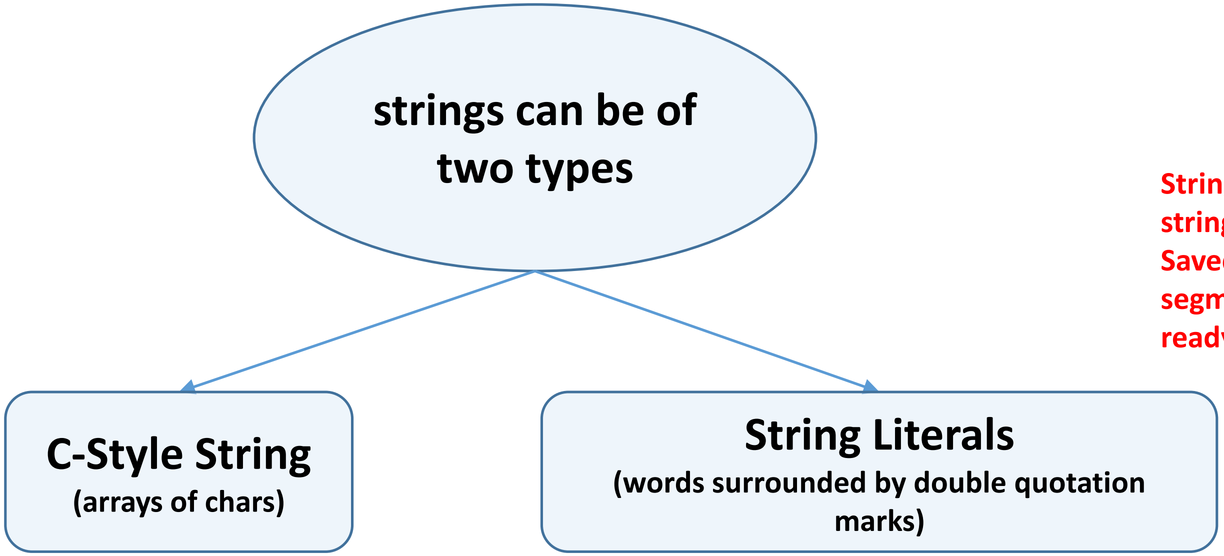
# Strings

- Strings are one-dimensional array of characters.

- A string is terminated by a null character '\0'.

- The size of the character array containing the string is one more than the number of characters in the string itself. e.g. the size of the array that holds the string “Hello” is 6.



