CS1101: Foundations of Programming

Strings

Dept. of Computer Science & Engineering

Indian Institute of Technology Patna

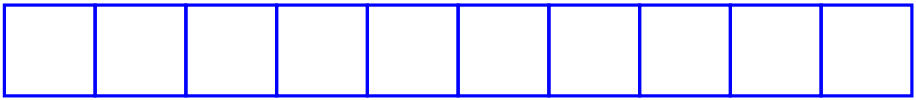
Integer array

- All the data items constituting the group share the same name

```
int a[10];
```

- Individual elements are accessed by specifying the index
- In a similar manner, we can define character array

```
char x[10];
```



Character arrays and Strings

- `char c[]={'a','r','r','a','y','\0'};`
- `c[0]` gets the value `'a'`, `c[4]` - `'y'` and `c[5]` receives NULL (’\0’) character
- Null terminated character arrays are also referred as null terminated strings or simply strings
- Strings can be initialized in the following manner also
  `char x[6] = "array";` or `char x[]= "array";`
- Compiler automatically puts the trailing null character at the end if you define like above
- You can have a bigger size array to store string like `"array"`
- A string ends as soon as it encounters NULL character
- The size of the array should be at least one more than the length of the string it stores

String literals

- For individual character C uses `''` and for strings it uses `""`
- String literal values are represented by sequence of characters between `""`
- Example
  - `"IIT Patna"`
  - `""` - empty string
- `"a" versus 'a'`
  - `'a'` - is single character value (stored in 1 byte) as the ascii value for letter `a`
  - `"a"` - is an array with 2 characters, first is `a`, second is `\0`

Reading strings

- %s reads a string into character array given the array name of start address
- It ends the string with the special null character \0
- scanf with %s can read a string without any whitespace (blank, tab, linebreak)
- Input assumed to end if whitespace is encountered

```
int main(){
   char str[100];
   scanf("%s",str);
   printf("str = %s\n",str);
   return 0;
}
```

Length of string

```
int main(){
   char str[100];
   int i, len=0;
   scanf("%s",str);
   printf("str = %s\n",str);
   for(i=0; str[i] != '\0'; i++)
     len++;
   printf("len = %d\n",len);
   return 0;
}
```

Output:

```
Parthasarathi
str = Parthasarathi
len = 13
```

The character strings read in %s format end with '\0'.

Counting number of a's

```
int main(){
   char str[100];
   int i, count=0;
   scanf("%s",str);
   printf("str = %s\n",str);
   for(i=0; str[i] != '\0'; i++)
     if( str[i] == 'a')
       count++;
   printf("count = %d\n",count);
   return 0;
}
```

Output:

```
Parthasarathi
str = Parthasarathi
count = 4
```

Palindrome

- A palindrome is a word, sentence, verse, or even number that reads the same backward or forward.
- Example: ”madam”, ”adabcbada”, etc.

Palindrome

- A palindrome is a word, sentence, verse, or even number that reads the same backward or forward.
- Example: "madam", "adabcbada", etc.

```
int main(){
   char str[100]; int i, len=0, flag=0;
   scanf("%s",str);
   while (str[len] != '\0') len++;
   for(i=0; str[i] != '\0'; i++)
     if( str[i] != str[len-1-i]) flag = 1;
   if(0 == flag) printf("%s is a palindrome\n",str);
   else  printf("%s is not a palindrome\n",str);
   return 0;
}
```

Counting vowels

```
int main(){
  char str[100], x[]={'a','e','i','o','u'};
  int i, len=0, flag=0, c[]={0,0,0,0,0};
  scanf("%s",str);
```

Counting vowels

```
int main(){
  char str[100], x[]={'a','e','i','o','u'};
  int i, len=0, flag=0, c[]={0,0,0,0,0};
  scanf("%s",str);
  for(len=0; str[len] != '\0'; len++);
  for(i=0; str[i] != '\0'; i++)
    for(j=0; j<5; j++)
      if(str[i] == x[j]) c[j]++;
  for(j=0; j<5; j++)
    printf("%c appears %d times\n",x[j],c[j]);
  return 0;
}
```

Reading strings with blanks

- Many times we need to read the whole line including blank spaces, but `scanf` with `%s` cannot read string with whitespaces
- `getchar()` is another alternative for such scenarios

```
char str[100],ch;
int c=0;
:
do{
   ch = getchar();
   str[c++] = ch;
} while (ch != '\n');
c--;
str[c] = '\0';
```

If the input contains more than 100 characters, then it will lead to a problem.

To avoid this, one can keep track of the value of the count and necessary action can be taken when it crosses the limit.

Example: Remove duplicates

Write a C function that takes a string as argument and modifies the string so as to remove all duplicate characters

Example: `mississippi` **to** `misp`

```
void remDuplicates(char str[]){
```

Example: Remove duplicates

Write a C function that takes a string as argument and modifies the string so as to remove all duplicate characters

Example: `mississippi` **to** `misp`

```
void remDuplicates(char str[]){
   int i, j, idx=0;
   for(i=0; str[i] != '\0'; i++){
     for(j=0; j<i; j++){
       if(str[i] == str[j])
         break;
     }
     if(i == j)
       str[idx++] = str[i];
   }
   str[idx] = '\0';
}
```

Example: Remove duplicates

Write a C function that takes a string as argument and modifies the string so as to remove all consecutive duplicate characters

Example: `mississippi` **to** `misisipi`

```
void remDuplicates(char str[]){
```

