CHAR & CHAR ARRAY

C uses ***char*** type to store characters and letters. However, the char type is integer type because underneath C stores integer numbers instead of characters.

In order to represent characters, the computer has to map each integer with a corresponding character using a numerical code. The most common numerical code is **ASCII**, which stands for American Standard Code for Information Interchange. The following table illustrates the ASCII code: http://www.asciitable.com/

In C, the char type has a 1-byte unit of memory so it is more than enough to hold the ASCII codes. Besides ASCII code, there are various numerical codes available such as extended ASCII codes. Unfortunately, many character sets have more than 127 even 255 values. Therefore, to fulfill those needs, the **Unicode** was created to represent various available character sets. Unicode currently has over 40,000 characters.

```c
#include <stdio.h>

char ch;

int main(void)
{
  ch = 'A'; // ch = 65;
  printf("%c\n", ch); // %c for chars
  printf("%d\n", ch); // %d for int, prints ASCII code of a character
  return 0;
}
```

How to convert to upper-case and lower-case.
Convert char to upper-case: ***toupper***(ch);
Convert char to lower-case: ***tolower***(ch);
Both functions are declared in <ctype.h>

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

char ch;

int main(void)
{
  ch = 'a';
  printf("%c\n",ch); // %c format is used for chars

  ch = toupper(ch); // to upper-case
  printf("%c\n",ch);

  ch = tolower(ch); // to lower-case
  printf("%c\n",ch);
  return 0;
}
```

Print the characters from ‘a’ to ‘z’:

```c
#include <stdio.h>

char ch;

int main(void)
{
  for(ch = 'a'; ch <= 'z'; ch++)
    printf("%c",ch);
  printf("\n");
  return 0;
}
```

E-OLYMP 8610. Previous and next letter Given letter of English alphabet. Print its previous and next letter.

Sample input	Sample output
D	C E

► Let `char ch` is current letter. Print as a letter the value of (*ch* – 1) and (*ch* + 1). Use %c format for printitng character, not integer.

Char array

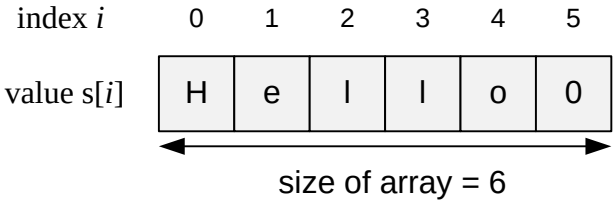
String is a sequence of characters that is treated as a single data item and terminated by null character '\0' Remember that C language *does not support strings* as a data type. A string is actually one-dimensional array of characters in C language. These are often used to create meaningful and readable programs.

For example: The string “hello” contains 6 characters including '\0' character which is automatically added by the compiler at the end of the string.

Declaring and Initializing a string variables

There are different ways to initialize a character array variable.

```c
#include <stdio.h>

char s1[6] = "Hello";
char s2[6] = { 'H','e','l','l','o','\0' };

int main(void)
{
  printf("%s\n", s1); // %s is used for printing strings
  printf("%s\n", s2);

  for(int i = 0; i < 6; i++)
    printf("%c", s1[i]);
  printf("\n");
  return 0;
}
```



Input function **scanf**() can be used with **%s** format specifier to read a string input from the terminal. But there is one problem with scanf() function, it terminates its input on the first white space it encounters. Therefore if you try to read an input string "Hello World" using scanf() function, it will only read Hello and terminate after encountering white spaces.

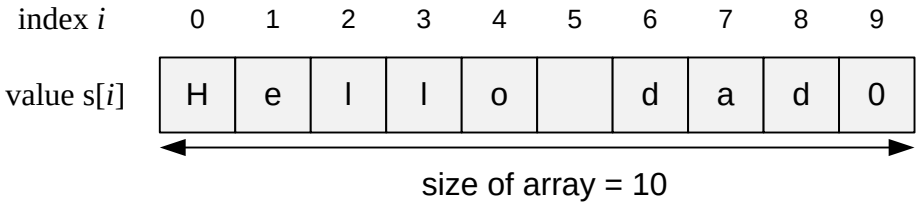
Another method to read character string with white spaces from terminal is by using the **gets**() function. **Warning:** use gets_s() for Visual Studio 2015 and more.

