POINTERS

Pointers in C are a powerful feature that allows you to directly access memory and manipulate data.

KEY POINTS

- Pointers are variables that store the **memory address** of another variable.
- Execution time is fast.
- Declaration : *
  - e.g. `int *p;`
  - the star can be anywhere, and doesn’t have to be closer to the variable name.
  - to declare 2 pointers `int *ptr1, *ptr2`
- You can get the memory address of a variable with the & operator.
  - e.g. `p = &x;` sets the pointer p to point to the variable x.

```
#include <stdio.h>

int main() {
    int * ptr;
    int var;

    var = 5;

    printf("Address of var is: %p\\n",&var);
    ptr = &var;

    printf("Pointer ptr is pointer to: %p", ptr);

    return 0;
}
```

- You can access the value that a pointer points to with the * operator.
    - e.g. `p` gives you the value of the variable that p points to.
    - AKA de-referencing of pointers

```
#include <stdio.h>

int main() {
    int * ptr, a;
    a = 5;
    ptr = &a;

    printf("Address of variable a is: %p\\n",&a);
    printf("Address where ptr points to is: %p\\n",ptr);

    // Notice the use of (*) before ptr here
    printf("Value at the address ptr points to is: %d
\\n", *ptr);

    // Update the value pointed by pointers
    printf("The value of var was: %d\\n",a);
    *ptr = 1;

    printf("The value of *ptr is updated to: %d\\n", a);
    printf("The value of *ptr is updated to: %d\\n", *pt
r);
```

```
    return 0;
}
```

- Pointers can be incremented and decremented, pointing to positions forward and backwards in an array.
- Pointers are used for dynamic memory allocation, as well as for function pointers and creating complex data structures like linked lists and trees.

TYPES OF POINTERS

Point to derived data types such as arrays
Point to primitive data types such as int
Point to user-defined data types such as arrays

- **Integer Pointer:**
  - `int *ptr`
  - nothing but a pointer that points to the integer value
- **Function Pointer:**
  - `int func(char,float)`
  - `int (* ptr)(char, float)`
  - `ptr = &func;`
  - Are actually little different from the normal pointers, as normal pointers points to the data but instead function pointer points to the code block.
- **Null Pointer:**
  - `int *ptr = NULL;`
  - They don't point to any memory location
  - They are created by assigning the NULL values to the pointers.
  - Pointer of any type can be assigned the NULL value

    - Not a good practice
- **Void Pointer:**
    - `void *ptr;`
    - AKA Generic Pointers
        - as they can point to any type of data and can be typeCasted(converting 1 datatype to another) to any type
    - Void pointers in C are the pointers of type Void.
        - i.e. they don't have any associated data type.
    - The void pointers can't be de-referenced
- **Wild Pointer:**
    - `<data type> *ptr;`
    - A pointer that has not been initialized to anything yet (not even NULL).
    - These pointers may point to some arbitrary memory location
    - Can cause a program to crash or behave badly.
    - Not recommended to use
- **Constant Pointers:**
    - `<data type> * const ptr_name`
    - The memory address stored inside the pointer is constant
    - They can not be modified once defined.
    - It will always point to the same memory address
- **Pointer to Constant:**
    - `const int *ptr_name`
    - The pointer pointing to a constant value that can't be modified.
    - Here we can only access the data pointed by the pointer, but can't modify it.
    - Although we can change the address stored in the pointer to constant.

- **Dangling Pointer:**
    - A pointer pointing to a memory location that has been deleted(freed).