Sunbeam Institute of Information Technology
Pune and Karad

Module - Embedded C Programming

Trainer - Devendra Dhande

Email – devendra.dhande@sunbeaminfo.com

Pointer arithmetic

- multiplication, division & modulus is not allowed with pointers.
- only addition & subtraction is allowed
- Above two operations are performed in two different ways.

1. operand 1 is pointer & operand 2 is number
   - addition & subtraction both are allowed

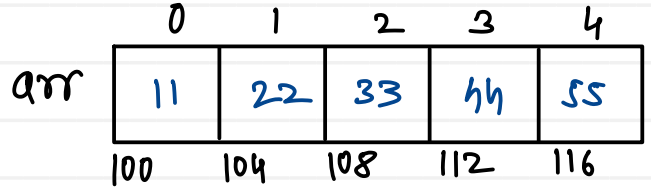
$$ptr \pm n = ptr \pm n * \text{scale factor of } ptr$$

2. operand 1 & 2, both are pointers
   - only subtraction is allowed

$$ptr1 - ptr2 = (ptr1 - ptr2) / \text{scale factor of } ptr1$$

Array

- Array is collection of similar type of data in contiguous memory locations
- Every element of array has unique index starting from 0 to N-1



- to access array elements, we need '[]' operator (subscript)

arr[i] → i^{th} index element

- Declaration :

```
int arr[5];
int arr[5] = {11, 22, 33, 44, 55};
```
← Array initializer list
```
int arr[5] = {11, 22, 33};
```
← partial initization (remaining elements will be made 0)
```
int arr[ ] = {11, 22, 33, 44, 55};
```
← length of array will be inferred by looking at array initializer list
```
int arr[ ]; — error (not allowed)
```

Array size = N * sizeof(datatype)

Where N – number of elements in array.

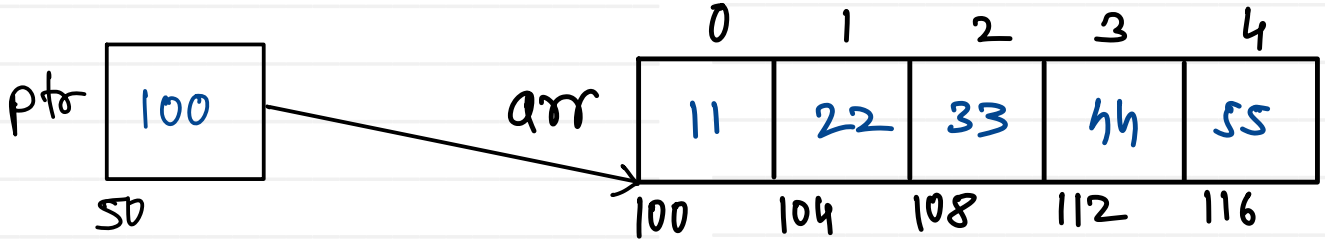
e.g. `int arr[5];`

sizeof(arr) = 5 * sizeof(int)
= 5 * 4
= 20 bytes

Pointer to array

```
int *ptr = arr;
```

```
int arr[5] = {11, 22, 33, 44, 55}
```



arr indicates base address of array

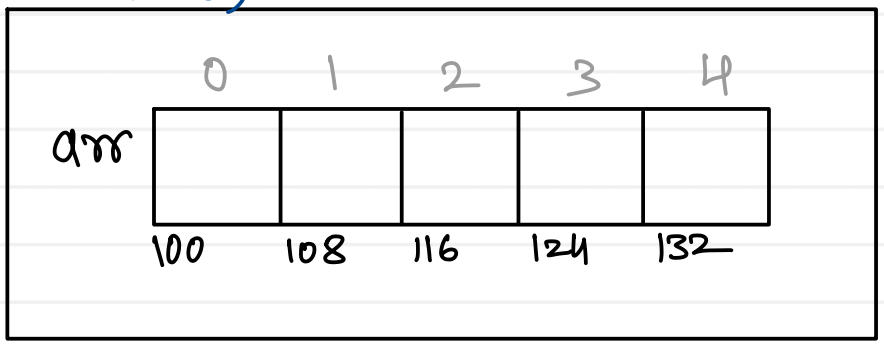
```
arr = 100
ptr = 100
```

```
arr[i] == ptr[i]
```