```
#include <stdio.h>

char s[100];

int main(void)
{
  scanf("%s", s); // read one word only
  //gets(s); // read the whole line
  printf("%s\n", s);
  return 0;
}
```

Consider the char array os size 10:

```
char s[10];
```

Read the line till the end: **gets**(s);
Print the string: **puts**(s);




```
#include <stdio.h>

char s[100];

int main(void)
{
  gets(s); // Hello dad
  puts(s);
  return 0;
}
```

C++ style (read one word only):

```
#include <iostream>
#include <string>
```

```
using namespace std;

string s;

int main(void)
{
 cin >> s; // read one word only
 cout << s;
 return 0;
}
```

C++ style (read line with spaces):

```
#include <iostream>
#include <string>
using namespace std;

string s;

int main(void)
{
  getline(cin,s); // read line with spaces
  cout << s;
  return 0;
}
```

	C	C++	
	#include <stdio.h> char s[100];	#include <iostream> string s;	This is a dog
read one word	scanf("%s",s);	cin >> s;	This
read all line	gets(s);	getline(cin,s);	This is a dog
print a string	printf("%s",s); puts(s);	cout << s;	

If you want to print ***string*** with **stdio.h** functions, you need to extract the **char array** out of the string object using ***c_str()*** method.

```
#include <iostream>
#include <cstdio>
#include <string>
using namespace std;

string s;

int main(void)
{
 getline(cin, s); // read line with spaces
 puts(s.c_str());
 printf("%s\n",s.c_str());
 return 0;
}
```

E-OLYMP 8569. String length Given a string. Print it and its length.

Sample input	Sample output
Programming Principles 1	Programming Principles 1 24

► To find the length of the string you can use the function ***strlen***(s) that is declared in the library <string.h>

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

char s[110];

int main(void)
{
  gets(s);
  puts(s);
  printf("%d\n", strlen(s));
  return 0;
}
```

C++ style (line with spaces):

```
#include <iostream>
#include <string>
using namespace std;

string s;

int main(void)
{
  getline(cin, s);
  cout << s << endl;
  cout << s.length() << endl;
  return 0;
}
```

E-OLYMP 8571. Count the letters Given a string *s* and a lowercase letter *c*. How many times the letter *c* appears in the string *s*? Uppercase and lowercase letter considered the same. For example, 'a' and 'A' considered the same letters.

Sample input	Sample output
Programming Principles 1 p	3

► Sample input string contains 3 letter p: two uppercase ‘P’ and one lowercase ‘p’. Letter *c* is lowercase, so let’s convert each letter of the string *s* to lowercase and then compare it with letter *c*.

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

```
char ch, s[100];
int i, cnt;

int main(void)
{
  gets(s);
  scanf("%c", &ch);

  cnt = 0;
  for (i = 0; i < strlen(s); i++)
    if (tolower(s[i]) == ch) cnt++;

  printf("%d\n", cnt);
  return 0;
}
```

C++ style:

```
#include <iostream>
#include <cctype>
#include <string>
using namespace std;

char ch;
string s;
int i, cnt;

int main(void)
{
  getline(cin,s);
  cin >> ch;

  cnt = 0;
  for (i = 0; i < s.length(); i++)
    if (tolower(s[i]) == ch) cnt++;

  cout << cnt;
}
```

E-OLYMP 8319. Simple calculator Input string contains an expression with one math operator (+, -, *, /). Find the value of expression.

