POINTER

In computer programming, a pointer is a variable that stores the memory address of another variable. Pointers are commonly used in C and C++, but are also present in other programming languages such as C# and Python.

The memory address of a variable is a unique number that identifies the location of the variable's value in the computer's memory. By storing the memory address of a variable in a pointer, the programmer can use the pointer to access the value of the variable, even if the variable's name is not known. This makes pointers a powerful tool for working with memory and creating more complex data structures.

For example, a pointer to an integer variable can be declared as follows in C:

```
int x = 5;

int *ptr; // Declare a pointer to an integer

ptr = &x; // Assign the address of x to the pointer
```

Now, the **ptr** variable contains the memory address of the x variable. To access the value stored in x, you can use the **dereference operator (*),** like this:

```
printf("The value of x is %d\n", *ptr); // Output: "The value of x is 5"
```

Pointers can also be used to dynamically allocate memory during runtime, which can be useful for creating large data structures like arrays or linked lists. Pointers are also useful for implementing various data structure such as tree, linked list and graph. Pointers are also used for passing the arguments to function by reference instead of by value.

It is important to note that working with pointers can also be dangerous, if not handled properly, it can cause segmentation fault or memory leak and other issues in the program.

what are the usages of pointer in c

1. Dynamic memory allocation: Pointers can be used to dynamically allocate memory during runtime, which is useful for creating large data structures like arrays or linked lists.
2. Array and pointer manipulation: Pointers can be used to manipulate arrays and other data structures in C, such as strings. This allows for more efficient manipulation of large data sets and provides a way to implement certain data structures, such as dynamic arrays and linked lists.
3. Pass by reference: Pointers can be used to pass arguments to a function by reference instead of by value. This allows a function to modify the value of a variable that is passed to it, rather than just using a copy of the variable's value. This can be useful for situations where a function needs to change the value of a variable that is defined in the main program.
4.

Pointer to Pointer

A pointer to a pointer is a type of pointer that holds the memory address of another pointer. In other words, it's a variable that stores the address of another variable that stores the address of a value.

Here is an example that illustrates the concept of a pointer to a pointer in C:

```
#include <stdio.h>

int main()

{

   int x = 5;

   int *ptr1;

   int **ptr2; // Declare a pointer to a pointer

   ptr1 = &x;  // Assign the address of x to ptr1

   ptr2 = &ptr1; // Assign the address of ptr1 to ptr2

   printf("The value of x is %d\n", **ptr2); // Output: "The value of x is 5"

   return 0;

}
```

In this example, x is an integer variable with the value 5. ptr1 is a pointer to an integer that stores the memory address of x. ptr2 is a pointer to a pointer that stores the memory address of ptr1. The * operator is used to dereference the pointers and access the value stored in x. By dereferencing ptr2, we can access the value of x through ptr1.

Here is how it works:

x: variable holding 5

ptr1: variable holding the memory address of x

ptr2: variable holding the memory address of ptr1

So when we dereference ptr2, we first get to the memory address stored in ptr1 and then by dereferencing ptr1 we can get the value stored in x.

Pointers to pointers are often used in situations where a function needs to return multiple values or where a pointer needs to be passed as a function argument. For example, a function that returns the address of a newly allocated block of memory might be implemented using a pointer to a pointer. This can be useful in dynamic memory allocation, where the address of a dynamically allocated block of memory needs to be passed back to the calling function.

Explain Null Pointer

A null pointer is a special value that is assigned to a pointer variable when the pointer is not pointing to a valid memory location. A null pointer represents the absence of a value or the lack of an object. It is used to indicate that a pointer does not point to a valid location in memory.

In most programming languages, a null pointer is represented by the keyword "null," "None," "nil," "nullptr," or a special constant such as 0 or -1.

For example, consider the following C++ code:

```
int *p = NULL;
```

In this example, the pointer p is assigned the value NULL, which indicates that it is a null pointer and does not point to a valid location in memory.

A common error that can occur when using pointers is dereferencing a null pointer, which occurs when you try to access the data stored at the memory location pointed to by a null pointer. This can cause a program to crash or produce unexpected results, and it is often referred to as a null pointer exception or a segmentation fault.

For example, consider the following C code:

```
int *p = NULL;

*p = 5; // Dereferencing a null pointer, this will cause program crash
```

The first line initialize p as a null pointer, and then it is trying to access memory location which is not exist and leads to crash.

To avoid such problem, developer can check whether pointer is null or not before trying to dereference it.

```
int *p = NULL;

if (p != NULL)

   *p = 5;
```

In modern programming languages like python and java, null pointer exception is handled by its own mechanism, and it will throw an exception when trying to access memory location via null pointer.

Explain Void Pointer

A void pointer, also known as a generic pointer, is a special type of pointer that can point to any type of data. In C and C++, a void pointer is declared using the keyword "void" and is represented by a variable of type "void *". Because a void pointer can point to any type of data, it can be used to pass a pointer of any type to a function or to store a pointer of any type in a data structure.

Here's an example of a function that takes a void pointer as an argument:

```
void printData(void *ptr) {

   printf("Data: %d\n", *(int *)ptr);

}
```

In the example above, the printData function takes a single argument of type void *, which is a pointer to an unspecified type of data. The function then casts the void pointer to a pointer of type int * using the (int *) type cast operator. This allows the function to dereference the void pointer and access the data it points to, which is assumed to be an integer.

Here's how you might use this function:

```
int value = 5;

printData(&value); // Outputs: "Data: 5"
```

In the example above, the address of the value integer is passed as an argument to the printData function. Because a void pointer can point to any type of data, the function is able to accept this argument even though the pointer is of type int * rather than void *.

It's worth noting that, unlike null pointer, void pointer has a memory location it pointing to, but it not specific about the type of data it's pointing to.

