C PROGRAMING

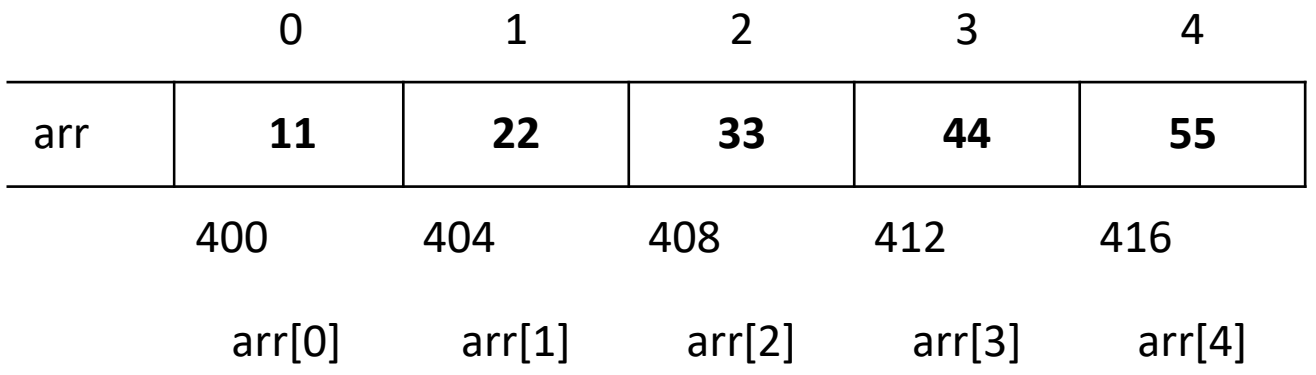
Ketan Kore

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;
}
```



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 0^{th} element in any runtime expression.
- Pointer to array is pointer to 0^{th} element of the array.

Pointer arithmetic

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

- const int * const ptr = &a

NULL pointer

- If pointer is uninitialized, it will hold garbage address (local pointer variables).
- Accessing such pointer may produce unexpected results. Such pointers are sometimes referred as wild pointers.
- C defined a symbolic const NULL, that expands to (void*)0.
- It is good practice to keep well known address in pointer (instead of garbage).
- NULL is typically used to initialize pointer and/or assign once pointer is no more in use.
- Many C functions return NULL to represent failure.
  - strchr(), strstr(), malloc(), fopen(), etc.

void pointer

- Void pointer is generic pointer it can hold address of any data type (without casting).
- Scale factor of void* is not defined, so cannot perform pointer arithmetic.
- To retrieve value of the variable need type-casting.
- void* is used to implement generic algorithms.

Thank you!

Ketan Kore<Ketan.Kore@sunbeaminfo.com>