Sample input	**Sample output**
3 * 12	36

► Read integer, char, integer. Depending on type of operation (char can be one of +, -, *, /) calculate the answer.

```
#include <stdio.h>

int a, b, res;
char c;

int main(void)
{
```

```
  scanf("%d %c %d", &a, &c, &b);

  if (c == '+') res = a + b;
  if (c == '-') res = a - b;
  if (c == '*') res = a * b;
  if (c == '/') res = a / b;

  printf("%d\n", res);
  return 0;
}
```

C++ style:

```
#include <iostream>
#include <string>
using namespace std;

int a, b, res;
char c;

int main(void)
{
  cin >> a >> c >> b;
  if (c == '+') res = a + b;
  if (c == '-') res = a - b;
  if (c == '*') res = a * b;
  if (c == '/') res = a / b;
  cout << res << endl;
  return 0;
}
```

Switch implementation:

```
#include <stdio.h>

int a, b, res;
char c;

int main(void)
{
  scanf("%d %c %d", &a, &c, &b);

  switch (c)
  {
  case '+':
    res = a + b;
    break;
  case '-':
    res = a - b;
    break;
  case '*':
    res = a * b;
    break;
  case '/':
    res = a / b;
  }
```

```
  printf("%d\n", res);
  return 0;
}
```

E-OLYMP **909. Number of words** Find the number of words in the given text.

Sample input	**Sample output**
Hello world! The country!	4

► Read word after word till the end of file and count the number of words.

```
#include <stdio.h>

char s[100];
int cnt;

int main(void)
{
  cnt = 0;
  while(scanf("%s",s) == 1)
    cnt++;

  printf("%d\n",cnt);
  return 0;
}
```

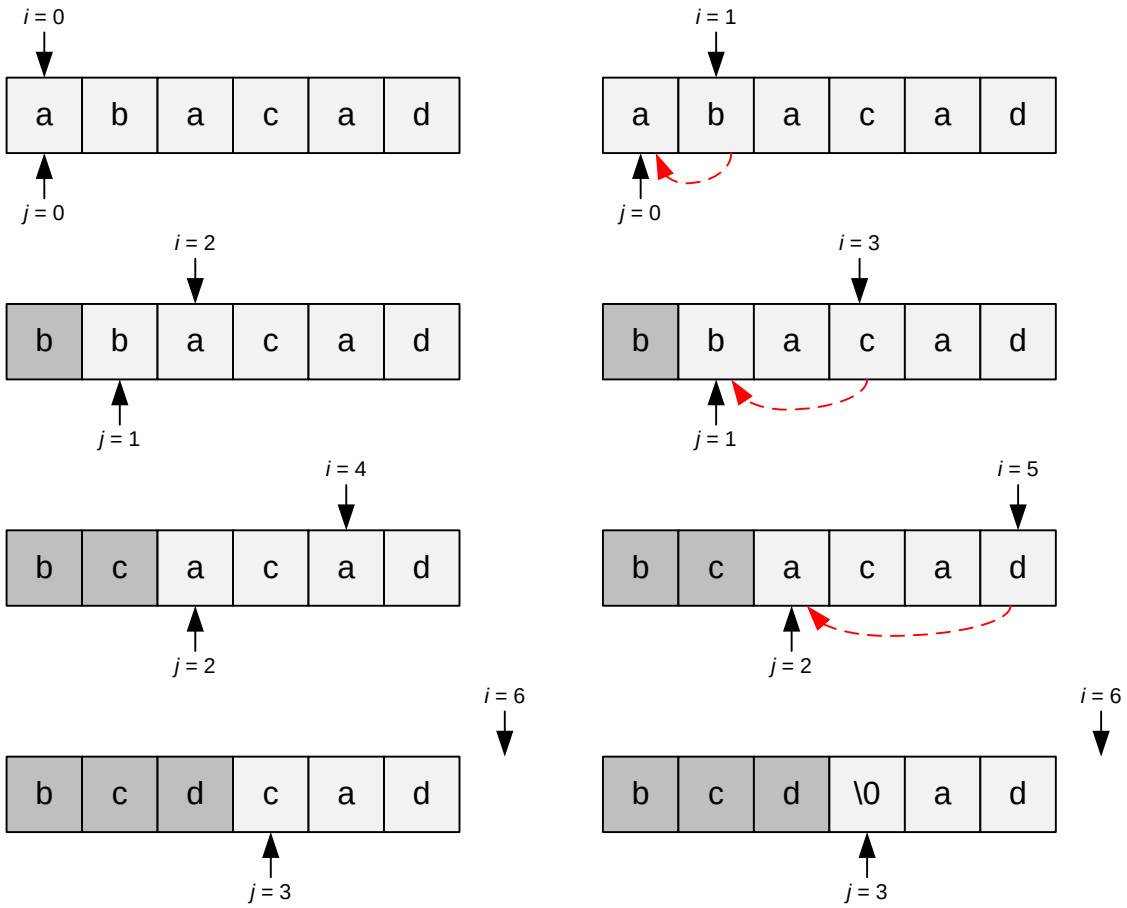
C++ style:

```
#include <iostream>
#include <string>
using namespace std;

string s;
int cnt;

int main(void)
{
  cnt = 0;
  while (cin >> s) cnt++;
  cout << cnt << endl;
  return 0;
}
```

