Day15.C

C String (2)

Young W. Lim

December 9, 2017

0.1 C string functions

```
::::::::::::::
t1.c
::::::::::::::
#include <stdio.h>
#include <stdlib.h>

int main(void) {
  const char  *str = "51.2 32.3% will be accepted.";
  const char  *ptr = str + 5;
        char  *p;
  double a;
  long   l;

  printf("str= %s \n", str); // (1)
  printf("ptr= %s \n", ptr); // (2)

/*-----
  *ptr= 'X';    // (3)

  printf("str= %s \n", str);
  printf("ptr= %s \n", ptr);
-----*/

  a = strtod(str, &p);

  printf("str= %s \n", str);  // (4)
  printf("ptr= %s \n", p);  // (5)
  printf("a= %lf \n", a); // (6)

  l = strtol(str, &p, 0);

  printf("str= %s \n", str);  // (7)
  printf("ptr= %s \n", p); // (8)
  printf("l= %ld \n", l); // (9)

}


::::::::::::::
t1.out
::::::::::::::
str= 51.2 32.3% will be accepted.
ptr= 32.3% will be accepted.
str= 51.2 32.3% will be accepted.
ptr=  32.3% will be accepted.
a= 51.200000
str= 51.2 32.3% will be accepted.
ptr= .2 32.3% will be accepted.
```

```
l= 51
```

c strings and string functions

- (2) prints : ptr= 32.3% will be accepted.
  - this is because ptr = str+5
  - ptr passes over the first 5 characters : 51.2 (including a space character)
  - ptr points to the first character 3 of the string "32.3% will be accepted."
- uncommenting (3)
  - because ptr points to the part of a string constant
  - any of its elements cannot be modified
  - *ptr= 'X' attempts to modify the first character 3 into X
  - this causes an error messages
  - t1.c:14:7: error: assignment of read-only location *ptr
- removing const qualifiers
  - enables a compilation
  - but produces a run-time error : Segmentation Fault
- (5) prints : ptr= 32.3% will be accepted.
  - it is the string pointed by p not by ptr
  - the correct printout : p= 32.3% will be accepted.
  - before calling to strtod(), p was not initialized
  - therefore strtod() assigned the resulting string to p
- double strtod(const char *str, char **endptr)
  - Parameters
    - str This is the value to be converted to a string.
    - endptr This is the reference to an already allocated object of type char*, whose value is set by the function to the next character in str after the numerical value.
  - Return Value
    - This function returns the converted floating point number as a double value, else zero value (0.0) is returned.
  - `https://www.tutorialspoint.com/c_standard_library/c_function_strtod.htm`
- strtod : string-to-double
  - extracting a double number from a string

  - returning the remaining string via endptr
  - pass by reference – need one more *
  - the type of p is char *
  - to pass by reference, use &p whose type is char **
  - now strtod can modify the content of p, that is the starting address of a string
  - extracting one double number 51.2 from the original string "51.2 32.3% will be accepted."
  - after removing also the delemiter character (space)
  - p will points to the remaining string "32.3% will be accepted."
  - as a default integer type is int, the default type of floating point numbers is double
  - but the conversion specifier %lf must be used with a double type number.

- long int strtol(const char *str, char **endptr, int base)
  - Parameters
    - str This is the string containing the representation of an integral number.
    - endptr This is the reference to an object of type char*, whose value is set by the function to the next character in str after the numerical value.
    - base This is the base, which must be between 2 and 36 inclusive, or be the special value 0.
  - Return Value
    - This function returns the converted integral number as a long int value, else zero value is returned.
  - `https://www.tutorialspoint.com/c_standard_library/c_function_strtol.htm`

- strtol : string-to-long
  - extracting a long number from the string
  - can assume the number in the string in a decimal, octal, hexadecimal, etc.

- %ld or %lld for some computers
  - to print 1, 2, 4-byte integer type in decimal, use %d.
  - to print 8-byte integer type in decimal, use %ld (some computers may use %lld)