strings can be of two types

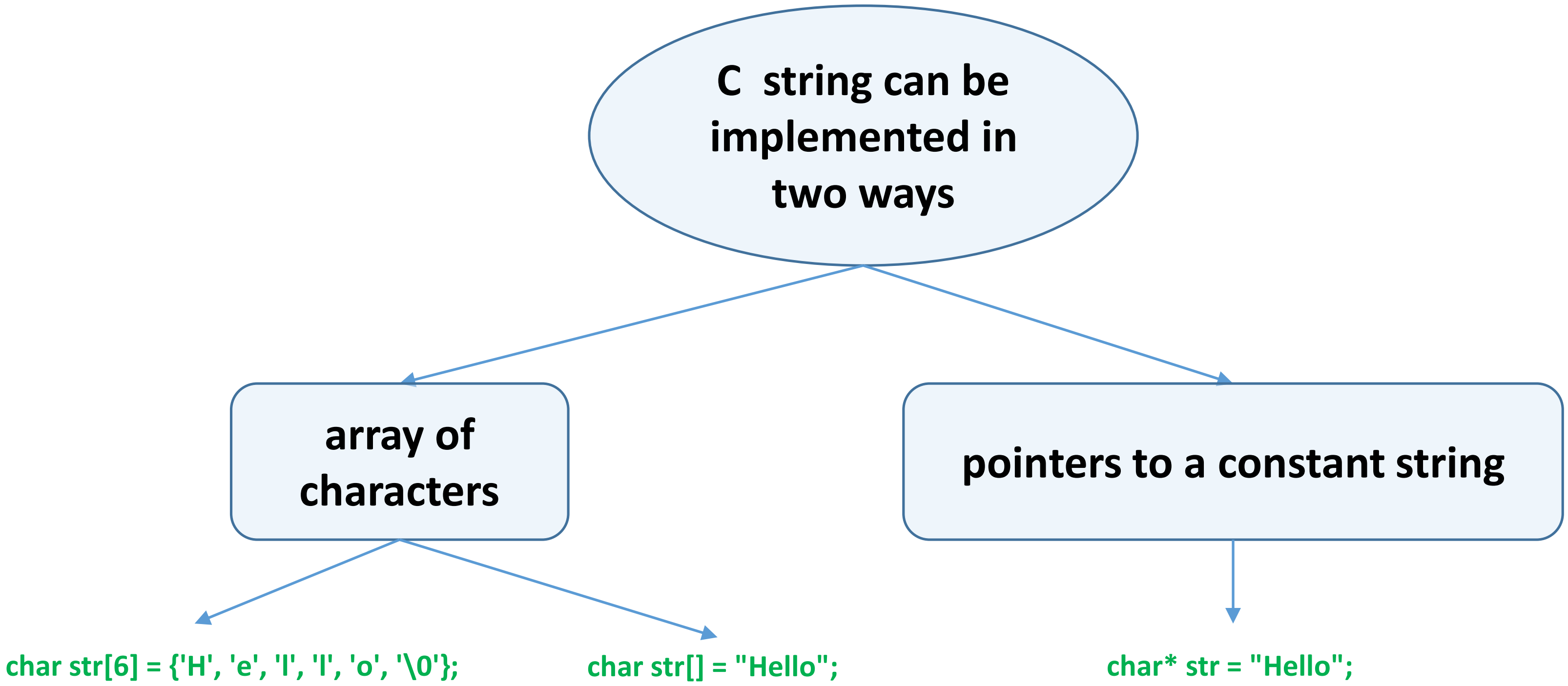
**C-Style String**
(arrays of chars)

**String Literals**
(words surrounded by double quotation marks)

String Literals: is a string constant Saved on data-segment and are ready-only.

✓ {'H', 'e', 'l', 'l', 'o', '\0'}; ← C-Style String
✓ "Hello"; ← String Literals

# Strings Declaration in C



C string can be implemented in two ways

array of characters

pointers to a constant string

```
char str[6] = {'H', 'e', 'l', 'l', 'o', '\0'};
```

```
char str[] = "Hello";
```

```
char* str = "Hello";
```

A disadvantage of creating strings using the character array *syntax* is the fact that the size of array must be specified at compile time. This type of array allocation, where the size of the array is determined at compile-time, is called *static allocation*.

# Declaring arrays initialized by ‘C-Style’ and ‘String Literals’ have the same presentation in memory

In this case the number of characters in the string literal must be at most one less than this length

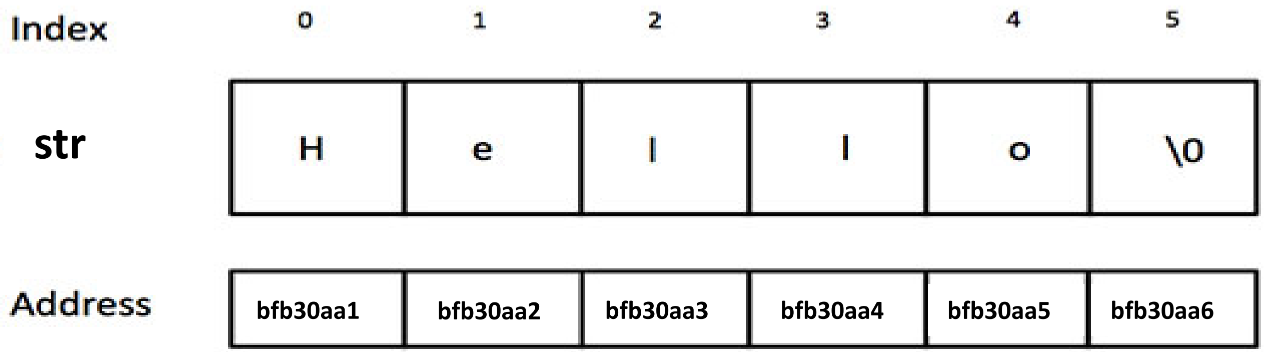
Remember to make sure to make the array long enough to include the null terminator.