E-OLYMP **8985. Delete the letter** Delete all small Latin letters ***a*** from the given string.

Sample input	**Sample output**
abrakadabra	brkdbr

► Declare two pointers to the beginning of the array: $i = j = 0$. Move the pointer i through the letters of the string. For each letter of string s that is not equal to 'a', copy s[i] to s[j] and move j one position forward.



Algorithm realization

Declare char array.

```
char s[1001];
```

Read the input string.

```
fgets(s, sizeof(s), stdin);
```

Letters other than '*a*' move to the left.

```
int j = 0;
for (int i = 0; i < strlen(s); i++)
  if (s[i] != 'a') s[j++] = s[i];
```

At the end of the resulting string put 0 byte.

```
s[j] = 0;
```

Print the answer.

```
puts(s);
```

C++ style:

Declare a new string *res*. If s[*i*] ≠ '*a*', then append s[*i*] to *res*. Plus operation (+) for strings is a concatenation.

```
#include <iostream>
#include <string>
using namespace std;

string s, res;
int i;

int main(void)
{
  getline(cin, s);
  for (i = 0; i < s.length(); i++)
    if (s[i] != 'a') res = res + s[i]; // concatenation
  cout << res;
  return 0;
}
```

E-OLYMP **1427. Calculator** Find the value of expression with numbers and + / - operations.

Sample input	**Sample output**
1+22-3+4-5+123	142

► Read the first number into variable *res*. Read both operation and number till the end of file. Do the operation.

```
#include <stdio.h>

int res, x;
char ch;

int main(void)
{
  scanf("%d", &res);
  while (scanf("%c%d", &ch, &x) == 2)
    if (ch == '+') res += x; else res -= x;
  printf("%d\n", res);
  return 0;
}
```

C++ style:

```
#include <iostream>
using namespace std;

int res, a;
char ch;

int main(void)
{
  cin >> res;
  while (cin >> ch >> a)
    if (ch == '+') res += a; else res -= a;
  cout << res << endl;
  return 0;
```

```
}
```

String substring

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

Copies the string pointed by *src* to *dest*.

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

char s[100] = "This is a tree";
char res[101];

int main(void)
{
  strcpy(res, s);
  puts(res);
  return 0;
}
```

```
char * strncpy ( char * destination, char * source, size_t num );
```

Copies the first *num* characters of *source* to *destination*. If the end of the source C string (which is signaled by a null-character) is found before *num* characters have been copied, destination is padded with zeros until a total of *num* characters have been written to it.

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

char s[100] = "This is a tree";
char res[101];

int main(void)
{
  strncpy(res, s + 5, 2); // is
  puts(res);
  strncpy(res, &s[10], 4); // tree
  puts(res);
  return 0;
}
```

String has a method ***substr*** to find the substring of a given string.

- `substr(int pos, int len)` – returns the substring of length *len* starting at position *pos*.
- `substr(int pos)` – returns the substring starting at position *pos* till the end.