0.2 scanf()

```
::::::::::::::
h1.c
::::::::::::::
#include <stdio.h>

#define CASE 3 // 1, 2, 3, 4

#ifndef CASE
#define 1
#endif

int main(void) {

   int i,r;
   char c;


#if CASE == 1

   puts("======(a)=======================================");

   puts("Type 123(LF)--------%d---------");
   r=scanf("%d", &i);
   printf("i=%d r=%d item(s) \n\n", i, r);  // 1 ... 123

   puts("Type 123A(LF)-------%d-----%c--");
   r=scanf("%d", &i);
   printf("i=%d r=%d item(s) \n\n", i, r);  // 2 ... 123
   r=scanf("%c", &c);
   printf("c=%c r=%d item(s) \n\n", c, r);  // 3 ... A

#elif CASE == 2

   puts("======(b)=======================================");

   puts("Type 123 A(LF)------%d %c------");
   r=scanf("%d %c", &i, &c);
   printf("i=%d c=%c r=%d item(s) \n\n", i, c, r); // 1 ... 123 A

   puts("Type 123A(LF)-------%d%c------");
   r=scanf("%d%c", &i, &c);
   printf("i=%d c=%c r=%d item(s) \n\n", i, c, r); // 2 ... 123 A

   puts("Type 123A(LF)-------%d %c-----");
   r=scanf("%d %c", &i, &c);
   printf("i=%d c=%c r=%d item(s) \n\n", i, c, r); // 3 ... 123 A

   puts("Type 123   A(LF)----%d %c------");
```

```
   r=scanf("%d %c", &i, &c);
   printf("i=%d c=%c r=%d item(s) \n\n", i, c, r); // 4 ... 123 A

   puts("Type 123(LF)--------%d%c-------");
   r=scanf("%d%c", &i, &c);
   printf("i=%d c=%c r=%d item(s) \n\n", i, c, r); // 5 ... 123 (LF)


#elif CASE == 3

   puts("======(c)=======================================");

   puts("Type 123(LF)(LF)1(LF)----%d\\n-------");
   r=scanf("%d ", &i);
   printf("i=%d r=%d item(s) \n\n", i, r);  // 1 ... 123

   puts("Type 123(LF)1(LF)--------%d\\n-------");
   r=scanf("%d ", &i);
   printf("i=%d r=%d item(s) \n\n", i, r);  // 2 ... 1

   puts("Type 123 a b c (LF)-------%d\\n-------");
   r=scanf("%d ", &i);
   printf("i=%d r=%d item(s) \n\n", i, r);  // 3 ... 123

   r=scanf("%d", &i);
   printf("i=%d r=%d item(s) \n\n", i, r);  // 4 ... 1

   r=scanf("%d", &i);
   printf("i=%d r=%d item(s) \n\n", i, r);  // 5 ... 123

   r=scanf(" %c", &c);
   printf("c=%c r=%d item(s) \n\n", c, r);  // 6 ... a

   r=scanf(" %c", &c);
   printf("c=%c r=%d item(s) \n\n", c, r);  // 7 ... b

   r=scanf(" %c", &c);
   printf("c=%c r=%d item(s) \n\n", c, r);  // 8 ... c

#elif CASE == 4

   puts("======(d)=======================================");

   puts("Type 123(LF)(LF)1(LF)----%d\\n-------");
   r=scanf("%d\n", &i);
   printf("i=%d r=%d item(s) \n\n", i, r);   // 1 ... 123

   puts("Type 123(LF)1(LF)--------%d\\n-------");
   r=scanf("%d\n", &i);
   printf("i=%d r=%d item(s) \n\n", i, r);   // 2 ... 1
```

```
   puts("Type 123 a b c (LF)-------%d\\n-------");
   r=scanf("%d\n", &i);
   printf("i=%d r=%d item(s) \n\n", i, r);   // 3 ... 123

   r=scanf("%d\n", &i);
   printf("i=%d r=%d item(s) \n\n", i, r);   // 4 ... 1

   r=scanf("%d\n", &i);
   printf("i=%d r=%d item(s) \n\n", i, r);   // 5 ... 123

#else
   puts("CASE = 1, 2, 3, or 4 ");
#endif

}


:::::::::::::::
h1.out
:::::::::::::::
```

```
======(a)=====================================
Type 123(LF)--------%d---------
123
i=123 r=1 item(s)

Type 123A(LF)-------%d-----%c--
123A
i=123 r=1 item(s)

c=A r=1 item(s)
```

```
======(b)=====================================
Type 123 A(LF)------%d %c------
123 A
i=123 c=A r=2 item(s)

Type 123A(LF)-------%d%c------
123A
i=123 c=A r=2 item(s)

Type 123A(LF)-------%d %c-----
123A
i=123 c=A r=2 item(s)

Type 123   A(LF)----%d %c------
123   A
i=123 c=A r=2 item(s)

Type 123(LF)--------%d%c-------
123
i=123 c=
 r=2 item(s)
```

```
======(c)=====================================
Type 123(LF)(LF)1(LF)----%d\n-------
123

1
i=123 r=1 item(s)

Type 123(LF)1(LF)--------%d\n-------
123
i=1 r=1 item(s)

Type 123 a b c (LF)-------%d\n-------
1
i=123 r=1 item(s)

i=1 r=1 item(s)

123 a b c
i=123 r=1 item(s)

c=a r=1 item(s)

c=b r=1 item(s)

c=c r=1 item(s)
```

```
======(d)=====================================
Type 123(LF)(LF)1(LF)----%d\n-------
123

1
i=123 r=1 item(s)

Type 123(LF)1(LF)--------%d\n-------
123
d=1 r=1 item(s)

Type 123 a b c (LF)-------%d\n-------
1
i=123 r=1 item(s)

123 a b c
i=1 r=1 item(s)

i=123 r=1 item(s)
```

case (a)

