# ESC101N
# Fundamentals of Computing

Arnab Bhattacharya
arnabb@iitk.ac.in

Indian Institute of Technology, Kanpur
http://www.iitk.ac.in/esc101/

1st semester, 2010-11

Tue, Wed, Fri 0800-0900 at L7

## Strings

- A string is an array of characters
- Strings are not supported as separate data types
- However, they have many specialities
- A string can be declared in the following way:

```
char name[30];
```

- Strings can be initialized in double quotes:

```
char city[] = ``Kanpur'';
```

- Strings can be initialized as array of characters:

```
char ct[] = {`K', `o', `l', `k', `a', `t', `a', `\0'};
```

- A string is an array of characters *terminated with the null character* `\0'
- So, array city contains `\0' in the end and its size is $6 + 1 = 7$
- Since a string is an array of characters, it may be specified as a pointer to character

```
char *str;
```

## String input and output

- Use %s specification

```
char name[30];
scanf(‘‘%s’’, name);
printf(‘‘%s’’, name);
```

- Reading will stop as soon as the first whitespace character (blank, tab, return) is encountered
  - If size of array is larger, rest of the characters are undefined
  - If size of array is smaller, reading overlaps beyond the array boundary
- A string is printed up to the null character

## String input and output

- Use %s specification

```
char name[30];
scanf(``%s'', name);
printf(``%s'', name);
```

- Reading will stop as soon as the first whitespace character (blank, tab, return) is encountered
    - If size of array is larger, rest of the characters are undefined
    - If size of array is smaller, reading overlaps beyond the array boundary
- A string is printed up to the null character
- To read a string with blanks and tabs, use gets

```
char name[30];
gets(name);
```

- gets function does not perform boundary checks and may thus overflow and write characters in memory locations not intended
- puts automatically appends a return character

```
puts(name);
```

## String operations I

```
#include <stdio.h>

int main()
{
    char city[] = "Kanpur";
    char ct[] = {'K', 'o', 'l', 'k', 'a', 't', 'a'};
    char ctn[] = {'K', 'o', 'l', '\n', 'k', 'a', 't', 'a', '\0'};
    char sr[6] = "India";
    char sm[7] = "Asia";
    char sl[4] = "World";

    char str1[4], str2[4], str3[4];

    char lstr[12];

    printf("%s\n", city);
    //printf("%d\n", strlen(city));

    printf("%s\n", ct);
    //printf("%d\n", strlen(ct));

    printf("%s\n", ctn);
    //printf("%d\n", strlen(ctn));

    printf("%s\n", sr);
    printf("%s\n", sm);
    printf("%s\n", sl);

    scanf("%s", str1);
```

## String operations II

```
    scanf("%s", str2);
    scanf("%s", str3);

    printf("%s\n", str1);
    printf("%s\n", str2);
    printf("%s\n", str3);

    //gets(lstr);
    //puts(lstr);
}
```

## String operations

- Include the library `string.h`

## String operations

- Include the library `string.h`
- `strlen(s)` returns the length of string `s`
- It does not count the null character

## String operations

- Include the library `string.h`
- `strlen(s)` returns the length of string `s`
- It does not count the null character
- `strcpy(s, t)` copies string `t` to `s`
- `s = t` copies only the pointer, and *not* the string

## String operations

- Include the library `string.h`
- `strlen(s)` returns the length of string `s`
- It does not count the null character
- `strcpy(s, t)` copies string `t` to `s`
- `s = t` copies only the pointer, and *not* the string
- `strcat(s, t)` appends string `t` to `s`

## String operations

- Include the library `string.h`
- `strlen(s)` returns the length of string `s`
- It does not count the null character
- `strcpy(s, t)` copies string `t` to `s`
- `s = t` copies only the pointer, and *not* the string
- `strcat(s, t)` appends string `t` to `s`
- `strcmp(s, t)` returns the lexicographic comparison between strings `s` and `t`
  - If they are equal, 0 is returned
  - If `s` is later than `t`, a positive integer is returned
  - If `s` is earlier than `t`, a negative integer is returned

