Engineering program development 6

Edited by Péter Vass

Pointers

Variables

When we define a variable with its identifier (name) and type in the source code, it will result the reservation of some memory space for the given variable.

The size of reserved memory space are measured in bytes and depends on the type of the variable (e.g. char → 1 byte, int → 4 bytes).

The value of a variable is stored in the memory space reserved for it. So, we can say that a value is assigned to a variable.

Of course the value of a variable may change during the execution of different computational steps.

But the position of the memory space reserved for the variable inside the whole range of memory allocated for the program cannot be changed.

Pointers

Variables

The structure of a range of memory allocated for a program is similar to the a sequence of bins.

The size of each bin is one byte (8 bits).

The position of each bin in the range of memory is identified by a unique memory address.

Generally more than one byte is allocated for a given variable.

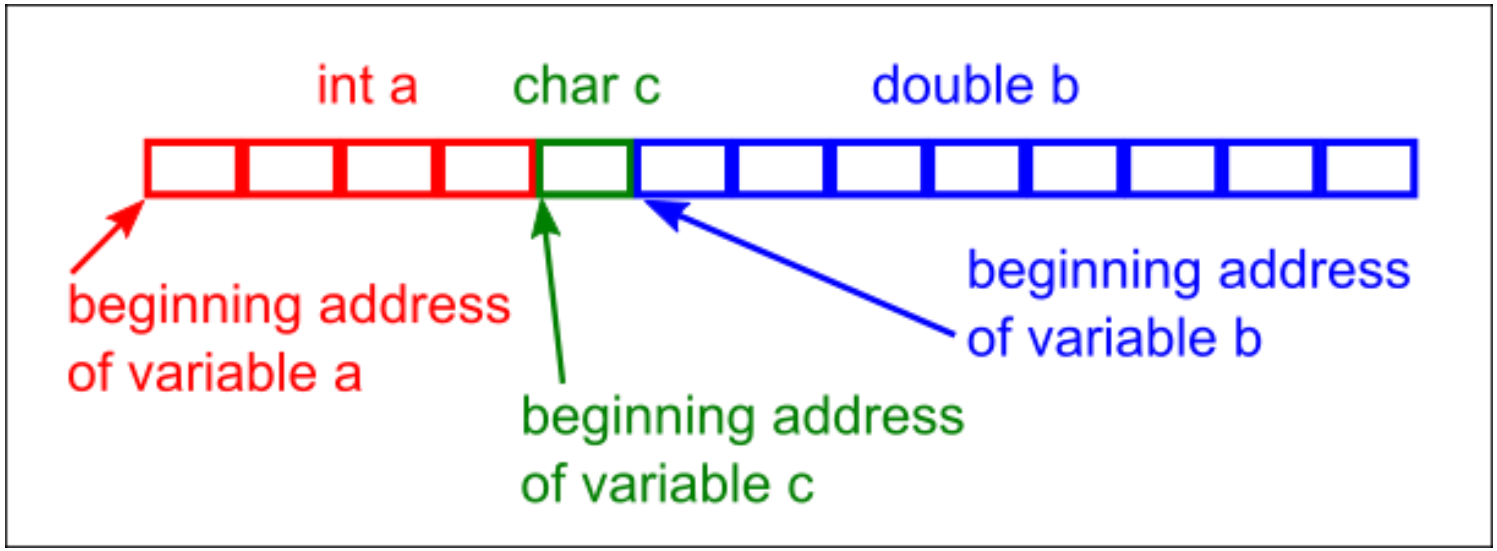
The number of neighbouring bytes belonging to a variable principally depends on the type of the variable (e.g. 4 bytes are allocated for a variable with int type).

So, the position of the memory space reserved for a given variable is identified by the memory address of its first byte (beginning address).

Since the position of the memory space reserved for a variable does not change during its lifetime, a variable is a static object of the memory.

Pointers

A section in the range of memory



Pointers

Pointer

In some cases, it is necessary to reach different memory spaces by means of their beginning addresses in a program. For the sake of that cause, a special variable called pointer was introduced in C language.

The main difference between a common variable and a pointer is that a pointer does not store a data value but a memory address.