1.
   - `r=scanf("%d", &i);`
   - input stream : 123 ↵
   - `%d` matches `123`
   - `printf("i=%d r=%d item(s) \n\n", i, r);`
   - output stream : i=123 r=1 item(s)↵↵

2.
   - `r=scanf("%d", &i);`
   - input stream : 123 A ↵
   - `%d` matches `123`
   - `printf("i=%d r=%d item(s) \n\n", i, r);`
   - output stream : i=123 r=1 item(s) ↵↵
   - `r=scanf("%c", &c);`
   - input stream updated : A ↵
   - `%c` matches `A`
   - `printf("c=%c r=%d item(s) \n\n", c, r);`
   - output stream : c=A r=1 item(s) ↵↵

case (b)

1. `r=scanf("%d %c", &i, &c));`
   - input stream : 123 ␣ A ↵
   - %d matches 123
   - " " matches " "
   - %c matches A
   - `printf("i=%d c=%c r=%d item(s) \n\n", i, c, r);`
   - output stream : i=123 c=A r=2 item(s)↵↵

2. `r=scanf("%d%c", &i, &c));`
   - input stream : 123 A ↵
   - %d matches 123
   - %c matches A
   - `printf("i=%d c=%c r=%d item(s) \n\n", i, c, r);`
   - output stream : i=123 c=A r=2 item(s)↵↵

3. `r=scanf("%d %c", &i, &c));`
   - input stream : 123 A ↵
   - %d matches 123
   - " " matches nothing
   - %c matches A
   - `printf("i=%d c=%c r=%d item(s) \n\n", i, c, r);`
   - output stream : i=123 c=A r=2 item(s)↵↵

4. `r=scanf("%d %c", &i, &c));`
   - input stream : 123 ␣␣␣ A ↵
   - %d matches 123
   - " " matches "   "
   - %c matches A
   - `printf("i=%d c=%c r=%d item(s) \n\n", i, c, r);`
   - output stream : i=123 c=A r=2 item(s)↵↵

5. `r=scanf("%d%c", &i, &c));`
   - input stream : 123 ↵
   - %d matches 123
   - %c matches ↵
   - `printf("i=%d c=%c r=%d item(s) \n\n", i, c, r);`
   - output stream : i=123 c=↵ r=2 item(s) ↵↵

case (c)

1.
   - `r=scanf("%d ", &i);`
   - input stream : 123 ↵ ↵ 1 ↵
   - `%d` matches 123
   - `" "` matches ↵ ↵
   - `printf("i=%d r=%d item(s) \n\n", i, r);`
   - output stream : i=123 r=1 item(s)↵↵

2.
   - `r=scanf("%d ", &i);`
   - input stream updated : 1 ↵ , 123 ↵
   - `%d` matches 1
   - `" "` matches ↵
   - `printf("i=%d r=%d item(s) \n\n", i, r);`
   - output stream : i=1 r=1 item(s)↵↵

3.
   - `r=scanf("%d ", &i);`
   - input stream updated: 123 ↵, 1 ↵
   - `%d` matches 123
   - `" "` matches ↵
   - `printf("i=%d r=%d item(s) \n\n", i, r);`
   - output stream : i=123 r=1 item(s)↵↵

4.
   - `r=scanf("%d", &i);` `"%d"`
   - input stream updagted: 1 ↵ (without additional input)
   - `%d` matches 1
   - `printf("i=%d r=%d item(s) \n\n", i, r);`
   - output stream : i=1 r=1 item(s)↵↵

5.
   - `r=scanf("%d", &i);`
   - input stream : 123 ␣ a ␣ b ␣ c ␣ ↵
   - `%d` matches 123
   - `printf("i=%d r=%d item(s) \n\n", i, r);`
   - output stream : i=123 r=1 item(s) ↵↵

6.
- `r=scanf(" %c", &c);`
- input stream updated: ␣ a ␣ b ␣ c ␣ ↵
- `" "` matches `" "`
- `%c` matches a
- `printf("c=%c r=%d item(s) \n\n", c, r);`
- output stream : c=a r=1 item(s) ↵↵