✓ char str[6] = {'H', 'e', 'l', 'l', 'o', '\0'}; ← str: array of chars.

✓ char str[6] = "Hello"; ← str: array of chars.

✓ char str[] = "Hello"; ← str: array of chars.

the compiler determines how large to make the array of characters.



| Index | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| str | H | e | l | l | o | \0 |
| Address | bfb30aa1 | bfb30aa2 | bfb30aa3 | bfb30aa4 | bfb30aa5 | bfb30aa6 |

arr is array of characters.

```
#include<stdio.h>
int main()
{
  char arr[] = "abc";
  char *pStr = arr;

  arr[0] = 'A';
  pStr[1] = 'B';
  *(pStr+2) = 'C';

  printf("%s %s\n",arr, pStr);

  return 1;
}
```

✓

**Output:**

```
ABC ABC
```

pStr is pointer to constant string. Constant strings are read only and cannot be modified

```
#include<stdio.h>
int main()
{
  char *pStr = "abc";

  pStr[0] = 'A';
  *(pStr+1) = 'B';

  return 1;
}
```

✗

**Output:**

```
Segmentation fault (core dumped)
```

Run time error

```
#include<stdio.h>
int main()
{
  char s1[4] = {'a','b','c','\0'};
  char s2[] = {'a','b','c','\0'};
  char s3[4] = "abc";
  char s4[] = "abc";
  char *s5 = "abc";
  printf("%s %s %s %s %s\n",s1, s2, s3, s4, s5);
  printf("%c %c %c %c %c\n",s1[0],s2[0],s3[0],s4[0],s5[0]);
  return 1;
}
```

array of characters

String-Literals

pointer to “String-Literals”

**Output:**

```
abc abc abc abc abc
a a a a a
```

# String Modification

```c
#include<stdio.h>
int main()
{
  char s1[4] = {'a','b','c','\0'};
  char s2[] = {'a','b','c','\0'};
  char s3[4] = "abc";
  char s4[] = "abc";
  s1[0] = s2[0] = s3[0] = s4[0] = 'A';
  printf("%s %s %s %s\n",s1, s2, s3, s4);
  return 1;
}
```

**Output:**

```
Abc Abc Abc Abc
```

```c
#include<stdio.h>
int main()
{
  char *ps = "abc";
  ps[0] = 'A';
  *ps = 'A';
  return 1;
}
```

Cannot update string literal through a pointer: String Literals are saved on data segment And are defined as read only.

**Output:**

```
Segmentation fault (core dumped)
```

# modify pStr[index] with upper case character

```
#include<stdio.h>
void upper(char pStr[], int index)
{
   if(pStr[index] >= 'a' && pStr[index] <= 'z')
      pStr[index] = pStr[index] - ('a' - 'A');
}
int main()
{
   char s1[4] = {'a','b','c','\0'};
   char s2[] = {'a','b','c','\0'};
   char s3[4] = "abc";
   char s4[] = "abc";

   upper(s1,0);    upper(s2,0);
   upper(s3,0);   upper(s4,0);
   printf("%s %s %s %s\n",s1, s2, s3, s4);

   return 1;
}
```

✓

**Output:**

```
Abc Abc Abc Abc
```

```
#include<stdio.h>
void upper(char *pStr, int index)
{
   if(pStr[index] >= 'a' && pStr[index] <= 'z')
      pStr[index] = pStr[index] - ('a' - 'A');
}
int main()
{
   char s1[4] = {'a','b','c','\0'};
   char s2[] = {'a','b','c','\0'};
   char s3[4] = "abc";
   char s4[] = "abc";

   upper(s1,0);    upper(s2,0);
   upper(s3,0);   upper(s4,0);
   printf("%s %s %s %s\n",s1, s2, s3, s4);

   return 1;
}
```