Similarly to a conventional variable, a pointer also has a type and a unique identifier as well as a suitable memory space will be reserved for it.

But the memory space reserved for a pointer is used for storing the beginning address of a given memory space.

Therefore we can say that a pointer points to the beginning of a memory space.

Pointers

Pointer

We can also say that a memory address is assigned to a pointer when we specify the memory address to be stored in the memory space of a pointer.

So, a pointer points to a memory address which is actually assigned to it.

Of course the value of memory address assigned to a pointer can be changed in the program.

In such a case, a pointer can point to different memory addresses during the execution of a program.

Pointers

Pointer

Similarly to the variables, all the pointers used in a program must be define at the beginning of the body of a main function.

The general form of the definition of a pointer:

```
type *identifier;
```

e.g.

```
int *ptr;
```

The example above defines a pointer which can be used to store the beginning address of a variable with int type.

The asterisk (*) in the definition indicates that not a variable but a pointer is defined here.

So, the type specification and the asterisk separated by a space character (e.g. int *) stand together.

Directly after the definition, a pointer does not point anywhere yet that is no memory address is assigned to it.

Pointers

Pointer

Therefore we must assign the required memory address to the defined pointer by means of an assignment statement.

Example.

```
int a;
int *ptr;
a = 5;
ptr = &a;
```

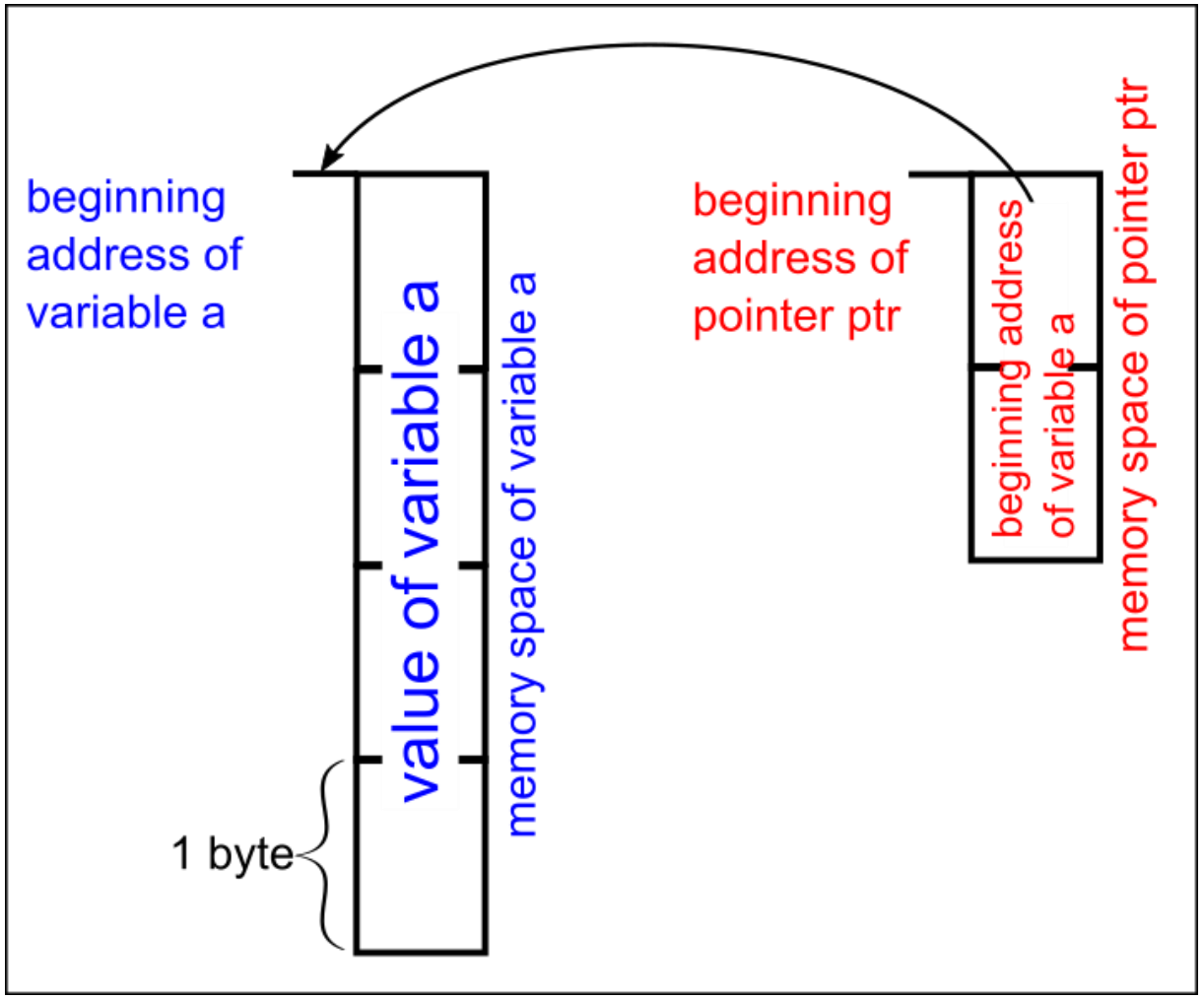
The address (&) operator determines the beginning address of the memory space reserved for the common variable after the operator (we have used it for the library function called scanf).