7.
- `r=scanf(" %c", &c);`
- input stream updated: ␣ b ␣ c ␣ ↵
- `" "` matches `" "`
- `%c` matches b
- `printf("c=%c r=%d item(s) \n\n", c, r);`
- output stream : c=b r=1 item(s) ↵↵

8.
- `r=scanf(" %c", &c);`
- input stream updated: ␣ c ␣ ↵
- `" "` matches `" "`
- `%c` matches c
- `printf("c=%c r=%d item(s) \n\n", c, r);`
- output stream : c=c r=1 item(s) ↵↵

case (d)

1.
   - `r=scanf("%d\n", &i);`
   - input stream : 123 ↵ ↵ 1 ↵
   - `%d` matches 123
   - `\n` matches ↵ ↵
   - `printf("i=%d r=%d item(s) \n\n", i, r);`
   - output stream : i=123 r=1 item(s)↵↵

2.
   - `r=scanf("%d\n", &i);`
   - input stream updated : 1 ↵ , 123 ↵
   - `%d` matches 1
   - `\n` matches ↵
   - `printf("i=%d r=%d item(s) \n\n", i, r);`
   - output stream : i=1 r=1 item(s)↵↵

3.
   - `r=scanf("%d\n", &i);`
   - input stream updated: 123 ↵, 1 ↵
   - `%d` matches 123
   - `\n` matches ↵
   - `printf("i=%d r=%d item(s) \n\n", i, r);`
   - output stream : i=123 r=1 item(s)↵↵

4.
   - `r=scanf("%d\n", &i);` `"%d\n"`
   - input stream updagted: 1 ↵ , 123 ␣ a ␣ b ␣ c ␣ ↵
   - `%d` matches 1
   - `\n` matches ↵
   - `printf("i=%d r=%d item(s) \n\n", i, r);`
   - output stream : i=1 r=1 item(s)↵↵

5.
   - `r=scanf("%d\n", &i);`
   - input stream updated : 123 ␣ a ␣ b ␣ c ␣ ↵
   - `%d` matches 123
   - `\n` matches ↵
   - `printf("i=%d r=%d item(s) \n\n", i, r);`
   - output stream : i=123 r=1 item(s) ↵↵

0.3 printf()

```
::::::::::::::
h2.c
::::::::::::::
#include <stdio.h>
#include <math.h>

//==========(0)==============================
#ifndef PI
#define PI 3.14159265358979323846
#endif

#define PIL 3.14159265358979323846L

// most compiler in <math.h>
// #define M_PI 3.14159265358979323846
//                1234567890123456789O

int main(void) {

  printf("--------------------------\n");
  printf("      1234567890123456789O\n");
  printf("      ...................\n");
  printf("PI= 3.14159265358979323846\n");
  printf("--------------------------\n");


//==========(a)==============================
  printf("PI= %f   \t\t\t\t%%f\n", PI);
  printf("PI= %e   \t\t\t%%e\n", PI);
  printf("PI= %g   \t\t\t\t%%g\n\n", PI);

//==========(b)==============================
  printf("PI= %.20f \t\t%%.20f\n", PI);
  printf("PI= %.20e \t\t%%.20e\n", PI);
  printf("PI= %.20g \t\t%%.20g\n\n", PI);

//==========(c)==============================
  printf("PI= %.20f \t\t%%.20lf\n", PI);
  printf("PI= %.20e \t\t%%.20le\n", PI);
  printf("PI= %.20g \t\t%%.20lg\n\n", PI);

//==========(d)==============================
  printf("100PI= %f   \t\t\t%%f\n", PI*100);
  printf("100PI= %e   \t\t\t%%e\n", PI*100);
  printf("100PI= %g   \t\t\t%%g\n\n", PI*100);

//==========(e)==============================
  printf("100PI= %.20f \t%%.20f\n", PI*100);
```

```
  printf("100PI= %.20e \t%%.20e\n", PI*100);
  printf("100PI= %.20g \t\t%%.20g\n\n", PI*100);

//==========(f)==============================
  printf("100PI= %.20f \t%%.20f\n", PI*100);
  printf("100PI= %.20e \t%%.20e\n", PI*100);
  printf("100PI= %.20g \t\t%%.20g\n\n", PI*100);

//==========( )==============================
  printf("PI= %.22Lf \t\t%%.22Lf\n", PIL);
  printf("PI= %.22Le \t%%.22Le\n", PIL);
  printf("PI= %.22Lg \t\t%%.23Lg\n", PIL);


}
```

```
::::::::::::::
h2.out
::::::::::::::
--------------------------
      12345678901234567890
      .....................
PI= 3.14159265358979323846
--------------------------
PI= 3.141593                              %f
PI= 3.141593e+00                          %e
PI= 3.14159                               %g

PI= 3.14159265358979311600                %.20f
PI= 3.141592653589793116000e+00           %.20e
PI= 3.14159265358979311600                %.20g

PI= 3.14159265358979311600                %.20lf
PI= 3.141592653589793116000e+00           %.20le
PI= 3.14159265358979311600                %.20lg

100PI= 314.159265                         %f
100PI= 3.141593e+02                       %e
100PI= 314.159                            %g

100PI= 314.15926535897932581065           %.20f
100PI= 3.141592653589793258110e+02        %.20e
100PI= 314.15926535897932581              %.20g

100PI= 314.15926535897932581065           %.20f
100PI= 3.141592653589793258110e+02        %.20e
100PI= 314.15926535897932581              %.20g

PI= 3.1415926535897932385128              %.22Lf
PI= 3.14159265358979323851280e+00         %.22Le
PI= 3.14159265358979323851300             %.23Lg
```

