Chapter 3

Arrays and Strings

Advanced Constructs

Along with the basic data types, C also supports some advanced data constructs.

`array` – ***A sequence of same-type values.***

```
int i[10];
```

`string` – ***An `array` of characters (text).***

```
char c[10];
```

`pointer` – ***Stores an address of some type.***

```
char *ptrc;
```

`structure` – ***A collection of mixed types.***

```
struct s
{ int i[10];
  char *ptrc;
}
```

Memory Map

```
int a[2];    // 8B
float b[3];  // 12B
double c[4];// 32B
char d[5];   // 5B
```

Address	Label	Value
400	a[0]	
404	a[1]	
408	b[0]	
412	b[1]	
416	b[2]	
420	c[0]	
428	c[1]	
436	c[2]	
444	c[3]	
452	d[0]	
453	d[1]	
454	d[2]	
455	d[3]	
456	d[4]	

```
a[0] = 5;

b[1] = 4.0;

c[2] = 14.7;

d[4] = 'a';

b[4] = 15.9
printf("%f\n", b[4]);
```

Works with no errors or warnings!

Wrong size? `b[4]` 4B, but `c[0]` 8B

Array Program

arrays.c

```
int main(void)
{   char n;
    char *ptrc, *min, *max;
…
    // store i in type[i]
    for (n=0; n<10; n++)
    { c[n] = n;
      si[n] = n;
      i[n] = n;
      f[n] = n;
    }
    PrintMemory(min, max);

    // store outside of array
    for (n=0; n<100; n++)
    { c[n] = n;
    }
    PrintMemory(min, max);

    // store by bytes
    for (n =0, ptrc = min; ptrc <=max; ptrc++)
    {   *ptrc = n++;
    }
    PrintMemory(min, max);
}
```

C does not check array boundaries

* Means: at the address given by

& Means: the address of

C Arrays

- We are “going off the end” of the arrays
- Access outside the array clobbers the variables
- Out-of-bound access compiles, but at run time:
  - Segmentation fault
  - Bus error
- Why does the C compiler allow this?
  - Decide on size of array at run time
  - Dynamically change array size

Multi-Dimensional Arrays

A two-dimensional array is really just an array of equal-size one-dimensional arrays.

```
char c[5][4];

// 5 rows of 4 columns
       [0] [1] [2] [3]
c[0]   [A] [B] [C] [D]
c[1]   [E] [F] [G] [H]
c[2]   [I] [J] [K] [L]
c[3]   [M] [N] [O] [P]
c[4]   [Q] [R] [S] [T]
```

mdarrays.c

Builds this matrix

Multi-Dimensional Array Program

mdarrays.c

```
int main(void)
{
  char c[5][4], n, m, *ptrc;

  printf("Address of c = %d\n", c);
  for (n=0; n<5; n++) printf("Addr. of c[%d] = %d\n",n,  &c[n]);
  for (n=0; n<4; n++) printf("Addr. of c[0][%d] = %d\n",n,  &c[0][n]);
  printf("\n");

  for (n=0, ptrc = &c[0][0]; n<20; ptrc++, n++)
  {  *ptrc = 'A' + n;
  }

  for (n=0; n<5; n++)
  { for (m=0; m<4; m++)
    { printf("c[%d][%d] = %c\n", n, m,  c[n][m]);
    }
  }
  printf("\n");
  for (n=0; n<4; n++)
  { for (m=0; m<5; m++)
    { printf("c[%d][%d] = %c\n", m, n,  c[m][n]);
    }
  }
}
```

Memory Map for `mdarrays.c`

```
char c[5][4], n, m, *ptrc;
```

Address	Label	Value
784	c[0][0]	A
785	c[0][1]	B
786	c[0][2]	C
787	c[0][3]	D
788	c[1][0]	E
789	c[1][1]	F
790	c[1][2]	G
791	c[1][3]	H
792	c[2][0]	I
793	c[2][1]	J
794	c[2][2]	K
795	c[2][3]	L
796	c[3][0]	M
797	c[3][1]	N
798	c[3][2]	O
799	c[3][3]	P

Address	Label	Value
800	c[4][0]	Q
801	c[4][1]	R
802	c[4][2]	S
803	c[4][3]	T
804	n	0 to 20
805	m	
806-809	*ptrc	784 to 804