By the effect of the equals sign, the beginning address will be assigned to the pointer.

Thus, the pointer will point to the beginning address of the variable.

Pointers

```
int a;
int *ptr;
a = 5;
ptr = &a;
```



Pointers

After assigning the beginning address of a variable to the pointer, we can reach the value of the variable by means of the pointer. Example:

```
#include <stdio.h>
main(){
        int a=5;
        int *ptr;
/*assigning the beginning address of variable a to the pointer
ptr*/
        ptr = &a;
/*direct access to the value of variable a*/
        printf("a = %d\n", a);
/*indirect access to the value of variable a by means of the
pointer ptr*/
        printf("*ptr = %d\n", *ptr);
/*the memory address stored by the pointer (the beginning
address of the variable a)*/
        printf("ptr = %p", ptr);
}
```

Pointers

The asterisk (*) is the indirection operator when we apply it to a pointer.

It accesses the value of the variable to which the pointer points.

We can also apply it in expressions either in the left or the right side of an assignment.

By means of the indirection operator, we can change the value of the variable, as well. Example:

```
int a;
int *ptr;
a = 5;
ptr = &a;
a = *ptr + 2;
```

The asterisk in the definition of a pointer is not an indirection operator but a symbol which indicates that a pointer is defined not a common variable.

Pointers

An example for changing the value of a variable by means of a pointer:

```
#include <stdio.h>
main(){
	int a=5;
	int *ptr;
	ptr = &a;
	printf("a = %d\n", a);
	printf("*ptr = %d\n", *ptr);
	*ptr = *ptr % 2;
	printf("a = %d\n", a);
	printf("*ptr = %d\n", *ptr);
}
```

Pointers

We can access more than one variable by means of a pointer if we change the beginning address stored in the pointer.

```
#include <stdio.h>
main(){
        double x=0.5, y;
        double *ptr;
        y = 3 * x;
        ptr = &x;
        printf("ptr = %p\n", ptr);
        *ptr = x + y;
        printf("x = %f\n", x);
        printf("y = %f\n", y);
        printf("*ptr = %f\n", *ptr);
        ptr = &y;
        printf("ptr = %p\n", ptr);
        *ptr = x + y;
        printf("x = %f\n", x);
        printf("y = %f\n", y);
        printf("*ptr = %f\n", *ptr);
}
```

Arrays

Array is a compound data type which can be derived from the basic data types (e.g. char, int, float, double).
It provides the storage and use of several data with the same type collectively. So, an array has several elements.
The simplest form of arrays are the so-called one-dimensional arrays. A one-dimensional array contains a sequence of data with the same type. The number of its elements gives the size of the array.
When the type of a one-dimensional array is numerical (e.g. int, float …), it can be correspond to the concept of vector in mathematics.
Before using an array in our program, we must define it in the body of the main function.

The general form of the definition of a one-dimensional array:

type array_identifier[size];

Type gives the type of the elements building up the array.
All the basic data types may be applied except the type void.
Array_identifier is a unique name for the array.
Size is a positive integer which gives the number of elements belonging to the array. The value of size is located between square brackets.