macro

- if `MACRO` is defined do the controlled text

```
#ifdef MACRO
controlled text
#endif /* MACRO */
```

- if `MACRO` is not defined do the controlled text

```
#ifndef MACRO
controlled text
#endif /* MACRO */
```

format conversion specifier

- `%e` exponential form ($m \cdot 10^e$)
  - 6 default digits after the decimal point
  - 3.1 4 1 5 9 3 e+00
- `%f` normal (fixed point) form
  - 6 default digits after the decimal point
  - 3.1 4 1 5 9 3
- `%g` either exponential or normal form
  - 6 default digits excluding decimal point
  - 3.1 4 1 5 9

double precision

- `----------------------`
- `       12345678901234567890`
- `       ....................`
- `PI= 3.14159265358979323846`
- `PI= 3.14159265358979311600` `%.20f`
- `PI= 3.14159265358979311600e+00` `%.20e`
- `PI= 3.14159265358979116` `%.20g`
- `PI= 3.14159265358979311600` `%.20lf`
- `PI= 3.14159265358979311600e+00` `%.20le`
- `PI= 3.14159265358979116` `%.20lg`

- the followings are from

  `https://en.wikipedia.org/wiki/Double-precision_floating-point_format`

- Sign bit: 1 bit
- Exponent: 11 bits
- Significand precision: 53 bits (52 explicitly stored)
- The exponent field can be interpreted as either an 11-bit signed integer from 1024 to 1023 (2's complement) or an 11-bit unsigned integer from 0 to 2047, which is the accepted biased form in the IEEE 754 binary64 definition. If the unsigned integer format is used, the exponent value used in the arithmetic is the exponent shifted by a bias for the IEEE 754 binary64 case, an exponent value of 1023 represents the actual zero (i.e. for 2^{e-1023} to be one, e must be 1023). Exponents range from 1022 to +1023 because exponents of 1023 (all 0s) and +1024 (all 1s) are reserved for special numbers.
- The 53-bit significand precision gives from 15 to 17 significant decimal digits precision ($2^{-53} \approx 1.11 \times 10^{-16}$).