## String operations

- Include the library `string.h`
- `strlen(s)` returns the length of string `s`
- It does not count the null character
- `strcpy(s, t)` copies string `t` to `s`
- `s = t` copies only the pointer, and *not* the string
- `strcat(s, t)` appends string `t` to `s`
- `strcmp(s, t)` returns the lexicographic comparison between strings `s` and `t`
    - If they are equal, 0 is returned
    - If `s` is later than `t`, a positive integer is returned
    - If `s` is earlier than `t`, a negative integer is returned
    - Capital letters come earlier than small letters
    - No letter comes earlier than any other letter
    - Blank comes earlier than other letters

## String operations

```
#include <stdio.h>
#include <string.h>

int main()
{
    char str[] = "Kanpur";
    char ca[] = {'K', 'O', 'L', 'K', 'A', 'T', 'A', '\0'};
    char extra[30] = "e";

    printf("%d\t%d\n", strlen(str), strlen(ca));

    printf("%s\n", extra);
    strcpy(extra, ca);
    printf("%s\n", extra);

    strcat(extra, str);
    printf("%s\n", extra);

    printf("%d\t%d\t%d\n", strcmp(str, ca), strcmp(ca, extra), strcmp(extra, str));

    printf("%d\n", strcmp("a b", "ab"));
}
```

## Pointer to character

- Since the size of a string depends on the null character and not the array size, a string is better handled as a pointer to character
- Declaration of a string still requires the array specification

```
char name[30];
```

- However, passing the string to a function is better done as a pointer to character

```
void f(char *)
```

## String copy using arrays

- `strcpy(s, t)` copies string `t` to `s`

## String copy using arrays

- `strcpy(s, t)` copies string t to s
- Array version

```
void strcpy(char s[], char t[])
{
  int i = 0;
  while (t[i] != '\0') // t has not finished
  {
    s[i] = t[i]; // copy
    i++;
  }
  s[i] = '\0'; // terminate s
}
```

## String copy using pointers

- `strcpy(s, t)` copies string `t` to `s`

## String copy using pointers

- `strcpy(s, t)` copies string t to s
- Pointer version

```
void strcpy(char *s, char *t)
{
  while (*t != '\0') // t has not finished
  {
    *s = *t; // copy
    s++; // point to next element
    t++; // point to next element
  }
  *s = '\0'; // terminate s
}
```

## String copy using pointers

- strcpy(s, t) copies string t to s
- Pointer version

```
void strcpy(char *s, char *t)
{
  while (*t != '\0') // t has not finished
  {
    *s = *t; // copy
    s++; // point to next element
    t++; // point to next element
  }
  *s = '\0'; // terminate s
}
```

- Really succinct version

```
void strcpy(char *s, char *t)
{
  for (; *s = *t; s++, t++) // *s = *t is false if *t ==
       '\0'
    ; // empty loop body
}
```

## String comparison

- `strcmp(s, t)` returns the comparison between strings `s` and `t`

## String comparison

- strcmp(s, t) returns the comparison between strings s and t
- Array version

```
int strcmp(char s[], char t[])
{
  int i;
  for (i = 0; s[i] == t[i]; i++) // traverse equal
      elements
    if (s[i] == '\0') // t[i] is also '\0'
      return 0; // only equal elements
  return (s[i] - t[i]); // not just +1 or -1
}
```

## String comparison

- strcmp(s, t) returns the comparison between strings s and t
- Array version

```
int strcmp(char s[], char t[])
{
  int i;
  for (i = 0; s[i] == t[i]; i++) // traverse equal
      elements
    if (s[i] == '\0') // t[i] is also '\0'
      return 0; // only equal elements
  return (s[i] - t[i]); // not just +1 or -1
}
```

- Pointer version

```
int strcmp(char *s, char *t)
{
  for (; *s == *t; s++, t++) // increment pointers
    if (*s == '\0')
      return 0;
  return (*s - *t);
}
```