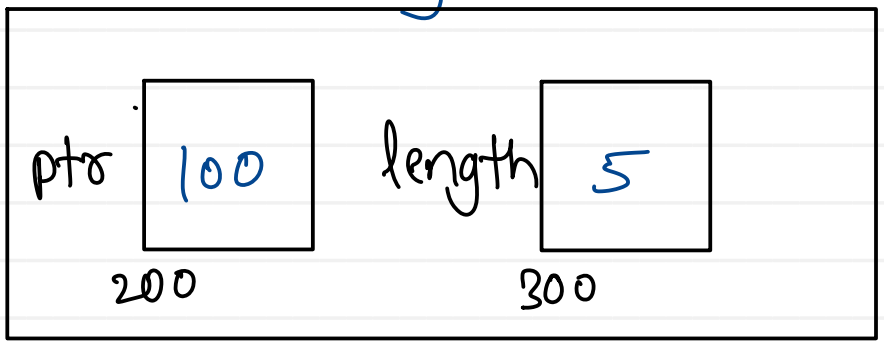
- Arrays are always passed to the function by address (starting address/base address)

Passing array to function

main()



accept_array(arr, 5)



pointers are used to pass array to the function efficiently.
(only base address is copied from actual to formal arg. instead coping of whole array).

```
scanf("%lf", &ptr[i]);
```

↓

~~&~~ *(ptr+i)

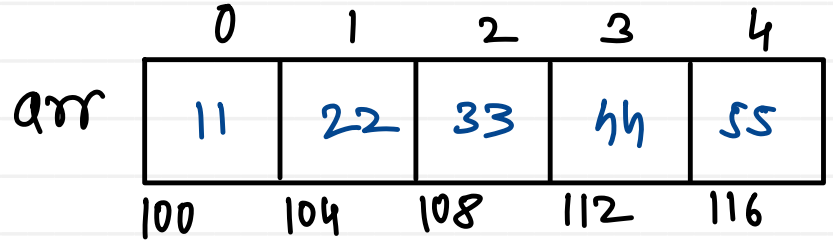
↓

ptr+i

ptr = 100
ptr+0 = 100
ptr+1 = 108
ptr+2 = 116

Pointer notation

```
int arr[5] = {11, 22, 33, 44, 55};
```



arr = 100

arr[0] = 11
arr[1] = 22
arr[4] = 55

arr + 0 = 100
arr + 1 = 104
arr + 4 = 116

arr[i] = *(arr + i)

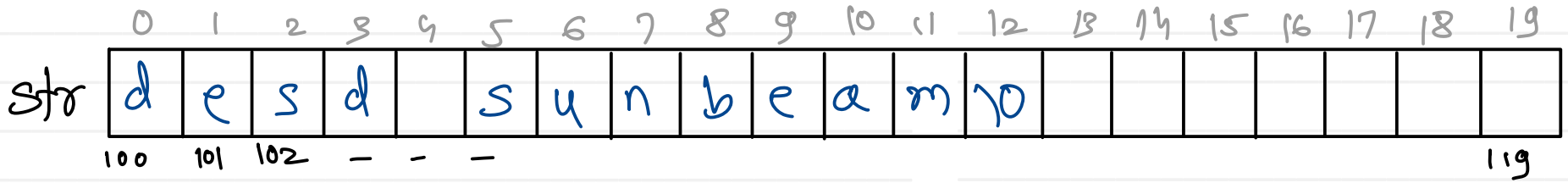
*(arr+0) = 11
*(arr+1) = 22
*(arr+4) = 55

*arr+1 = 11 + 1 = 12

arr[i] = *(arr+i)
*(i+arr) = i[arr]

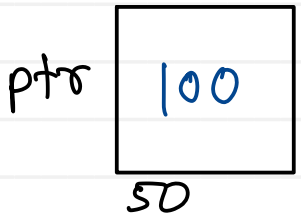
~~&~~arr[i] = ~~&~~*(arr+i)

```
char str[20];
```



str - base address of array (starting address)
(address of 0th index element)

```
char *ptr = str;
```



ptr = 100
&ptr = 50
*ptr = 'd'

ptr+1 = 101
ptr+3 = 103

```
str1[] = "    ";
str2[] = "    ";
```

```
str1 = str2;  - error
str1 == str2 }
str1 != str2 } - 0
```

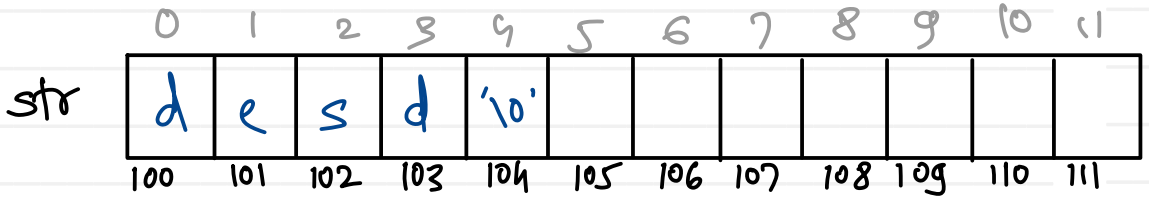
```
str[i] = *(str + i)
ptr[i] = *(ptr + i)
```