- `%f` or `%lf` for `double` type numbers
- `%Lf` for `long double` type numbers

0.4 string-to-double strtod()

```
::::::::::::::
h1.c
::::::::::::::
#include <stdio.h>
#include <stdlib.h>


int main(void) {
  char S[] = "111.1 222.2 333.3 444.4 555.5 AAA BBB";
  char *p, *q, *old_p;
  double x;

  for (int i=0; i<sizeof(S); ++i)
    printf("S[%2d]= %c : %p \n", i, S[i], S+i);

  p = old_p = S; q = NULL;

  while (old_p != q) {
    printf("%p : ", p);
    x = strtod(p, &q);
    old_p = p;
    p = q;
    printf(" %f\n", x);
```

```
  }

}


::::::::::::::
h1.out
::::::::::::::
S[ 0]= 1 : 0x7fff60a93200
S[ 1]= 1 : 0x7fff60a93201
S[ 2]= 1 : 0x7fff60a93202
S[ 3]= . : 0x7fff60a93203
S[ 4]= 1 : 0x7fff60a93204
S[ 5]=   : 0x7fff60a93205
S[ 6]= 2 : 0x7fff60a93206
S[ 7]= 2 : 0x7fff60a93207
S[ 8]= 2 : 0x7fff60a93208
S[ 9]= . : 0x7fff60a93209
S[10]= 2 : 0x7fff60a9320a
S[11]=   : 0x7fff60a9320b
S[12]= 3 : 0x7fff60a9320c
S[13]= 3 : 0x7fff60a9320d
S[14]= 3 : 0x7fff60a9320e
S[15]= . : 0x7fff60a9320f
S[16]= 3 : 0x7fff60a93210
S[17]=   : 0x7fff60a93211
S[18]= 4 : 0x7fff60a93212
S[19]= 4 : 0x7fff60a93213
S[20]= 4 : 0x7fff60a93214
S[21]= . : 0x7fff60a93215
S[22]= 4 : 0x7fff60a93216
S[23]=   : 0x7fff60a93217
S[24]= 5 : 0x7fff60a93218
S[25]= 5 : 0x7fff60a93219
S[26]= 5 : 0x7fff60a9321a
S[27]= . : 0x7fff60a9321b
S[28]= 5 : 0x7fff60a9321c
S[29]=   : 0x7fff60a9321d
S[30]= A : 0x7fff60a9321e
S[31]= A : 0x7fff60a9321f
S[32]= A : 0x7fff60a93220
S[33]=   : 0x7fff60a93221
S[34]= B : 0x7fff60a93222
S[35]= B : 0x7fff60a93223
S[36]= B : 0x7fff60a93224
S[37]=
0x7fff60a93200 :  111.100000
0x7fff60a93205 :  222.200000
0x7fff60a9320b :  333.300000
0x7fff60a93211 :  444.400000
```

```
0x7fff60a93217 :  555.500000
0x7fff60a9321d :  0.000000
```

using while loop

- `p` and `q` values

starting p	string	returning q
0x7fff60a93200	`"111.1 ..."`	0x7fff60a93205
0x7fff60a93205	`" 222.2 ..."`	0x7fff60a9320b
0x7fff60a9320b	`" 333.3 ..."`	0x7fff60a93211
0x7fff60a93211	`" 444.4 ..."`	0x7fff60a93217
0x7fff60a93217	`" 555.5 ..."`	0x7fff60a9321d
0x7fff60a9321d	`" AAA ..."`	0x7fff60a9321d

- after extracting `555.5`, the remaining string will be `" AAA BBB"`
- for this string, no `double` number can be extracted
- returned `double` value will be `0.0` and `q` points to the same address as `p`
- after returning from `strtod()`, `p` must be updated with `q`
- this updated `p` (=`q`) will be compared with the old `p` in the condition of the `while` loop
- only if there is no more number to be extracted, the new `p` (=`q`) becomes the same as the old `p`
- but it is possible to eliminate `old_p` variable :

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


int main(void) {
  char S[] = "111.1 222.2 333.3 444.4 555.5 AAA BBB";
  char *p, *q;
  double x;

  for (int i=0; i<sizeof(S); ++i)
    printf("S[%2d]= %c : %p \n", i, S[i], S+i);

  p = NULL; q = S;

  while (p != q) {
    p = q;
    printf("%p : ", p);
    x = strtod(p, &q);
    printf(" %f\n", x);
  }
```

```
}
```

using do ... while loop

- `old_p` is not necessary

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


int main(void) {
  char S[] = "111.1 222.2 333.3 444.4 555.5 AAA BBB";
  char *p, *q;
  double x;

  for (int i=0; i<sizeof(S); ++i)
    printf("S[%2d]= %c : %p \n", i, S[i], S+i);

  p = q = S;

  do {
    p = q;
    printf("%p : ", p);
    x = strtod(p, &q);
    printf(" %f\n", x);
  } while (p != q);

}
```

using for loop

- `for` loop requires the knowledge of iteration numbers
- in this problem, it is safe to assume that we do not know how many numbers are present in a given string.

