Programming with C and C++

CSC-101 (Lecture 24)

Dr. R. Balasubramanian
Professor
Department of Computer Science and Engineering
Mehta Family School of Data Science and Artificial Intelligence
Indian Institute of Technology Roorkee
Roorkee 247 667

bala@cs.iitr.ac.in
https://faculty.iitr.ac.in/cs/bala/

Void Pointer

```
1   #include<stdio.h>
2   int main()
3   {
4       float a[4]={6.1,2.3,7.8,9.0};
5       void *ptr;
6       ptr=a;
7       ptr=ptr+12;
8       printf("%f\n",*(float*)ptr);
9   }
10
```

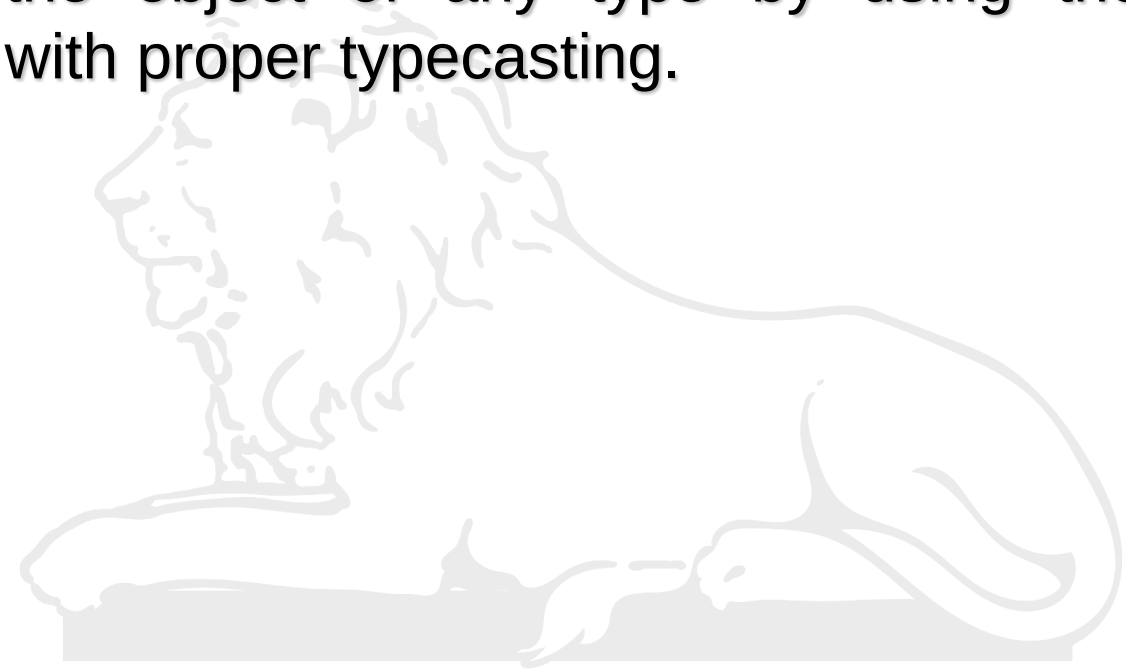
https://ideone.com/Kn90k4

stdout

```
9.000000
```

Void pointers

- We use void pointers because of its reusability.
- Void pointers can store the object of any type, and we can retrieve the object of any type by using the indirection operator with proper typecasting.



Why we use void pointers?

https://ideone.com/EkpNGp

```c
#include<stdio.h>
int main()
{
  int a=1000; // initialization of a integer variable 'a'.
  float b=10.0; // initialization of a float variable 'b'.
  char c='B'; // initialization of a char variable 'c'.
   void *ptr; // declaration of void pointer.
   // assigning the address of variable 'a'.
   ptr=&a;
   printf("value of 'a' is : %d",*((int*)ptr));
   // assigning the address of variable 'b'.
   ptr=&b;
   printf("\nvalue of 'b' is : %f",*((float*)ptr));
   // assigning the address of variable 'c'.
   ptr=&c;
    printf("\nvalue of 'c' is : %c",*((char*)ptr));
    return 0;
}

```

stdout

```
value of 'a' is : 1000
value of 'b' is : 10.000000
value of 'c' is : B
```

Pointer to Pointer (**)

```c
#include <stdio.h>

int main(void)
{ int n=80;
printf("%d\n",n);
printf("%X\n",&n);

int *pn;
pn=&n;

printf("%X\n",pn);
printf("%X\n",&pn);
printf("%d\n",*pn);