✓

**Output:**

```
Abc Abc Abc Abc
```

both pStr[] and *pStr
are pointer to ‘char*’s.

```
#include<stdio.h>
void upper(char pStr[], int index)
{
  if(pStr[index] >= 'a' && pStr[index] <= 'z')
  pStr[index] = pStr[index] - ('a' - 'A');
}
int main()
{
  char *s1 = "abc";
  upper(s1,0);
  printf("%s\n",s1);

  return 1;
}
```

```
Output:
Segmentation fault (core dumped)
```

```
#include<stdio.h>
void upper(char *pStr, int index)
{
  if(pStr[index] >= 'a' && pStr[index] <= 'z')
  pStr[index] = pStr[index] - ('a' - 'A');
}
int main()
{
  char *s1 = "abc";
  upper(s1,0);
  printf("%s\n",s1);

  return 1;
}
```

```
Output:
Segmentation fault (core dumped)
```

In C, there is a difference in the use of brackets ‘[]’ when declaring an array variable *versus* using this array notation as a parameter to a function. With a parameter to a function, a *pointer* is being declared even if using array notation.

# reverse a string in place

```
#include<stdio.h>
#include <string.h>
void strrev(char s[])
{
    int c, i, j;
    for (i = 0, j = strlen(s)-1; i < j; i++, j--)
    {
        c = s[i];
        s[i] = s[j];
        s[j] = c;
    }
}
int main() {
  char s1[4] = {'a','b','c','\0'};
  char s2[] = {'a','b','c','\0'};
  char s3[4] = "abc";
  char s4[] = "abc";
  strrev(s1); strrev(s2); strrev(s3); strrev(s4);
  printf("%s %s %s %s\n",s1, s2, s3, s4);
  return 1;
}
```

**Output:**

```
cba cba cba cba
```

# convert an integer to characters

generate digits in reverse order

from previous page

```
void itoa(int n, char s[]) {
    int i=0, sign;

    if ((sign = n) < 0)    n = -n;

    do {
        s[i++] = n % 10 + '0';
    }
    while ((n /= 10) > 0);

    if (sign < 0)    s[i++] = '-';
    s[i] = '\0';
    strrev(s);
}
int main() {
  int n =123;
  char str[5];
  itoa(n, str);
  printf("%s\n",str);
  return 1;
}
```

# C-Character Supported Functions
# <ctype.h>

| Function | Description |
|---|---|
| int isalpha(char c) | non-zero if c is alphabetic, 0 if not |
| int isupper(char c) | non-zero if c is upper case, 0 if not |
| int islower(char c) | non-zero if c is lower case, 0 if not |
| int isdigit(char c) | non-zero if c is digit, 0 if not |
| int isalnum(char c) | non-zero if isalpha(c) or isdigit(c), 0 if not |
| int isspace(char c) | non-zero if c is blank, tab, newline, return, formfeed, vertical tab |
| int toupper(char c) | return c converted to upper case |
| int tolower(char c) | return c converted to lower case |

**The C programming Language W. Kernighan and Dennis M. Ritchie**

```
#include <stdio.h>
#include <ctype.h>
int main() {
  printf("%d\n",isdigit('1'));
  printf("%c\n",toupper('a'));
  printf("%d\n",isalpha('a'));
  return 1;
}
```