0.5 string-concatenate strcat()

```
::::::::::::::
h2.c
::::::::::::::
#include <stdio.h>
#include <string.h>

void prstring(char *S, char *N) {
  int   i;
  char *p;
```

```
  puts("-----------------------------------");
  p = S; i = 0;
  while (1) {
    printf("%s+%2d= %p : ", N, i, S+i);
    printf("%1c %2x\n", *(S+i), *(S+i));
    if (*p) { p++; i++; }
    else return;
  }
}



int main(void) {
  char S1[] = "Happy ";
  char S2[] = "New Year ";

  printf("S1= %s \n", S1);
  printf("S2= %s \n", S2);

  prstring(S1, "S1");
  prstring(S2, "S2");

  puts("\n.....................................");
  printf("strcat(S1,S2)= %s \n", strcat(S1, S2));
  puts(".....................................\n");

  printf("S1= %s \n", S1);
  printf("S2= %s \n", S2);

  prstring(S1, "S1");
  prstring(S2, "S2");

  puts("\n.....................................");
  puts("strcat(S1, S2);");
  puts(".....................................\n");
  strcat(S1, S2);

  printf("S1= %s \n", S1);
  printf("S2= %s \n", S2);

  prstring(S1, "S1");
  prstring(S2, "S2");
}
::::::::::::::
h2.out
::::::::::::::
S1= Happy
S2= New Year
-----------------------------------
S1+ 0= 0x7ffdcdb168b0 : H 48
```

```
S1+ 1= 0x7ffdcdb168b1 : a 61
S1+ 2= 0x7ffdcdb168b2 : p 70
S1+ 3= 0x7ffdcdb168b3 : p 70
S1+ 4= 0x7ffdcdb168b4 : y 79
S1+ 5= 0x7ffdcdb168b5 :   20
S1+ 6= 0x7ffdcdb168b6 :   0
----------------------------------
S2+ 0= 0x7ffdcdb168c0 : N 4e
S2+ 1= 0x7ffdcdb168c1 : e 65
S2+ 2= 0x7ffdcdb168c2 : w 77
S2+ 3= 0x7ffdcdb168c3 :   20
S2+ 4= 0x7ffdcdb168c4 : Y 59
S2+ 5= 0x7ffdcdb168c5 : e 65
S2+ 6= 0x7ffdcdb168c6 : a 61
S2+ 7= 0x7ffdcdb168c7 : r 72
S2+ 8= 0x7ffdcdb168c8 :   20
S2+ 9= 0x7ffdcdb168c9 :   0

.....................................
strcat(S1,S2)= Happy New Year
.....................................

S1= Happy New Year
S2= New Year
----------------------------------
S1+ 0= 0x7ffdcdb168b0 : H 48
S1+ 1= 0x7ffdcdb168b1 : a 61
S1+ 2= 0x7ffdcdb168b2 : p 70
S1+ 3= 0x7ffdcdb168b3 : p 70
S1+ 4= 0x7ffdcdb168b4 : y 79
S1+ 5= 0x7ffdcdb168b5 :   20
S1+ 6= 0x7ffdcdb168b6 : N 4e
S1+ 7= 0x7ffdcdb168b7 : e 65
S1+ 8= 0x7ffdcdb168b8 : w 77
S1+ 9= 0x7ffdcdb168b9 :   20
S1+10= 0x7ffdcdb168ba : Y 59
S1+11= 0x7ffdcdb168bb : e 65
S1+12= 0x7ffdcdb168bc : a 61
S1+13= 0x7ffdcdb168bd : r 72
S1+14= 0x7ffdcdb168be :   20
S1+15= 0x7ffdcdb168bf :   0
----------------------------------
S2+ 0= 0x7ffdcdb168c0 : N 4e
S2+ 1= 0x7ffdcdb168c1 : e 65
S2+ 2= 0x7ffdcdb168c2 : w 77
S2+ 3= 0x7ffdcdb168c3 :   20
S2+ 4= 0x7ffdcdb168c4 : Y 59
S2+ 5= 0x7ffdcdb168c5 : e 65
S2+ 6= 0x7ffdcdb168c6 : a 61
S2+ 7= 0x7ffdcdb168c7 : r 72
```

```
S2+ 8= 0x7ffdcdb168c8 :   20
S2+ 9= 0x7ffdcdb168c9 :   0

....................................
strcat(S1, S2);
....................................

S1= Happy New Year New Year
S2= ew Year
----------------------------------
S1+ 0= 0x7ffdcdb168b0 : H 48
S1+ 1= 0x7ffdcdb168b1 : a 61
S1+ 2= 0x7ffdcdb168b2 : p 70
S1+ 3= 0x7ffdcdb168b3 : p 70
S1+ 4= 0x7ffdcdb168b4 : y 79
S1+ 5= 0x7ffdcdb168b5 :   20
S1+ 6= 0x7ffdcdb168b6 : N 4e
S1+ 7= 0x7ffdcdb168b7 : e 65
S1+ 8= 0x7ffdcdb168b8 : w 77
S1+ 9= 0x7ffdcdb168b9 :   20
S1+10= 0x7ffdcdb168ba : Y 59
S1+11= 0x7ffdcdb168bb : e 65
S1+12= 0x7ffdcdb168bc : a 61
S1+13= 0x7ffdcdb168bd : r 72
S1+14= 0x7ffdcdb168be :   20
S1+15= 0x7ffdcdb168bf : N 4e
S1+16= 0x7ffdcdb168c0 : e 65
S1+17= 0x7ffdcdb168c1 : w 77
S1+18= 0x7ffdcdb168c2 :   20
S1+19= 0x7ffdcdb168c3 : Y 59
S1+20= 0x7ffdcdb168c4 : e 65
S1+21= 0x7ffdcdb168c5 : a 61
S1+22= 0x7ffdcdb168c6 : r 72
S1+23= 0x7ffdcdb168c7 :   20
S1+24= 0x7ffdcdb168c8 :   0
----------------------------------
S2+ 0= 0x7ffdcdb168c0 : e 65
S2+ 1= 0x7ffdcdb168c1 : w 77
S2+ 2= 0x7ffdcdb168c2 :   20
S2+ 3= 0x7ffdcdb168c3 : Y 59
S2+ 4= 0x7ffdcdb168c4 : e 65
S2+ 5= 0x7ffdcdb168c5 : a 61
S2+ 6= 0x7ffdcdb168c6 : r 72
S2+ 7= 0x7ffdcdb168c7 :   20
S2+ 8= 0x7ffdcdb168c8 :   0
```