Example: Remove duplicates

Write a C function that takes a string as argument and modifies the string so as to remove all consecutive duplicate characters

Example: `mississippi` **to** `misisipi`

```
void remDuplicates(char str[]){
   int i, j, idx=0;
   char prev='\0';
   for(i=0; str[i] != '\0'; i++){
     if(prev != str[i])
       str[idx++] = str[i];
     prev = str[i];
   }
   str[idx] = '\0';
}
```

Example: Copy string using recursion

```
int main(){
   char a[20]="abcde", b[20];
   copy(a,b,0);  printf("%s\n",b);
   return 0;
}

void copy(char src[], char dest[], int idx){
```

Example: Copy string using recursion

```
int main(){
   char a[20]="abcde", b[20];
   copy(a,b,0);  printf("%s\n",b);
   return 0;
}

void copy(char src[], char dest[], int idx){
   dest[idx] = src[idx];
   if(src[idx] == '\0') return;
   copy(src, dest, idx+1);
}
```

String library functions

- A collection of functions primarily operated on null terminated strings
- Need to include header file: `#include <string.h>`
- When an integer / float array is sent to a function as an argument, it is recommended to pass the number of valid entries in the array as an additional argument
- When a string is sent to a function, it is not needed to pass the number of valid characters since `'\0'` can be used to determine the end of the string

Important `string` library functions

- `size_t strlen(const char *)` —

  returns the length of the string

- `int strcmp(const char *, const char *)` —

  compares to two strings in lexicographic manner

- `char *strcat(char *dest, const char *src)` —

  concatenates two strings

- `char *strcpy(char *dest, const char *src)` —

  copy one string to another

strlen()

- It calculates the length of the string pointed to by the given argument, excluding the terminating null character
- Returns the number of bytes in the string
- Example: `len = strlen(str);`
  - The null character at the end is not counted
  - Counting ends at the first null character

strcmp(s1, s2)

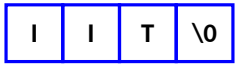
- It compares two strings
- Returns an integer less than, equal to, or greater than zero if `s1` is found to be less than, to match, or be greater than `s2`. Comparison is done in lexicographic manner
- 20 > 9 as integers, ”20” < ”9” as strings
- Example:
  - `if(strcmp(city,"Patna")==0){…}`
  - `if(!strcmp(city,"Patna")){…}`

strcat(dest, src)

- **It appends the `src` string to the `dest` string, overwriting the terminating null character at the end of `dest`, and then adds a terminating null byte**
- **Returns a pointer to the resulting string `dest`**
- **`dest` should have enough space to add `src` - Programmer's responsibility**
- **Example:**
  - `char a[]="IIT";`
  - `char b[10]="Patna ";`
  - `strcat(b, a);`

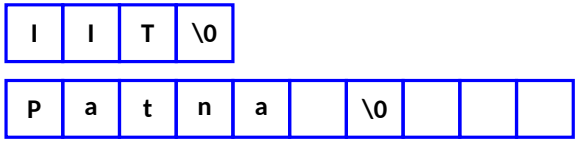
strcat(dest, src)

- **It appends the `src` string to the `dest` string, overwriting the terminating null character at the end of `dest`, and then adds a terminating null byte**
- **Returns a pointer to the resulting string `dest`**
- **`dest` should have enough space to add `src` - Programmer's responsibility**
- **Example:**
  - `char a[]="IIT";`
  - `char b[10]="Patna ";`
  - `strcat(b, a);`



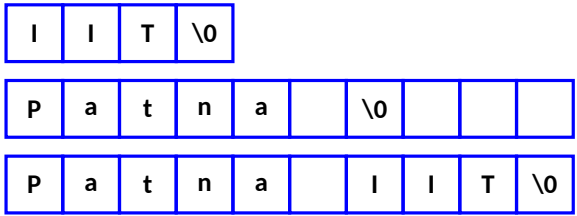
strcat(dest, src)

- It appends the `src` string to the `dest` string, overwriting the terminating null character at the end of `dest`, and then adds a terminating null byte
- Returns a pointer to the resulting string `dest`
- `dest` should have enough space to add `src` - Programmer's responsibility
- Example:
  - `char a[]="IIT";`
  - `char b[10]="Patna ";`
  - `strcat(b, a);`



strcat(dest, src)

- It appends the `src` string to the `dest` string, overwriting the terminating null character at the end of `dest`, and then adds a terminating null byte
- Returns a pointer to the resulting string `dest`
- `dest` should have enough space to add `src` - Programmer's responsibility
- Example:
  - `char a[]="IIT";`
  - `char b[10]="Patna ";`
  - `strcat(b, a);`



strcpy(dest, src)

- **Copies the string pointed to by `src`, including the terminating null byte, to the memory pointed to by `dest`**
- **Returns a pointer to the destination string `dest`**
- **`dest` should have sufficient size to accommodate `src`**
- **Example:**
  - `strcpy(name,"Ayesha")`
  - `strcpy(name1,name2)`
- **Assignment operator does not work for strings.**

```
char name[100];
name = "Ayesha"; // INVALID
```

Example

```
int main(){
  char x[20], y[20]; int val;
  scanf("%s%s",x,y);
  printf("lenx=%d leny=%d", strlen(x), strlen(y));
  val=strcmp(x, y);
  if(val == 0) printf("Same\n");
  else if(val < 0) printf("%s < %s\n",x,y);
  else printf("%s < %s\n",y,x);
  return 0;
}
```