Programming Languages 1
Lesson 5 - Guide

Pointers

A variable has name, value, type and address. Address is an identifier of memory cells in RAM. It is used to refer to memory a field (without name). Formally an address is a not negative integer, a kind of sequential number of a cell. Memory bytes can be addressed separately (bits cannot be addressed).

Pointer (sometimes called reference) is a variable that can store the memory address of another variable. It refers/points to other variables/values. Type of the pointer is important, it means the type of the pointed variable. Pointers also have to be declared. The declaration instruction contains an asterisk (`*`) to express it is a pointer. Examples:

```
int *abc;  // abc is a pointer to an integer (store address of int)
float *def;// def is a pointer to a real (float) variable
char *ghi; // ghi can refer to a character variable
```

There is a special pointer value: `NULL`. The `int *X=NULL;` instruction means that the `X` pointer does not refer to any variable (so its value is not a real address).

There are two useful address operators. One of them is the address operator denoted by the `&` (and) symbol. It has only one operand, a variable and it provides the memory address of the given variable, which can be stored in a pointer. For instance:

```
int v=123;
int *p=&v; // declaration and initialization of a pointer
```

Here the `p` pointer will point/refer to the `v` variable, thus `p` contains the RAM address of `v`, where the `123` is stored.

The other address operator is the dereference operator denoted by the * (asterisk) symbol. It has only one operand, a pointer and it provides the value which is pointed by the given operator. For instance (using the previous 2 declarations):

```
int c= *p + 10;
```

Here the integer variable `c` will contain the sum of constant `10` and the value pointed by `p` (currently the value of `v`, so `123`).

Pointers are also variables, so it is possible to refer to the address of a pointer. This is the indirect referencing.

```
int **DP=&p; // indirect pointer points to a simple pointer
int d= **DP * 2;// *DP -> p and *p -> v thus **DP -> v
```

In this example `p` contains the address of `v` variable and `DP` contains the address of `p` pointer, thus the double of `123` will be assigned to `d`.

Parameter passing by address

Pointers are often used together with parameter passing by (pseudo)address. Usually we use parameter passing by value, where the value of the actual parameter is copied into the formal parameter. It is a one-way communication (from the caller to the callee). Sometimes two-way information transmission is necessary between the caller and the callee subroutine vice-versa. Due to the parameter passing by address the return value can be replaced by a special parameter and more the one value can be “pushed” back to the caller. Here you can see an example:

```
void circle(float R, float *P, float *A){
   /* procedure to calculate the perimeter and area of circles */
   *P = 2*R*3.14;
   *A = R*R*3.14;
}
int main(){
   float radius, perimeter, area, ratio;
   radius = 1.5;
   circle(radius, &perimeter, &area);
   ratio = area/perimeter;
   …
}
```

The `circle` procedure has 3 parameters, a float (parameter passing by value, input parameter) and two float pointers (parameter passing by address, output parameter). In the main program unit the procedure is invoked/called. The value of the `radius` variable (`1.5`) is copied into the local variable `R`. The values of the second and third actual parameters are the addresses of `perimeter` and `area` variables. They are copied into the `P` and `A` pointers, respectively. In the body of the procedure the required quantities are calculated using the value of `R` and then the results are assigned to the variables pointed by `P` and A. Since `P` contains the address of `perimeter` and `A` refers to the `area` variable, the values are saved into the variables of the `main` function. (The name of these variables are not available within the procedure that is why the pointers are applied.) After the `circle` procedure is terminated the `perimeter` and `area` variables contain the desired values (however earlier they were uninitialized in the `main`), so we can use them for further calculation/assignment.

It was used in the `scanf` function, when not the value of a variable is passed but its address (where the input value will be stored).

```
scanf(”%d”, &x );
```

Array as a parameter

Passing optional size of an array as a parameter is available by pseudo-address parameter passing. Since in C, we do not know the size of a parameter array, it must be passed as a separate parameter.

```
double sum(double *T, int N){ // Sum of N elements from address T

   int i;

   double s=0.0;

   for(i=0;i<N;i++)

      s+=T[i];

   return s;

   }

...

double total1,total2;

double A[8]={1.2, 2.3, 3.4, 4.5, 5.6, 6.7, 7.8, 8.9};
```

```
total1 = sum(A,8); // sum of elements from A[0] to A[7]

total2 = sum(&A[2],3); // sum of elements from A[2] to A[4]
```

As one can see pointers (`T`) can handle as an array (`T[i]`).

It is important to mention, that the elements of the parameters array are not passed, just the starting address. There is no local copy of the array, the subroutine works on the memory area of the caller.

Further readings:

- Brian W. Kernighan, Dennis M. Ritchie: *The C programming language*, Prentece Hall (2012)
- https://www.tutorialspoint.com/cprogramming/c_pointers.htm
- http://www.cs.fsu.edu/~myers/cgs4406/notes/pointers.html