memory contamination problem

- the final `strcat()` overwrites the memory locations of `S2`

- we can observe this error in the Linux
- Visual C++ does check this kind error beforehand
- this problem is only for Linux users.

```
S1+ 0= 0x7ffdcdb168b0 : H 48
S1+ 1= 0x7ffdcdb168b1 : a 61
S1+ 2= 0x7ffdcdb168b2 : p 70
S1+ 3= 0x7ffdcdb168b3 : p 70
S1+ 4= 0x7ffdcdb168b4 : y 79
S1+ 5= 0x7ffdcdb168b5 :   20
S1+ 6= 0x7ffdcdb168b6 : N 4e
S1+ 7= 0x7ffdcdb168b7 : e 65
S1+ 8= 0x7ffdcdb168b8 : w 77
S1+ 9= 0x7ffdcdb168b9 :   20
S1+10= 0x7ffdcdb168ba : Y 59
S1+11= 0x7ffdcdb168bb : e 65
S1+12= 0x7ffdcdb168bc : a 61
S1+13= 0x7ffdcdb168bd : r 72
S1+14= 0x7ffdcdb168be :   20
S1+15= 0x7ffdcdb168bf : N 4e
S1+16= 0x7ffdcdb168c0 : e 65 ........S2+ 0= 0x7ffdcdb168c0 : e 65
S1+17= 0x7ffdcdb168c1 : w 77 ........S2+ 1= 0x7ffdcdb168c1 : w 77
S1+18= 0x7ffdcdb168c2 :   20 ........S2+ 2= 0x7ffdcdb168c2 :   20
S1+19= 0x7ffdcdb168c3 : Y 59 ........S2+ 3= 0x7ffdcdb168c3 : Y 59
S1+20= 0x7ffdcdb168c4 : e 65 ........S2+ 4= 0x7ffdcdb168c4 : e 65
S1+21= 0x7ffdcdb168c5 : a 61 ........S2+ 5= 0x7ffdcdb168c5 : a 61
S1+22= 0x7ffdcdb168c6 : r 72 ........S2+ 6= 0x7ffdcdb168c6 : r 72
S1+23= 0x7ffdcdb168c7 :   20 ........S2+ 7= 0x7ffdcdb168c7 :   20
S1+24= 0x7ffdcdb168c8 :   0  ........S2+ 8= 0x7ffdcdb168c8 :   0
-----------------------------------
```

prstring() - using while(1)

- `(*p)` becomes false only when `p` points to the null terminating character.
- in that case, `prstring()` returns to its caller.

prstring() - using while(*p)

- additional printf statements are necessary to print the null terminating character.

```
void prstring(char *S, char *N) {
  int   i;
  char *p;

  puts("-----------------------------------");
  p = S; i = 0;
  while (*p) {
```

```
    printf("%s+%2d= %p : ", N, i, S+i);
    printf("%1c %2x\n", *(S+i), *(S+i));
    p++; i++;
  }
  printf("%s+%2d= %p : ", N, i, S+i);
  printf("%1c %2x\n", *(S+i), *(S+i));
  return;
}
```

prstring() - using do ... while(*p)

- additional printf statements are necessary to print the null terminating character.

```
void prstring(char *S, char *N) {
  int   i;
  char *p;

  puts("-----------------------------------");
  p = S; i = 0;
  do {
    printf("%s+%2d= %p : ", N, i, S+i);
    printf("%1c %2x\n", *(S+i), *(S+i));
    p++; i++;
  } while (*p);
  printf("%s+%2d= %p : ", N, i, S+i);
  printf("%1c %2x\n", *(S+i), *(S+i));
}
```