Arrays

Example:

```
int x1[5], x2[10];
double y[10];
```

In fact, the definition of an array entails the reservation of a memory space and the array identifier will point to the beginning address of that memory space.

The size of the reserved memory space depends on the type of the array and the size of the array (e.g. an array with int type and 5 elements requires 5 * 4 bytes for the storage of the elements).

The array identifier is similar to a pointer because both of them point to a given memory address.

But there is an essential difference between them.

While the value of the memory address may be changed in the case of a pointer, the array identifier points to the same memory address during the program.

Arrays

During the definition of an array, we can also assign initial values to the elements of the array.

Example:

```
float x[4] = { -1.56, 0.87, 6.78, -23.87 };
char w[] = { 'c', 's', 'o', 'k', 'i' };
```

We cannot assign more initial values to an array than the number of its elements.

When we assign initial values to an array in its definition, we do not need to specify the size of the array, because the compiler can determine the size from the number of initial values.

Arrays

It often occurs that the elements of an array give their values as a result of the computations.
When we want to assign values or expressions to the elements of an array, we must use the index values (or subscript) of the elements for their unambiguous identification.
Example:

```
x[2] = a * b;
```

In the example, the element of array x with the index 2 is actually the third element of the array because index values always start with zero in C language. The index value is enclosed by a pair of square brackets.
To assign values to the elements of an array, for loops are used more frequently in the source code.
The following example shows how we can assign values to the elements of an array and how we can use the values of elements in computations by means of for loops.

Arrays

```
#include <stdio.h>
#include <stdlib.h>      /*it contains the prototype of exit*/
#include <math.h>        /*it contains the prototype of sqrt*/
#define N 4      /*a symbolic constant is used for specifying the size
of the array*/
main(){
    int i;
    float data[N];
    printf("Type non-negative numbers!\n");
    for (i=0; i<N; i++){
        printf("the value of %d. number: ", i+1);
        scanf("%f",&data[i]);
        if (data[i]<0){
            printf("Wrong datum!\nThe program stops");
            exit(-1);
        }
    }
    printf("\nThe square roots of the given numbers.\n");
    for (i=0; i<N; i++)
        printf("the square root of %d. number: %f\n", i+1,
sqrt(data[i]));
}
```

Arrays

Multi-dimensional arrays

Multi-dimensional arrays can be also created and used in our programs.

The general form of the definition of a multi-dimensional array:

```
type array_identifier[size_1] [size_2] … [size_n];
```

So, we must specify the size for each dimension of the array.

Example for the definition of a two-dimensional array with int type:

```
int array2D[3][4];
```

By means of 2D arrays we can represent matrixes in our programs.

The 2D array defined above has 3 rows and 4 coloumns, so it corresponds to a matrix with the size 3 x 4.

We can also assign initial values to the elements of a 2D array in the definition. Example:

```
float matrix[3][4]= {{-1.2, 5.8, 0.6, 11.5},
                     {8.5, -5.7, 14.2, 0.1},
                     {3.7, -4.5, 5.1, -6.5}};
```

Arrays

In the case of a 2D array, two indexes (or subscripts) are used for identifying a single element of the array.

Both indexes start with zero.

So, `matrix[2][3]` means the element of the third row and fourth column, whose value is `-6.5` by the previous definition.

In our programs, generally two for loops are used for assigning values to the elements of a 2D array.

One of the for loop is embedded in the other.

Arrays

```
#include <stdio.h>
#include <math.h>
#define N 3
#define M 4
main(){
        int i, j;
        float matrix[N][M]= {{-1.2, 5.8, 0.6, 11.5}, {8.5, -5.7, 14.2, 0.1},
                            {3.7, -4.5, 5.1, -6.5}};

        printf("The elements of the matrix\n");
        for (i=0; i<N; i++){
                for (j=0; j<M; j++){
                        printf("%f\t", matrix[i][j]);
                }
                putchar('\n');
        }
        printf("\nThe square of the elements of matrix\n");
        for (i=0; i<N; i++){
                for (j=0; j<M; j++){
                        printf("%f\t", pow(matrix[i][j],2));
                }
                putchar('\n');
        }
}
```