# C++

A S R Pavan
Scientist ‘B’
NIELIT Calicut

# Topics to be discussed

- Pointer
  - What is a pointer?
  - Declaration & Initialization
  - Dereferencing a pointer
  - Passing and returning pointers to a function
  - Pointer arithmetics
  - Pointer to pointer
  - Const and pointers
  - Pointer pitfalls
- Reference

# What is a pointer?

- Pointer is a variable which holds the address of another variable or function
- Both pointers and which it refers to should be of same datattype
- Can access specific address in memory
- Can allocate memory dynamically on the heap or free store

# Declaration

- data_type *pointer_name  or data_type* pointer_name;

Eg:

```
int *int_ptr;  or int* int_ptr;
double *double_ptr;
char * char_ptr;
string *str_ptr;
```

# Initialization

- Always Initialize pointers
- Unintialized pointers contain garbage value
  - It can point to anywhere
- If you don’t initialize a pointer to a address of variable then initialize to NULL or nullptr
  - Eg: int *ptr = NULL; // {nullptr} // value of ptr is 0
  - int *ptr; // garbage value
  - int i=10; float f=2.34; int *ptr = &i; // valid

    ptr = &f; // compiler error

# Dereferencing a pointer

- Accessing or manipulating the data where pointer pointing to

Eg:

```
int value = 15;
int *ptr = &value;
*ptr = 25; // assigning 25 to value
```

# Passing & returning pointers to a func.

- Function can even return a pointer like int, float or other datatype
- Eg: int *sum(int *x, int *y); // definition // returns a pointer

```
int *p;
p=sum(&a, &b);

int *sum(int *x, int *y){
        .....................
        return &total;
}
```

# Dynamic memory allocation

- We can allocate storage for a variable at run time in heap
- We can use pointers to access newly allocated heap storage
- Allocation of storage at run time using **new**
- Eg:  int *ptr = new int; *ptr = 150;
- deallocation of storage at run time using **delete**
- Eg: delete ptr;

# Pointer arithmetics

- Pointers can be used in
  - Assignment expressions
  - Arithmetic expressions
  - Comparison expressions
- C++ allows pointer arithmetic
- Eg: int val[] = {10,20,30};

  int *ptr; ptr = val; // ptr = &val[0];  // both are same

  ptr++; // refers to val[1]

# Pointer to pointer

- Holds the address of a pointer
- Declaration : data_type  **pptr;
- Eg: int val = 10;

  int *ptr;

  int **pptr;

  ptr = &val;

  pptr = &ptr;
- *ptr, **ptr all locates the same variable val

# Const and pointers

- Several ways to qualify pointers using const
  - Pointers to constants
    - Data is constant, can assign address to pointer
    - Eg: const int *ptr {&marks}; // marks is int datatype
  - Constant pointers
    - pointer is constant by can change data
    - Eg: int *const ptr {&marks}; // marks is int datatype
  - Constant pointers to constants
    - Both pointer and data are constant
    - Eg: const int *const ptr {&marks}; // marks is int datatype

# Pointer pitfalls

- Uninitialized pointers
- Dangling pointers
- Leaking memory

# References

- Reference is alias to anpther existing variable
- Must be initialized to a variable when declared
- Cannot be null
- Once initialized cannot be made to refer to a different variable

# Doubts

Q&A

# End of the session

Thank You