*(ptr+i) → i^{th} index character

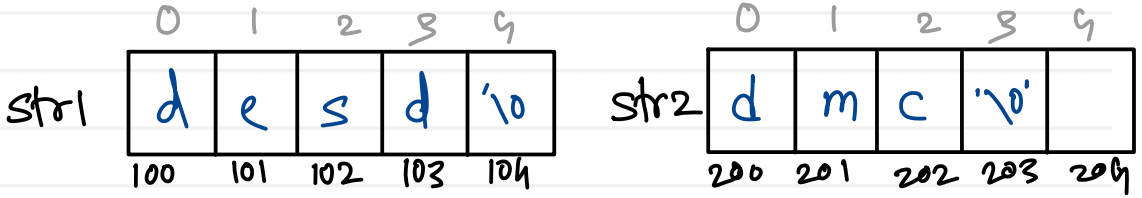
ptr+i → address of i^{th} index character

String

```
char str[12] = "desd";
```




```
char str1[5] = "desd";
char str2[5] = "dmc";
```



str1 == str2 100 == 200 ✗

str1 = str2 - error

string.h

1. strlen() - length of string
2. strcpy() - copy the string
3. strcat() - concat one string with other
4. strcmp() - compare two strings
5. strchr() - locate character in string
6. strstr() - locate substring in string
7. strtok() - tokenize the string (split into multiple tokens)
8. strncpy() - copy first n characters
9. strncat() - concat first n characters
10. strncmp() - compare first n characters
11. strupr() - convert into uppercase
12. strlwr() - convert into lowercase
13. strrev() - reverse the string

```
char str1[20] = "desd";
```

str

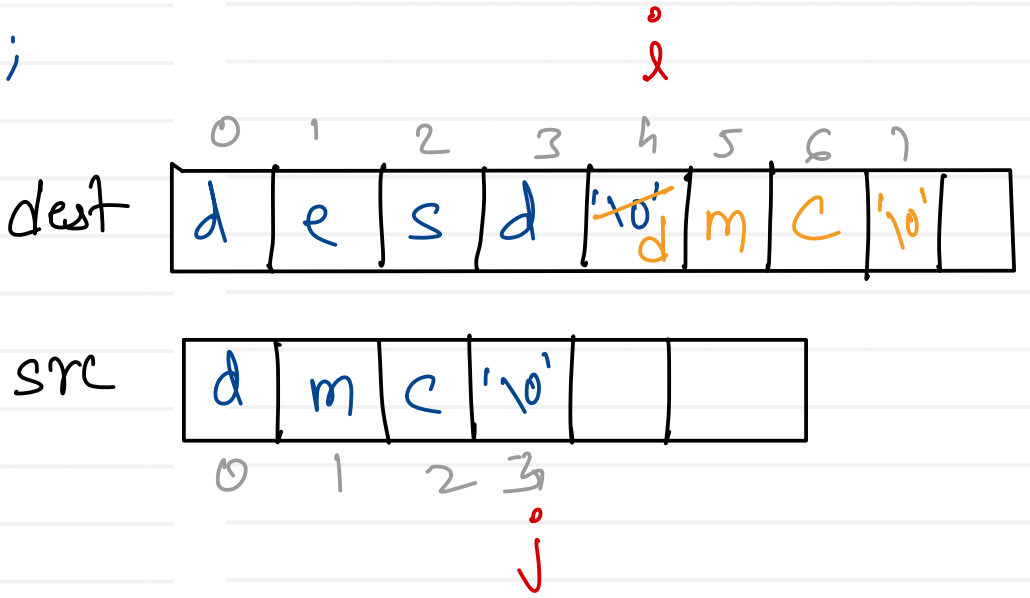
```
int _strlen(char *str)
{
    int len = 0;
    for(i=0; str[i] != '\0'; i++)
        len++;
    return len;
}
```

```
char str1[] = "desd";
char str2[20];
```

desd'\0'

```
char *_strcpy(char *dest, char *src)
{
    int i;
    for(i=0; src[i] != '\0'; i++)
        dest[i] = src[i];
    dest[i] = '\0';
    return dest;
}
```

```
char *_strcat(char *dest, char *src)
{
    int i,j;
    for(i=0; dest[i]!='\0'; i++);
    for(j=0; src[j]!='\0'; j++)
        dest[i+j] = src[j];
    dest[i+j] = '\0';
    return dest;
}
```