```

https://ideone.com/w8taIt

```
80
8A78BBD4
8A78BBD4
8A78BBD8
80
```

```
15  int **ppn;
16  ppn=&pn;
17  printf("%X\n",ppn);
18  printf("%X\n",&ppn);
19  printf("%X\n",*ppn);
20  printf("%d\n",**ppn);
21
22      return 0;
23  }
```

```
8A78BBD8
8A78BBE0
8A78BBD4
80
```

Dynamic Array in 2D

https://ideone.com/8cu2xb

```c
#include <stdio.h>
#include <stdlib.h>
int main() {
    int row = 3, col = 4;
    int *arr = (int *)malloc(row * col * sizeof(int));
    int i, j;
    for (i = 0; i < row; i++)
        for (j = 0; j < col; j++)
            *(arr + i*col + j) = i + j;
    printf("The matrix elements are: \n");
    for (i = 0; i < row; i++) {
        for (j = 0; j < col; j++) {
            printf("%d ", *(arr + i*col + j));
        }
        printf("\n");
    }
    free(arr);
    return 0;
}
```

```
The matrix elements are:
0 1 2 3
1 2 3 4
2 3 4 5
```

Dynamic Array in 2D

https://ideone.com/ZH7OqA

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
int r = 3, c = 4, i, j, count;
//number of rows=3 and number of columns=4
int** arr = (int**)malloc(r * sizeof(int*));
for (i = 0; i < r; i++)
arr[i] = (int*)malloc(c * sizeof(int));

// Note that arr[i][j] is same as *(*(arr+i)+j)
count = 0;
for (i = 0; i < r; i++)
for (j = 0; j < c; j++)
arr[i][j] = ++count; // OR *(*(arr+i)+j) = ++count

```

```
18  for (i = 0; i < r; i++)
19  for (j = 0; j < c; j++)
20  printf("%d ", arr[i][j]);
21
22  // free the dynamically allocated memory
23
24  for (i = 0; i < r; i++)
25  free(arr[i]);
26
27  free(arr);
28
29  return 0;
30  }
31
```

stdout

```
1 2 3 4 5 6 7 8 9 10 11 12
```

Function Pointers

```c
#include <stdio.h>

void test() {
    // This function does nothing
    return ;
}

int main() {
    int a = 5;
    // printing the address of a
    printf("Address of variable = %p\n", &a);

    // printing the address of test()
    printf("Address of a function = %p\n", test);

    // printing the address of main()
    printf("Address of a function = %p", main);
    return 0;
}
```

```
Success #stdin #stdout 0s 5532KB
Address of variable = 0x7ffe0a34d7d4
Address of a function test = 0x564dd6d721e0
Address of a function main = 0x564dd6d72060
```

https://ideone.com/4tcpuE

Syntax of function pointer

- `return type (*ptr_name)(type1, type2…);`
- **For example:**
- `int (*ip) (int);`
- `float (*fp) (int , int);` // Declaration of a function //pointer.

  `float func( int , int );` // Declaration of function.

  `fp = func;` // Assigning address of func to the fp pointer

Function Pointers

https://ideone.com/1BFXjq

```c
#include<stdio.h>

// function declaration
int areaRectangle(int, int);

int main() {
    int length, breadth, area;

    // function pointer declaration
    // note that our pointer declaration has identical
    // arguments as the function it will point to
    int (*fp)(int, int);

    printf("Enter length and breadth of a rectangle\n");
    scanf("%d%d", &length, &breadth);

```

Function Pointers

```
17      // pointing the pointer to functions memory address
18      fp = areaRectangle;
19
20      // calling the function using function pointer
21      area = (*fp)(length, breadth);
22
23      printf("Area of rectangle = %d", area);
24      return 0;
25  }
26
27  // function definition
28  int areaRectangle(int l, int b) {
29      int area_of_rectangle = l * b;
30      return area_of_rectangle;
31  }
```

Function Pointers

stdin

```
10 5
```

stdout

```
Enter length and breadth of a rectangle
Area of rectangle = 50
```

Function Pointers

https://ideone.com/hlsuXW

```c
#include <stdio.h>

int* increment(int a) {
    int *b = (int*)malloc(sizeof(int));
    // Allocate memory for an int
    if (b == NULL) {
        printf("Memory allocation failed.");
        return 1;
    }

    *b = a;
    (*b)++; // Increment the value

    return b; // Return pointer to the incremented value
}

```

Function Pointers

```
17  int main() {
18      int num = 44;
19
20      int *b = increment(num);
21      printf("Incremented value = %d", *b);
22
23      free(b); // Free the dynamically allocated memory
24
25      return 0;
26  }
27
```

stdout

```
Incremented value = 45
```

Thank You!