Lecture 9

Pointers in C/C++ languages

In C/C++ languages **pointers** (or pointer variables) are special variables used for storing (holding) memory addresses of other variables. Being a memory address, the value of a pointer variable points or refers to the memory location of another variable so, that we have the possibility of accessing to the different variables (memory locations) by using pointers, still having directly accessing to these memory locations using names of corresponding variables. Pointers are often used also for accessing to the elements of an array, for interchanging data between two functions and for dynamic memory allocation in C/C++ languages.

There are different data types and kinds of pointers. For each data type defined in C/C++ languages there is corresponding pointer data type. What is way any pointer data type is considered as a derived data type from the first one. For example, an integer data type pointer (or pointer to integer data type) is used for holding addresses of integer type variables and float pointer is used for holding addresses of float type variables and so on. In C/C++ languages are also used different kinds of pointers such as: pointer to pointer, arrays of pointers, pointer to function, void type pointer and others.

Pointer operators & and *

For pointer processing are used two pointer operators. The first operator is the 'address of 'operator, also called 'reference' operator and notated by **&** (ampersand character). We just used it before in function scanf( ) for input operation from keyboard. The second operator is named the 'dereferencing' operator and notated by * (asterisk character). Both operators are unary operators having only one operand..

The address of operator **&** is used for taking (determining) address of a variable.

Ex.: int a; float b; scanf("%d%f ", &a, &b); In this example expressions **&a** and **&b** represent the memory addresses of variables **a** and **b** being introduced from keyboard**.**

The dereferencing operator * is used for accessing to the variable by the pointer(address) to this variable.

Ex.: Suppose **p** is the name of the pointer variable which holds the address of another variable **a.** So the expression ***p** represents the value of variable **a** pointed by **p.** And the expression ***p** is used as another name of variable **a**.

Declaring pointers

Declaration of a pointer is similar to a regular variable declaration and defers only by using the asterisk * character, as follows:

type * name;

Where **type** is any type defined in C/C++ languages and **name** is any possible name (identifier) for pointer variable and spaces before and after the asterisk character are not important.

Ex.:

int * **pi;** // declaration of one integer pointer having name **pi ,** it is said that the type of pointer **pi** is **int** * // and expression ***pi** represents the variable (value) of type **int** pointed by pointer **pi ;**

float ***pf,** ***a,** ***fptr;** // declaration of three float pointers having names **pf**, **a, fptr** of type **float** * and // expressions ***pf,** ***a**, ***fptr** are the variables (values) of type **float** pointed by pointers **pf, a** and **fptr** ;

char ***cptr;** // declaration of character pointer having name **cptr** of type **char** * and expression // ***cptr** is the variable (value) of type **char** pointed by pointer **cptr ;**

double* **dp, *A[10];** // declaration of one double type pointer having name **dp** and an array with name **A** // of ten double type pointers (type **double** *)

void * **p;** // declaration of one void type pointer (type **void** *) having name **p.**

int ** **b;** // declaration of one integer pointer to pointer named **b** of type **int**** and expression ** **b** is the variable (value) of type **int** pointed by pointer to pointer **b.**

Pointer initialization. NULL pointer

As with regular (ordinary) variable pointer variable can be initialized during declaration by assigning to it the address of another variable of corresponding type declared before.

Ex.: int a; // declaration of regular variable of **int** type having name **a;**

int* **p = &a;** // declaration of pointer variable of **int*** type having name **p** and its initialization by // the address of integer variable **a** declared before**.**

We can also initialize a pointer in run-time by assigning to it the address of corresponding data type variable declared before and so will obtain the equivalent result:

Ex.: int a;

int* **p;**

p= &a;

It is possible and it is recommended to initialize pointers by **NULL** pointer value which indicates that it is not pointing to any valid address. **NULL** pointer value can be assigned to the pointer of any type.

Ex.: a) **int** * **p=NULL;** or b) **int** * **p**; **p=NULL;**

Pointers and Arrays

In C/C++ languages the name (identifier) of an array is equivalent to the address of its first element ,so in fact they are the same concept. For example, supposing these two declarations: **int A[20]; int** * **p;**

The following assignment operations would be valid and equivalent: a) **p=A;** b) **p=&A[0];**

After that, **p** and **A** would be the same and would have the same properties so, that name of pointer **p** can be used as another name of array **A**. The only difference is that we could change the value of pointer **p** by another one, whereas **A** will always point to the first of the 20 elements of type int with which it was defined. Therefore, unlike **p,** which is an ordinary pointer, **A** is an array, and an array can be considered a *constant pointer*.

Therefore, the following assignments would not be valid: **A=p; A=A+1**;

Because **A** is an array, so it operates as a constant pointer, and we cannot assign values to constants. Concerning 1-D arrays we used square brackets **[ ]** in order to specify the index (subscript) of an element of the 1-D array to which we wanted to refer. Well, this square brackets operator **[ ]** or the indexation operator is also a dereferencing operator known as *offset operator*. It dereferences the variable (name of array or pointer) it follows just as the dereferencing pointer operator * does after adding the subscript to the name of array and thus obtaining the memory address of the corresponding element.

So, the expression **A[5]** is equivalent to the expression * **(A+5)** both represents the value of the 6-th element of int type counted from the starting address **A**. Analogically, expressions **&A[5]** and **(A+5)** are equivalent and both represent the address of the memory where this element is stored.

Lecturer: Mihail Kulev, associate professor