str1 = "desd"

str2 = "desd"

return 0

str1 = "desd"

str2 = "dmc"

return 'e' - 'm' = -8

str1 = "dmc"

str2 = "desd"

return 'm' - 'e' = 8

```
int strcmp(char *s1, char *s2)
{
    int i;
    for(i=0; s1[i] != '\0'; i++)
    {
        if(s1[i] != s2[i])
            return s1[i]-s2[i];
    }
    return s1[i]-s2[i];
}
```

s1 = d m c '\0'

s2 = d m '\0'

s1 = d m '\0'

s2 = d m c '\0'

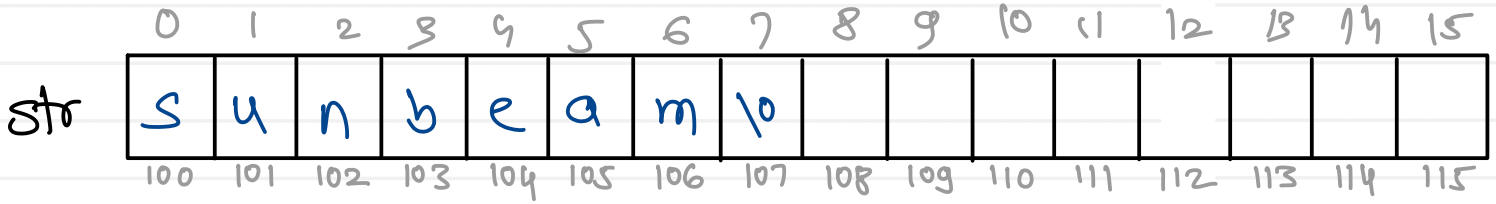
s1 = d m c '\0'

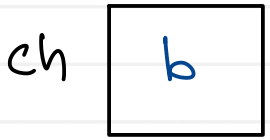
s2 = d m c '\0'

strchr()

```
char *strchr(const char *s, int c);
```

```
strchr(str, ch);
```



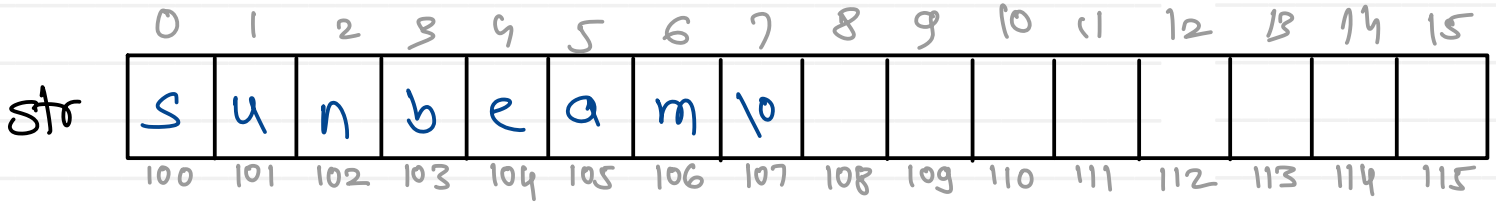


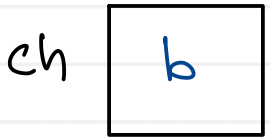

```
char *strchr(const char *s, int c)
{
for(int i=0; s[i] != '\0'; i++)
{
    if(s[i] == c)
        return s+i;
}
return NULL;
}
```

strrchr()

```
char *strrchr(const char *s, int c);
```

```
strchr(str, ch);
```

