements
- In this array, each element is going to point to a different function but each function will have return type of `int` and will receive two `int` type numbers as arguments e.g.
  - `int f1(int,int);`
  - `int f2(int,int);`
  - `int f2(int,int);`

Array of Function Pointers(cont…)

- Now to make array elements to point to these functions, we have to write the following statements
  - `fptr_arr[0]=&f1;`
  - `fptr_arr[1]=&f2;`
  - `fptr_arr[2]=&f3;`
- Instead of first declaring the array and then separately initializing its each element, we can all do this in a single statement like
  - `int rv=(*fptr_arr[3])(int,int)={f1,f2,f3};`

Array of Function Pointers(cont…)

- Now, for example, we are to call `f1()` using `fptr_arr`, we have to use the following statement
  - `int rv=(*fptr_arr[0])(10,5);`
- To call `f2()`, we will use subscript `1`, and for `f3()`, subscript `2`
- It is the subscript value which decides that which function to call
- Let's write a program to understand array of function pointers

Array of Function Pointers(cont…)

```
#include<stdio.h>
int add(int,int);
int sub(int,int);
int mul(int,int);
int compute(int(*)(int,int),int,int);
int main(){
  int a=15, b=10;
//int (*fptr_arr[])(int,int)={add,sub,mul};
  int (*fptr_arr[3])(int,int);
  fptr_arr[0]=&add;
  fptr_arr[1]=&sub;
  fptr_arr[2]=mul;
  int ch;
```

Array of Function Pointers(cont…)

```
 printf("1 for Add\n2 for Sub\n3 for Mul\nEnter Your
choice:\n");
  scanf("%d",&ch);
 if(ch==1){
    int rv1=compute(fptr_arr[ch-1],a,b);
    printf("%d+%d=%d\n",a,b,rv1);
  }else if(ch==2){
    int rv1=compute(fptr_arr[ch-1],a,b);
    printf("%d-%d=%d\n",a,b,rv1);
  }else if(ch==3){
    int rv1=(*fptr_arr[ch-1])(a,b);
    printf("%d*%d=%d\n",a,b,rv1);
  }
```

Array of Function Pointers(cont…)

```
  else{
    printf("Wrong Option!\n");}
  return 0;}
int compute(int(*fptr)(int a,int b),int a,int b){
  int result=(*fptr)(a,b);
  return result;
}
int add(int a,int b){
  return a+b;}
int sub(int a,int b){
  return a-b;}
int mul(int a,int b){
  return a*b;}
```

Array of Function Pointers(cont…)

- Output of the above program is:

```
1 for Add
2 for Sub
3 for Mul
Enter Your choice: 1
15+10=25
```

- Another output is:

```
1 for Add
2 for Sub
3 for Mul
Enter Your choice: 3
15*10=150
```

C qsort() Library Function

- There is a C built-in function `qsort()`, that is used for sorting an array
- Array can be of any datatype, i.e. of integer type, character type, or may be of some structure type or some other datatype
- **Syntax of `qsort()`**
- `void qsort(void* base, int numofElem, int sizeOfElem,int(*func)(const void*,const void*));`
- A brief description of the parameters of `qsort()` has been provided

C qsort() Library Function(cont…)

i. base

- `base` is a pointer of type `void`, which points to the base of the array, i.e. the array name

ii. numOfElem

- It is the no. of elements in the array

iii. sizeofElem

- It is the size of each element of the array, e.g. in case of integers size is `sizeof(int)`

iv. Function Pointer

- It is basically a pointer function that points to our own written function

C qsort() Library Function(cont…)

- Our function will have return type of `int` and two parameters of type `void*` which point to constant data(note that the pointers are not constant rather the data is constant)
- The return value of the function can be of any of the following three
  - `0 =>arg1==arg2`
  - `1 =>arg1>arg2`
  - `2 =>arg1<arg2`
- Our function is basically a comparison function
- Now, let's understand this through a program example

C qsort() Library Function(cont…)

/*Program uses `qsort()` to sort an array of integers in ascending order*/

```
#include<stdio.h>
#include<stdlib.h>
int mySort(const void*,const void*);
int main(){
  int arr[]={100,20,56,29,22};
  qsort(arr,5,sizeof(int),mySort);
  for(int i=0;i<5;i++)
    printf("%d\n",arr[i]);
  return 0;}
int mySort(const void* x,const void* y){
  return *(int*)x-*(int*)y;}    /*x has been first casted to an integer pointer and then dereferenced*/
```

C qsort() Library Function(cont…)

- Output of the above program is:

```
20
22
29
56
100
```

- If we change the `mysort()` function like below, we can sort the array in descending order

```
int mySort(const void* x,const void* y){
  return *(int*)y-*(int*)x;}
```

- Now, let's write another program that uses `qsort()` to arrange an array of character of strings

C qsort() Library Function(cont…)

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
int strSort(const void*,const void*);
int main(){
  char* arr[30]={"Zeeshan","Arif
Butt","Rauf","Haris","Hadeed"};
  qsort(arr,5,sizeof(char*),strSort);
  for(int i=0;i<5;i++)
    printf("%s\n",arr[i]);
  return 0;}
int strSort(const void* x,const void* y){
  return strcmp(*(char* const*)x,*(char* const*)y);
}
```

C qsort() Library Function(cont…)

- Output of the above program is:

```
Arif Butt
Hadeed
Haris
Rauf
Zeeshan
```

- You have seen that how function pointers are helpful and how important they are in programming practices

SUMMARY