**Output:**

```
2048
A
1024
```

# C-String Supported Functions
# <string.h>

| Function | Description |
|---|---|
| char *strcpy(char* s, const char* ct) | copy string ct to string s, including '\0'; return s. |
| char *strncpy(char* s, const char* ct, size_t n) | copy at most n characters of string ct to s; return s. Pad with '\0''s if ct has fewer than n characters. |
| char *strcat(char* s, const char* ct) | concatenate string ct to end of string s; return s. |
| char *strncat(char* s, const char* ct, size_t n) | concatenate at most n characters of string ct to string s, terminate s with '\0'; return s. |
| int strcmp(const char* cs, const char* ct) | compare string cs to string ct, return <0 if cs<ct, 0 if cs==ct, or >0 if cs>ct. |
| int strncmp(const char* cs, const char* ct, size_t n) | compare at most n characters of string cs to string ct; return <0 if cs<ct, 0 if cs==ct, or >0 if cs>ct |
| char *strchr(const char* cs, int c) | return pointer to first occurrence of c in cs or NULL if not present. |
| char *strrchr(const char* cs, int c) | return pointer to last occurrence of c in cs or NULL if not present. |
| size_t strspn(const char* cs, const char* ct) | return length of prefix of cs consisting of characters in ct. |
| size_t strcspn(const char* cs, const char* ct) | return length of prefix of cs consisting of characters *not* in ct. |
| char *strpbrk(const char* cs, const char* ct) | return pointer to first occurrence in string cs of any character string ct, or NULL if not present. |
| char *strstr(const char* cs, const char* ct) | return pointer to first occurrence of string ct in cs, or NULL if not present |
| char *strerror(size_t n) | return pointer to implementation-defined string corresponding to error n. |
| char *strtok(char* s, const char* ct) | strtok searches s for tokens delimited by characters from ct; |

**The C programming Language W. Kernighan and Dennis M. Ritchie**

# char *strcpy(char* s, const char* ct)
# char *strncpy(char* s, const char* ct, size_t n)

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

int main()
{
  char str1[6];
  char str2[]= {'H','e','l','l','o','\0'};

  strcpy(str1, str2);

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

  return 1;
}
```

```
Output:
Hello
```

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

int main()
{
  char str1[6];
  char str2[]= {'H','e','l','l','o',' ','S','t','u','d','e','n','t','s','\0'};

  strncpy(str1,str2,5);

  str1[5] = '\0';

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

  return 1;
}
```

```
Output:
Hello
```

## char *strcat(char* s, const char* ct)
## char *strncat(char* s, const char* ct, size_t n)

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

int main()
{
  char str1[100] = "Hello";
  char str2[] = "Students";

  strcat(str1,str2);

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

  return 1;
}
```

Output:

```
HelloStudents
```

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

int main()
{
  char str1[100] = "Hello";
  char str2[] = "Students";

  strncat(str1,str2,4);

  str1[10]=0;

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

  return 1;
}
```

Output:

```
HelloStud
```

# int strcmp(const char* cs, const char* ct)
# int strncmp(const char* cs, const char* ct, size_t n)

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

int main()
{
  char s1[] = "abc";
  char* s2 = "abc";
  char* s3 = "def";

  printf("%d\n", strcmp(s1,s2));
  printf("%d\n", strcmp(s2,s3));
  printf("%d\n", strcmp(s3,s2));

  return 1;
}
```

```
Output:
0
-1
1
```

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

int main()
{
  char s1[] = "abc";
  char* s2 = "abC";

  printf("%d\n", strncmp(s1,s2,2));
  printf("%d\n", strncmp(s1,s2,3));

  return 1;
}
```

```
Output:
0
1
```

# char *strstr(const char* cs, const char* ct)

```c
#include <string.h>
#include <stdio.h>
int main(void)
{
    char* str = "one two three";
    char* s1,*s2,*s3;

    s1=strstr(str, "two");
    s2=strstr(str, "nine");
    s3=strstr(str, "n");

    printf("%d %p %d\n",s1-str,(void*)s2,s3-str);

    return 0;
}
```

**Output:**

```
4 (nil) 1
```

# int strlen(const char* s)

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

int main()
{
  char str1[100];
  char str2[]= {'H','e','l','l','o',0};
  char *pStr = "Students";

  strcpy(str1,str2);

  strcat(str1,pStr);

  printf("%d %d %d %s\n", strlen(str2), strlen(pStr), strlen(str1), str1);
  return 1;
}
```

**Output:**