E-OLYMP **8222. The length of a substring** Given a string *s* and two positions *a* and *b*. Print the length of a substring s[*a*..*b*] and substring itself. The numbering of characters in the string starts from 1.

► Declare the resulting string *res*. Copy the substring S[*i*..*j*] into it using the ***strncpy*** function.

Declare the input string *s* and resulting string *res*.

```
char s[101], res[101];
```

Read the input data.

```
fgets(s, sizeof(s), stdin);
scanf("%d %d", &i, &j);
```

Copy s[*i* … *j*] to *res*. Indexing in char array starts from 0. The indices in the problem statement are specified starting from 1. Therefore, in reality s[*i* – 1 … *j* – 1] is copied to *res*. In total *j* – *i* + 1 letters are copied.

```
strncpy(res, s + i - 1, j - i + 1);
```

Print the answer.

```
printf("%d\n", j - i + 1);
puts(res);
```

C++ style:

To make the numbering of characters in the string from 0, subtract 1 from *a* and *b*. Use ***substr*** method.

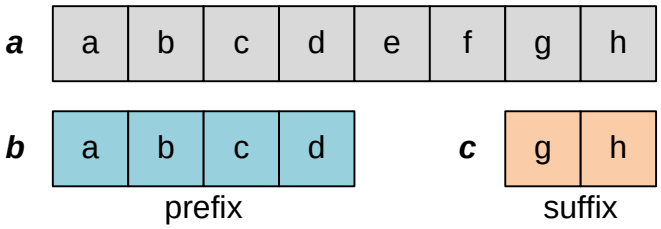
```
#include <iostream>
#include <string>
using namespace std;

int i, j;
string s, res;

int main(void)
{
  cin >> s;
  cin >> i >> j;
  i--; j--;
  res = s.substr(i, j - i + 1);
  cout << res.length() << endl;
  cout << res << endl;
  return 0;
}
```

E-OLYMP **9626. startsWith endsWith** Three lines of characters is given. Check if the second line is a prefix of the first line. Check if the third line is a suffix of the first line.

► Function

```
char *strstr(const char *haystack, const char *needle)
```

finds the first occurrence of the substring *needle* in the string *haystack*.



String *b* is a prefix of *a* if *b* appears in *a* starting at position 0.
String *c* is a suffix of *a* if *c* appears in *a* starting at position
a + strlen(*a*) – strlen(*c*)

Declare three char arrays.

```
char a[101], b[101], c[101];
```

Read three input strings.

```
gets(a);
gets(b);
gets(c);
```

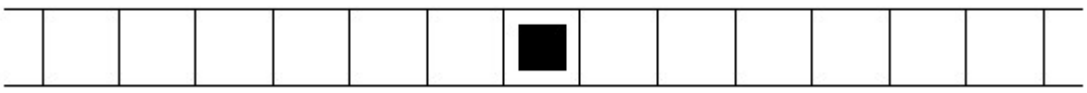
String *b* is a prefix of *a* if *b* appears in *a* starting at position 0.

```
if (strstr(a, b) == a)
  puts("true");
else
  puts("false");
```

String *c* is a suffix of *a* if *c* appears in *a* starting at position *a* + strlen(*a*) – strlen(*c*).

```
if (strstr(a + strlen(a) - strlen(c), c) == a + strlen(a) - strlen(c))
  puts("true");
else
  puts("false");
```

E-OLYMP **87. Robot** The infinite in both directions stripe with width 1 is divided into blocks of size 1 * 1. In one of these blocks the robot is located. It can move from one cell to another (the robot at the figure is marked with square). Its movements are determined by the program, each instruction is given by one of three capital letters: L, R, S. The instruction L says the robot to move one cell to the left, the instruction R – to move one square right, and S – to stay in the same cell. Program execution means the sequential execution of all instruction in it.



Write a program that will determine how many different cells visits the robot.

► Let the robot initially be in the cell with number 0. Simulate its movements, memoizing the numbers of the leftmost *l* and rightmost *r* cells where it could get. Then the number of different cells that the robot will visit while executing its program will be *r* – *l* + 1.

