Lecture 5 Pointers

1. Variable properties

A variable is used to represent a data value of certain type in programming. It has properties of name, memory block, address, value.

Int x=10;

X is the variable name

&x represents the address of the first memory cell of the memory block, called reference of x

\*(&x) or x represents the value of the variable called dereference of x

2. Concept of pointers

- a pointer is a variable that stores the memory address of another variable

Declaring a pointer:

Data\_type \*ptr; //declare pointer ptr

Data\_type x;

Ptr = &x; // assign address of x to ptr

* The \* informs the compiler that ptr is a pointer type variable and data\_type specifies that it will store the address of a data\_type *type* variable
* Ptr “points to x” or points to the memory location of x
* Dereferencing a pointer is to get the value stored at the memory location that the pointer is pointing to
* \* is the dereference pointer
* \*ptr gets the value at the memory address of ptr
* \*ptr = value; assigns value to memory location of address of ptr

Example

Int x = 10;

Int \*p;

P = &x; //p is a pointer, holding the address of variable x, pointing to x

Int b = \*p //\*p is the dereference of p, that gets the value stored in memory location pointed by p. Now

B has value 10

\*p = 20; //set value 20 to memory location pointed by p

Example

Int \*p;

\*p = 10; // this is not correct, because p does not point to any address

**Pointer is a tool to access data**

1. Pointers provide an alternative method to access memory block and the value stored at the memory block
2. Pointers make it possible to do address operations, so as to access data for data operations
3. The value (i.e. address values) of pointers are meaningful only at runtime.

**Variable**

Int x = 10;

&x represents address of x

X represents the value stored at &x

**Pointer**

Int \*p; //p is declared to be an int pointer

P = &x; //p holds address of x

\*p represents the value stored at p, i.e. x

Int b= \*p; //b will have value 10

\*p = 20; // x will have value 20

**Pointer operations**

1. Dereference operation

Int num1 = 2, num2 =3, sum=0,mul=0,div=1;

Int \*ptr1, \*ptr2;

Ptr1 = &num1;ptr2 = &num2;

Sum = \*ptr1+\*ptr2;

Mul = sum\*ptr1;

//sum =5 and mul=10

1. Comparisons

Value of a pointer is an address of integer value. We can compare pointers by using relational operators in the expressions for example.

P1>p2, p1==p2, p1!=p2 are all valid in C

1. Add and minus operations

We can add an integer value c to a pointer, it increases the address by sizef(data\_type)\*c

Example

Int x, \*p = &x; p=p+2

P’s value will be increased by 2\*4 =8 ex. If p’s value is 1000, then after p=p+2; p’s value becomes 1008

1. Special pointers

* A null pointer is a special pointer value 0, means it doesn’t point to anywhere
* To declare a null pointer you may use the macro NULL (NULL is defined by preprocessor as 0)

Int \*ptr = NULL;

* In programming, if we don’t want a pointer to point to a memory location, we set it to NULL

**Generic Pointers**

* A generic pointer is a pointer that has void as its data type
* The generic pointer can be pointed at variable of ANY data type
* It is declared by syntax:

Void \*ptr;

* You need to cast a void pointer to the type of pointer before using it in reference

Int a = 10;

Void \*ptr = &a;

Printf(“%d”, \*(int \*)ptr);

* Generic pointers are used when a pointer has to point to different types of variables at different times

**Pointer to pointer**

* A pointer to pointer is a pointer pointing to a pointer of the same type. To declare a pointer to pointer just add an asterisk to a pointer

Example

Int x=10;

Int \*p, \*\*pp; //pp is a pointer to pointer

P=&x;

Pp=&p;

Now if we write,

Printf(“%d”,\*\*pp); //this prints 10

**Function type pointer**

Int max(int x, int b) {return x<y?y:x;}

Int (\*p)();//declare a function pointer

P=max //let p point to function max, p holds the address of function max

Int c = (\*p)(2,3);//use function pointer p to call the function equivalent to int c=max(2,3);

**Applications of pointers**

1. Pointers are used in pass-by-references. This enables to output multiple values from a function through function arguments
2. Pointer increases the efficiency of accessing data for algorithms
3. Pointers enable dynamic memory allocation and management
4. Pointers are used to implement complex data structures like linked lists, queues, stacks, trees.

**Memory allocations in C**

1. Static memory allocation – for global and static variables, variables instanced with absolute address in the data region
2. Automatic memory allocation – the memory block is put in the stack region where the function is called
3. Dynamic memory allocation – the allocation done by stdlib function malloc(), the memory block of a dynamic allocation is located in the heap region, and it wont be released automatically after the calling function finishes. Dynamically allocated memory blocks can be shared by differed functions. When a dynamically allocated memory block is not used anymore, it needs to use free() function to release the memory block.