```
5 8 13 HelloStudents
```

# String Copy Version

version1

```
void strCopy(char s1[], char s2[])
{
 int i=0;
 while(s2[i])
 {
  s1[i] = s2[i];
  i++;
 }
 s1[i]=0;
}
```

s1[i]='\0'

version2

```
void strCopy(char s1[], char s2[])
{
 int i=0;
 while((s1[i++] = s2[i]));

 s1[i]=0;
}
```

version3

```
void strCopy(char* s1, char* s2)
{
 while(*s2)
  *s1++=*s2++;

 *s1=0;
}
```

version4

```
void strCopy(char* s1, char* s2)
{
 while((*s1++ = *s2++));
}
```

# strncat Version

version1

```
void strnCat (char s1[], char s2[],int n)
{
 int i=0,j=0;
 while(s1[i])
    i++;
 while(s2[j] && j<n)
    s1[i++] = s2[j++];
 s1[i] = 0;
}
```

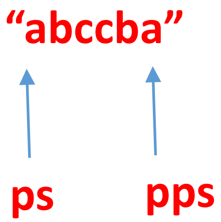
version2

```
void strnCat (char *s1, char *s2,int n)
{
 while(*s1)
  s1++;
 while((*s2 != '\0') && (n-- != 0))
 *s1++=*s2++;
 *s1 = 0;
}
```

# ispalindrome("abc")

Note: if we remove the "const" from "pps" then we will get a warning.

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

int ispalindrome(const char *ps)
{
   const char *pps = ps + strlen(ps)-1;

   while((*ps == *pps) && (ps++<pps--));

   return ps<pps ? 0:1;
}

int main()
{
   printf("%d\n",ispalindrome("abc"));
   return 1;
}
```



"abccba"

ps pps

# ispalindrome(n) , n>=0

```
void strrev(char s[])
{
    char *ps=s+strlen(s)-1,tmp;
    while(s < ps)
    {
        tmp = *ps;
        *ps-- = *s;
        *s++ = tmp;
    }
}
```

```
void itoa(char s[], int n)
{
    char *ps =s;
    do
    {
        *ps++ = (n%10) + '0';
        n/=10;
    }
    while(n != 0);
    *ps = '\0';
    strrev(s);
}
```

```
int ispalindrome(int n)
{
    char s1[100],s2[100];

    itoa(s1,n);
    strcpy(s2,s1);
    strrev(s2);

    return (strcmp(s1, s2) == 0) ? 1:0;
}
```

# Dynamically Allocated String

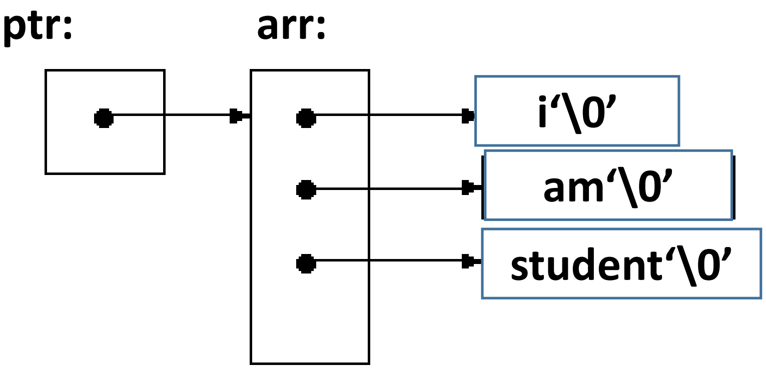
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main()
{
  char *str;
  int n;

  scanf("%d", &n);

  str = (char *)malloc(sizeof(char) * (n+1));

  strcpy(str, “hello");

  *str = 'H'; ✓

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

  free(str);

  return 1;
}
```

extra 1 for the *null character*

array and its content are strings.

