C PROGRAMING

Ketan Kore

Sunbeam Infotech

Function arguments

```
void sumpro(int a, int b, int ps, int pp) {
    ps = a + b;
    pp = a * b;
}
int main() {
    int x = 12, y = 4, s, p;
    sumpro(x, y, s, p);
    printf("%d %d", s, p);
    return 0;
}
```

Function arguments

```
void sumpro(int a, int b, int ps, int pp) {
    ps = a + b;
    pp = a * b;
}
int main() {
    int x = 12, y = 4, s, p;
    sumpro(x, y, s, p);
    printf("%d %d", s, p);
    return 0;
}
```

Passing arguments: Call by value vs Call by address/reference

- Call by value
  - Formal argument is of same type as of actual argument.
  - Actual argument is copied into formal argument.
  - Any change in formal argument does not reflect in actual argument.
  - Creating copy of argument need more space as well as time (for bigger types).
  - Most of data types can be passed by value – primitive & user defined types.
- Call by address
  - Formal argument is of pointer type (of actual argument type).
  - Address of actual argument is collected in formal argument.
  - Actual argument can be modified using formal argument.
  - To collect address only need pointer. Pointer size is same irrespective of data type.
  - Array and Functions can be passed by address only.

Pointer - Introduction

- Pointer is a variable that stores address of some memory location.
- Internally it is unsigned integer (it is memory address).
- In C, pointer is a special data type.
- It is not compatible with unsigned int.
- Pointer is derived data type (based on primitive data type).
  - To store address of int, we have int pointer.
  - To store address of char, we have char pointer, …
- Size of pointer variable is always same, irrespective of its data type (as it stores only the address).

Pointer - Syntax

- Pointer syntax:
  - Declaration:
    - double *p;
  - Initialization:
    - p = &d;
  - Dereferencing:
    - printf("%lf\n", *p);
- Reference operator - &
  - Also called as direction operator.
  - Read as “address of”.
- Dereference operator - *
  - Also called as indirection operator.
  - Read as “value at”.

Pointer - Syntax

```
int main() {
  double a = 1.2;
  double *p = &a;
  double **pp = &p;
  printf("%lf\n", a);
  printf("%lf\n", *p);
  printf("%lf\n", **pp);
  return 0;
}
```

- Pointer to pointer stores address of some pointer variable.
- Level of indirection: Number of dereference operator to retrieve value.

Pointer - Scale factor

- Size of data type of pointer is known as Scale factor.
- Scale factor defines number of bytes to be read/written while dereferencing the pointer.
- Scale factor of different pointers
  - Pointer to primitive types: char*, short*, int*, long*, float*, double*
  - Pointer to pointer: char**, short**, int**, long**, float**, double**, void**
  - Pointer to struct/union.
  - Pointer to enum.

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

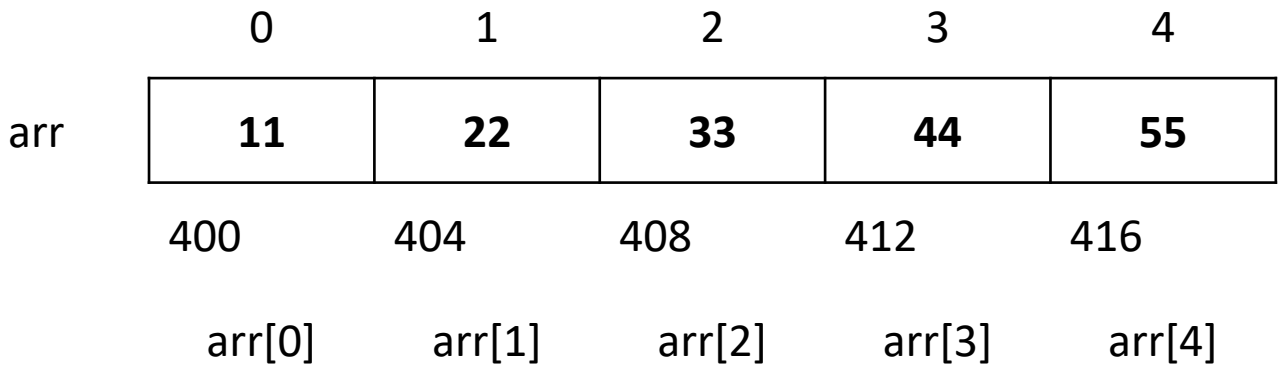
Pointer arithmetic

- 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.

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.

Thank you!

Ketan Kore<Ketan.Kore@sunbeaminfo.com>