It requires explicit type casting in order to access the data.

It is mainly used for following things:

1. To pass a pointer to a function that does not know the type of data to which the pointer points.
2. To create generic data structures that can store pointers to different types of data.
3. To store pointer to variable with different data types in same array.

Explain Dangling Pointer

A dangling pointer is a pointer that refers to a memory location that is no longer valid. This can happen when a memory location that was previously allocated using the heap (dynamic memory allocation) is deallocated, but one or more pointers to that memory location still exist.

For example, consider the following C code:

```
int *p = new int; // Allocate memory on the heap

*p = 5;

delete p; // Deallocate memory
```

In this example, the pointer p is initially set to point to a block of memory that was dynamically allocated on the heap using the new operator. The value 5 is then stored at that memory location. Later, the memory is deallocated using the delete operator.

However, p still points to that memory location which is now invalid, leading to undefined behaviour if the pointer is accessed again. This is a classic example of a dangling pointer.

Another way is when a pointer is pointing to a local variable inside a function and the variable goes out of scope after the function call, pointer will point to memory location which is not in use anymore, this is also another example of dangling pointer.

```
void func() {

   int x = 5;

   int *p = &x;

   // code..

}

int main() {

   func();

   printf(c); // this will produce undefined behaviour as x is out of scope and memory location
is not available anymore

}
```

In the above example, variable x is created inside func and p points to that variable, as soon as func is returned, x goes out of scope, and memory location is deallocated but p still pointing to that memory location, so accessing it will result in undefined behaviour.

Dangling pointers can cause serious problems in a program, such as memory leaks, crashes, or unexpected results, so it is important to be aware of them and take steps to avoid them. To avoid such scenarios, the pointer should be set to nullptr after deallocating the memory or reinitialize them to point to valid memory location.

What is the size of a generic pointer?

Ans: For a system of 16-bit, the size of a generic pointer is 2 bytes. If the system is 32-bit, the size of a generic pointer is 4 bytes. If the system is 64-bit, the size of a generic pointer is 8 bytes.

What is a dangling pointer in C?

Ans: A pointer pointing to a non-existing memory location is a dangling pointer. A dangling pointer is a pointer that has a value (not NULL) which refers to some memory that is not valid or does not exists.

What is a generic pointer? When can we use a generic pointer?

Ans: When a variable is declared void type it is known as a generic pointer. It is a pointer that can point to any data. They are used when we want to return such a pointer which applies to all types of pointers. They can also be used to increase the re-usability of the pointer.

What is a wild pointer?

Ans: A pointer that is not initialized properly before its first use is known as the wild pointer. They are called so because the uninitialized pointer's behavior is undefined as it may point to some arbitrary location that can cause the program to crash. Generally, compilers warn about the wild pointer.

Explain near, far, and huge pointers?

Ans:

A far pointer is of size 32 bit, which includes a segment selector, making it possible to point the addresses outside of the default segment.

Near pointer is utilized for storing 16-bit addresses means within the current segment on a 16-bit machine. The drawback is that we can only access 64kb of data each time.

The huge pointer is also 32-bit and can access outside segments. In the far pointer, Huge can be changed but the segment part cannot be changed.

What is the difference between a null pointer and an uninitialized pointer?

Ans: An uninitialized pointer is one that points to a memory location that isn't known. If we will try to dereference the uninitialized pointer, code behavior will be undefined.

According to C standard, an integer constant expression having the value 0, or such an expression cast to type void *, is known as a null pointer constant. If you will try to dereference the null pointer then your code will crash.

Explain the meaning of the below declarations?

1. const int ptr;

2. const int *ptr;

3. int * const ptr;

4. int const * a const;

Ans: Here,

The “ptr” is a constant integer.

Here “ptr” is a pointer to a const integer, the value of the integer is not modifiable, but the pointer is not modifiable.

Here “ptr” is a constant pointer to an integer which means that the value of the pointed integer is changeable, but the pointer is not modifiable.

Here “a” is a const pointer to a const integer which means the value of the pointed integer and pointer both cannot be changed.

What is Dereference or Indirection Operator (*)?

Ans: Dereference operator is a unary operator that is used in the declaration of the pointer and accesses a value indirectly, through a pointer. The operand of such operator should be a pointer and the result of the operation is value addressed by the operand (pointer).

For example,

int *iPointer; // Use of indirection operator in the declaration of pointer

a = *iPointer; //Use of indirection operator to read the value of the address pointed by the pointer

***iPointer = a; //Use of indirection operator to write the value to the address pointed by pointer.**

Explain Array of Pointer

An array of pointers is a data structure that stores a collection of pointers to other objects, variables or memory locations. In C and C++, an array of pointers is declared by appending an empty set of square brackets after the type of pointer being used.

For example, an array of pointers to integers would be declared as:

int *array[10];

This creates an array of 10 elements, where each element is a pointer to an integer. Each element in the array can be accessed using an index, just like a regular array.

Here's an example that shows how you might use an array of pointers to store the addresses of several integers:

```
int x = 5, y = 10, z = 15;

int *array[3] = {&x, &y, &z};
```

In this example, an array of 3 elements is created and each element is assigned the address of the variables x, y, and z respectively.

You can also initialize an array of pointers as null and later initialize them at runtime.

```
int *array[10] = {nullptr};

for(int i = 0; i< 10; i++)

    array[i] = new int;
```

You can also use them to store the address of object of different class or structs.

```
struct data {

  int a;

  float b;

};

data *p[5];

for(int i=0;i<5;i++)

  p[i]=new data;
```

Using an array of pointers allows you to store and manipulate collections of dynamically allocated objects or data structures with a single variable. It can also be used as a two-dimensional array where it uses a single pointer to point to an array of pointers.