```
char *arr[3] = {"i","am","student"};
char *(*ptr)[3] = &arr;
```

pointer to array of string of size 3



```
#include<stdio.h>
int main()
{
  char *arr[3] = {"i","am","student"};
  char *(*ptr)[3] = &arr;

  printf("%s %s %s\n", arr[0], arr[1], arr[2]);
  printf("%s %s %s\n", (*ptr)[0], (*ptr)[1], (*ptr)[2]);
  printf("%s %s %s\n", **ptr, *(*ptr+1), *(*ptr+2));

  printf("%p %p %p\n",&arr[0],&arr[1],&arr[2]);
  printf("%p %p %p\n",&(*ptr)[0],&(*ptr)[1],&(*ptr)[2]);
  printf("%p %p %p\n",arr,arr+1,arr+2);
  printf("%p %p %p\n",*ptr,*ptr+1,*ptr+2);

  printf("%s\n",++(*ptr)[1]);
  printf("%s\n",++(*ptr)[2]);
  printf("%s\n",(*ptr)[2]);

return 0;
}
```

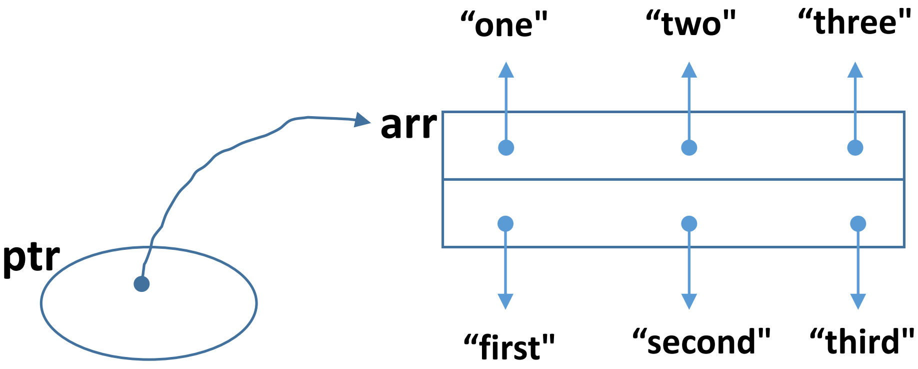
**Output:**

```
i am student
i am student
i am student
0xbf9f5090 0xbf9f5094 0xbf9f5098
0xbf9f5090 0xbf9f5094 0xbf9f5098
0xbf9f5090 0xbf9f5094 0xbf9f5098
0xbf9f5090 0xbf9f5094 0xbf9f5098
m
tudent
tudent
```

2D array and its content are strings.

```
char *arr[2][3] = {
                 {"one","two","three"},

{"first","second","third"}
                 };
 char *(*ptr)[3] = arr;
```

pointer to array of string of size 3



```c
#include<stdio.h>
int main()
{
  char *arr[2][3] = {
                     {"one","two","three"},
                     {"first","second","third"}
                    };
  char *(*ptr)[3] = arr;

  printf("%s %s %s\n",arr[0][0],arr[0][1],arr[1][0]);
  printf("%s %s %s\n",(*ptr)[0],(*ptr)[1],(*ptr)[3]);
  printf("%s %s %s\n",**ptr,*(*ptr+1),*(*ptr+3));

  printf("%p %p\n",arr,ptr);
  printf("%p %p\n",&arr[0][0],&arr[1][0]);
  printf("%p %p\n",&(*ptr)[0],&(*ptr)[3]);
  printf("%p %p\n",arr,arr+1);
  printf("%p %p\n",*ptr,*ptr+3);

  printf("%p %s\n",ptr,**ptr);
  ptr++;
  printf("%p %s\n",ptr,**ptr);

return 0;
}
```

Output:

```
one two first
one two first
one two first
0xbff82084 0xbff82084
0xbff82084 0xbff82090
0xbff82084 0xbff82090
0xbff82084 0xbff82090
0xbff82084 0xbff82090
0xbff82084 one
0xbff82090 first
```

# END