



C PROGRAMING

Ketan Kore

Sunbeam Infotech



Arrays

- Array is collection of similar data elements in contiguous memory locations.
- Elements of array share the same name i.e. name of the array.
- They are identified by unique index/subscript. Index range from 0 to n-1.
- Array indexing starts from 0.
- Checking array bounds is responsibility of programmer (not of compiler).
- Size of array is fixed (it cannot be grow/shrink at runtime).

```
int main() {  
    int i, arr[5] = {11, 22, 33, 44, 55};  
    for(i=0; i<5; i++)  
        printf("%d\n", arr[i]);  
    return 0;  
}
```

	0	1	2	3	4
arr	11	22	33	44	55
	400	404	408	412	416
	arr[0]	arr[1]	arr[2]	arr[3]	arr[4]



Arrays

- If array is initialized partially at its point of declaration rest of elements are initialized to zero.
- If array is initialized partially at its point of declaration, giving array size is optional. It will be inferred from number of elements in initializer list.
- The array name is treated as address of 0th element in any runtime expression.
- Pointer to array is pointer to 0th element of the array.



Pointer arithmetic

- Scale factor plays significant role in pointer arithmetic.
- n locations ahead from current location
 - $\text{ptr} + n = \text{ptr} + n * \text{scale factor of ptr}$
- n locations behind from current location
 - $\text{ptr} - n = \text{ptr} - n * \text{scale factor of ptr}$
- number of locations in between
 - $\text{ptr1} - \text{ptr2} = (\text{ptr1} - \text{ptr2}) / \text{scale factor of ptr1}$



Pointer arithmetic

- When pointer is incremented or decremented by 1, it changes by the scale factor.
- When integer 'n' is added or subtracted from a pointer, it changes by $n * \text{scale factor}$.
- Multiplication or division of any integer with pointer is not allowed.
- Addition, multiplication and division of two pointers is not allowed.
- Subtraction of two pointers gives number of locations in between. It is useful in arrays.



Pointer to array

```
int main() {  
    int i, arr[5] = { 11, 22, 33 };  
    int *ptr = arr;    it start from the initial array arr[0]  
    for(i=0; i < 5; i++) {  
        printf("%d %d %d %d\n",  
            arr[i], *(arr+i), *(i+arr), i[arr]);  
        printf("%d %d %d %d\n",  
            ptr[i], *(ptr+i), *(i+ptr), i[ptr]);  
    }  
    return 0;  
}
```



Passing array to function

- Array can be passed to function by address only.
- To collect it in formal argument, array or pointer notation can be used.
 - `void print_array(int arr[]);`
 - `void print_array(int *arr);`
- Since it is pass by reference, any changes done in array within called function will be visible in calling function.
- You should not return address of local array from the function, because local variables will be destroyed when function returns.

```
#include <stdio.h>

int main() {
    int arr[5] = { 11, 22, 33, 44, 55 };
    print_array(arr, 5);
    return 0;
}

void print_array(int arr[], int n) {
    int i;
    for(i=0; i<n; i++)
        printf("%d\n", arr[i]);
}
```



Type qualifier – const

- const keyword inform compiler that the variable is not intended to be modified.
 - Compiler do not allow using any operator on the variable which may modify it e.g. ++, --, =, +=, -=, etc.
- |
- Note that const variables may be modified indirectly using pointers. Compiler only check source code (and do not monitor runtime execution).



Constant pointers

- `int a = 10;`
- `const int *ptr = &a;`
- `int const *ptr = &a;`
- `int * const ptr = &a;`
- `int * ptr const = &a;`
- `const int * const ptr = &a;`
- `const int * const ptr = &a;`





Thank you!

Ketan Kore<Ketan.Kore@sunbeaminfo.com>