E-OLYMP 504. Parking You want to park the car guests who have come to the party, on the street. According to the rules the cars cannot park:

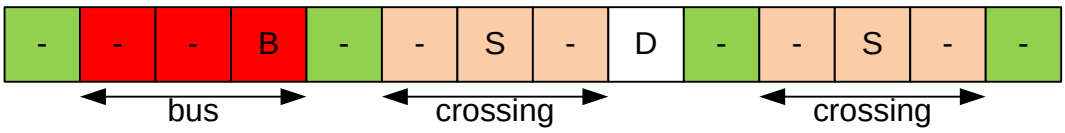
1. In front of the private departure;
2. At the bus stop and less than 10 meters before it;
3. At the pedestrian crossing, and less than 5 meters before him or after him.

You made plans for the surrounding streets, smashing them into sections the length of 5 meters (this is the minimum length for parking). Land with departure on the plane is denoted by 'D', bus stops – 'B', transitions – 'S', others – '-'. Write a program that for each street will determine the number of parking spaces.

► Iterate over all possible sections of the street of 5 meters long and check is it possible to park a car there. Let the street plan be stored in a character array *s*. Then it is possible to park on the section s[*i*] if the following conditions are simultaneously satisfied:

- s[*i*] = '-', the section is free;
- s[*i* – 1] ≠ 'S' и s[*i* + 1] ≠ 'S', there is no pedestrian crossing nearby;
- s[*i* + 1] ≠ 'B' и s[*i* + 2] ≠ 'B', there is no bus stop up to 10 meters ahead. Note that, according to condition of the problem, you can park right behind the bus stop;

For the first example, consider all possible parking positions. They are marked in green.



Store the street plan in string *s*.

```
char s[100];
```

For each test case read the street plan into a character array *s*, starting from the first position (to simplify the processing).

```
scanf("%d\n",&n);
while(n--)
{
  gets(s+1); res = 0;
  for(i = 1; i <= strlen(s+1); i++)
```

For each section s[*i*] check is it possible to park on it.

```
    if ((s[i] == '-') && (s[i+1] != 'S')  && (s[i-1] != 'S') &&
        (s[i+1] != 'B')  && (s[i+2] != 'B')) res++;
```

Print the number of possible parking spaces on the current street.

```
  printf("%d\n",res);
}
```

Compare strings

```
int strcmp ( const char * str1, const char * str2 );
```

Function starts comparing the first character of each string. If they are equal to each other, it continues with the following pairs until the characters differ or until a terminating null-character is reached.

Returns an integral value indicating the relationship between the strings:

return value	indicates
<0	the first character that does not match has a lower value in *str1* than in *str2*
0	the contents of both strings are equal
>0	the first character that does not match has a greater value in *str1* than in *str2*

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

char s1[100] = "Apple";
char s2[100] = "Apple";

int main()
{
  int res = strcmp(s1,s2);
  printf("%d", res);
  return  0;
}
```

```
int strncmp ( const char * str1, const char * str2, size_t num );
```

Compares up to *num* characters of the C string *str1* to those of the C string *str2*. This function starts comparing the first character of each string. If they are equal to each other, it continues with the following pairs until the characters differ, until a terminating null-character is reached, or until *num* characters match in both strings, whichever happens first.

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

char s1[100] = "abcdefg";
char s2[100] = "abcxyz";

int main()
{
  int res = strncmp(s1, s2, 3);
  printf("%d", res);
  return  0;
}
```

Concatenate strings

```
char * strcat ( char * destination, const char * source );
```

Appends a copy of the source string to the destination string. The terminating null character in destination is overwritten by the first character of source, and a null-character is included at the end of the new string formed by the concatenation of both in destination. Destination and source shall not overlap. Destination is returned.

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

char s1[100] = "Orange";
char s2[100] = "Apple";

int main()
{
  strcat(s1,s2);
  printf("%s", s1);
  return  0;
}
```