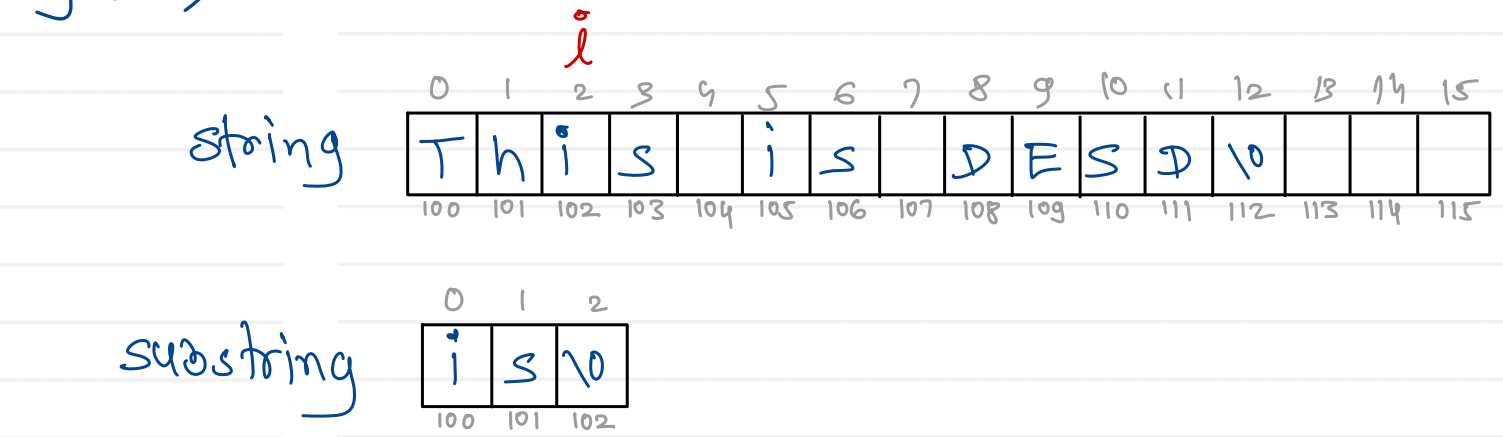





```
char *strrchr(const char *s, int c)
{
    char *ptr = NULL;
    for(int i=0; s[i] != '\0'; i++)
    {
        if(s[i] == c)
            ptr = s + i;
    }
    return ptr;
}
```

strstr()

```
char * strstr(const char* string , const char* substring)
{
    size_t len = strlen(substring);
    for(int i=0; string[i] != '\0'; i++)
    {
        if(string[i] == substring[0])
        {
            if(strncmp(string+i, substring, len)==0)
                return string+i;
        }
    }
    return NULL;
}
```



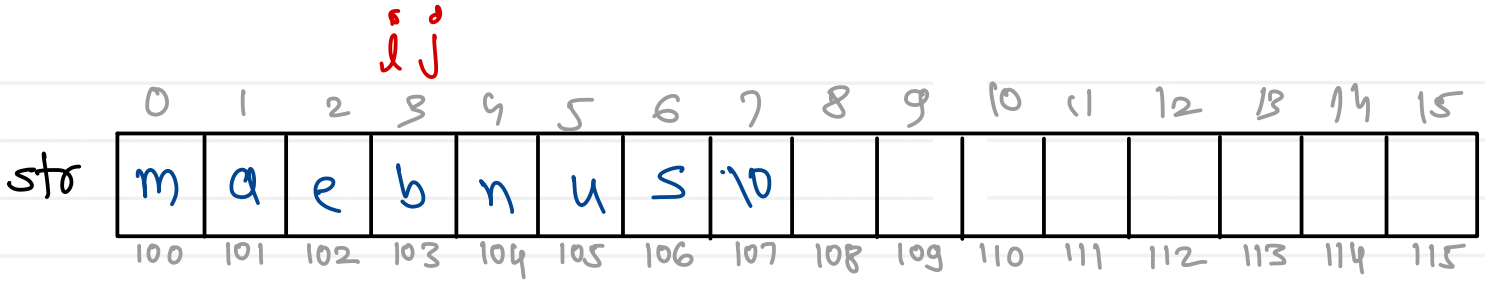
strupr() and strlwr()

Upper case letters 65 → 90
Lower case letters 97 → 122

```
char * strupr( char *s )
{
    for(int i=0; s[i]!='\0'; i++)
    {
        if( s[i]>=97 && s[i]<=122)
            s[i]-=32;
    }
    return s;
}
```

```
char * strlwr( char *s )
{
    for(int i=0; s[i]!='\0'; i++)
    {
        if( s[i]>=65 && s[i]<=90 )
            s[i]+=32;
    }
    return s;
}
```

strrev()




```
char * strrev ( char *s)
{
    int i=0, j= strlen(s)-1;
    while(i<j)
    {
        char temp = s[i];
        s[i] = s[j];
        s[j] = temp;
        i++; j--;
    }
    return s;
}
```

```
Char * strrev ( char * s)
{
    size_t len = strlen(s);
    char temp[len+1];
    int i=0
    for(int j= len-1 ; j>=0 ; j--)
        temp[i++] = s[j];
    temp[i] = '\0';
    for(i=0 ; temp[i] != '\0' ; i++)
        s[i] = temp[i];

    return s;
}
```

not efficient soln in terms of time & space

Thank you!!!

Devendra Dhande

devendra.dhande@sunbeaminfo.com