Address of c = 784

Address of
c[0] = 784
c[1] = 788
c[2] = 792
c[3] = 796
c[4] = 800

Address of
c[0][0] = 784
c[0][1] = 785
c[0][2] = 786
c[0][3] = 787

Multi-Dimensional Array Program

mdarrays.c

```
int main(void)
{
  char c[5][4], n, m, *ptrc;

  printf("Address of c = %d\n", c);
  for (n=0; n<5; n++) printf("Addr. c
  for (n=0; n<4; n++) printf("Addr. c
  printf("\n");

  for (n=0, ptrc = &c[0][0]; n<20; pt
  {  *ptrc = 'A' + n;
  }

  for (n=0; n<5; n++)
  { for (m=0; m<4; m++)
    { printf("c[%d][%d] = %c\n", n, m
    }
  }
  printf("\n");
  for (n=0; n<4; n++)
  { for (m=0; m<5; m++)
    { printf("c[%d][%d] = %c\n", m, n
    }
  }
}
```

```
c[0][0] = A     c[0][0] = A
c[0][1] = B     c[1][0] = E
c[0][2] = C     c[2][0] = I
c[0][3] = D     c[3][0] = M
c[1][0] = E     c[4][0] = Q
c[1][1] = F     c[0][1] = B
c[1][2] = G     c[1][1] = F
c[1][3] = H     c[2][1] = J
c[2][0] = I     c[3][1] = N
c[2][1] = J     c[4][1] = R
c[2][2] = K     c[0][2] = C
c[2][3] = L     c[1][2] = G
c[3][0] = M     c[2][2] = K
c[3][1] = N     c[3][2] = O
c[3][2] = O     c[4][2] = S
c[3][3] = P     c[0][3] = D
c[4][0] = Q     c[1][3] = H
c[4][1] = R     c[2][3] = L
c[4][2] = S     c[3][3] = P
c[4][3] = T     c[4][3] = T
```

Multi-Dimensional Array Program

mdarrays2.c

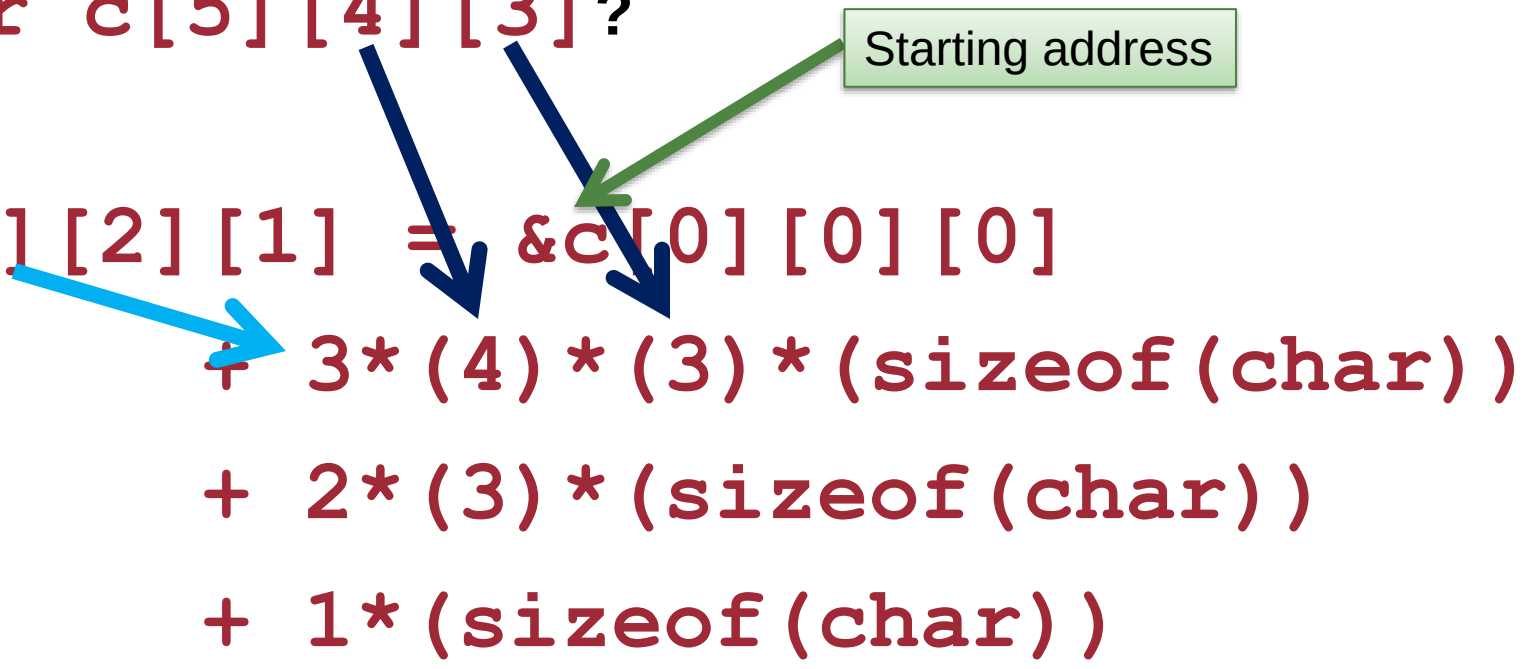
```
Address of c[0][0][0] = 4210784
Address of c[0][0][1] = 4210785
Address of c[0][0][2] = 4210786
Address of c[0][1][0] = 4210787
Address of c[0][1][1] = 4210788
Address of c[0][1][2] = 4210789
Address of c[0][2][0] = 4210790
Address of c[0][2][1] = 4210791
Address of c[0][2][2] = 4210792
Address of c[0][3][0] = 4210793
Address of c[0][3][1] = 4210794
Address of c[0][3][2] = 4210795
Address of c[1][0][0] = 4210796
Address of c[1][0][1] = 4210797
…
Address of c[2][0][0] = 4210808
…
Address of c[3][0][0] = 4210820
…
Address of c[4][0][0] = 4210832
```

```
Address of c = 784
Address of c[1] = 796
Address of c[2][1] = 811
Address of c[3][2][1] = 827
```

Multi-Dimensional Array Program

How does the computer translate the construct `c[3][2][1]`, from the declaration

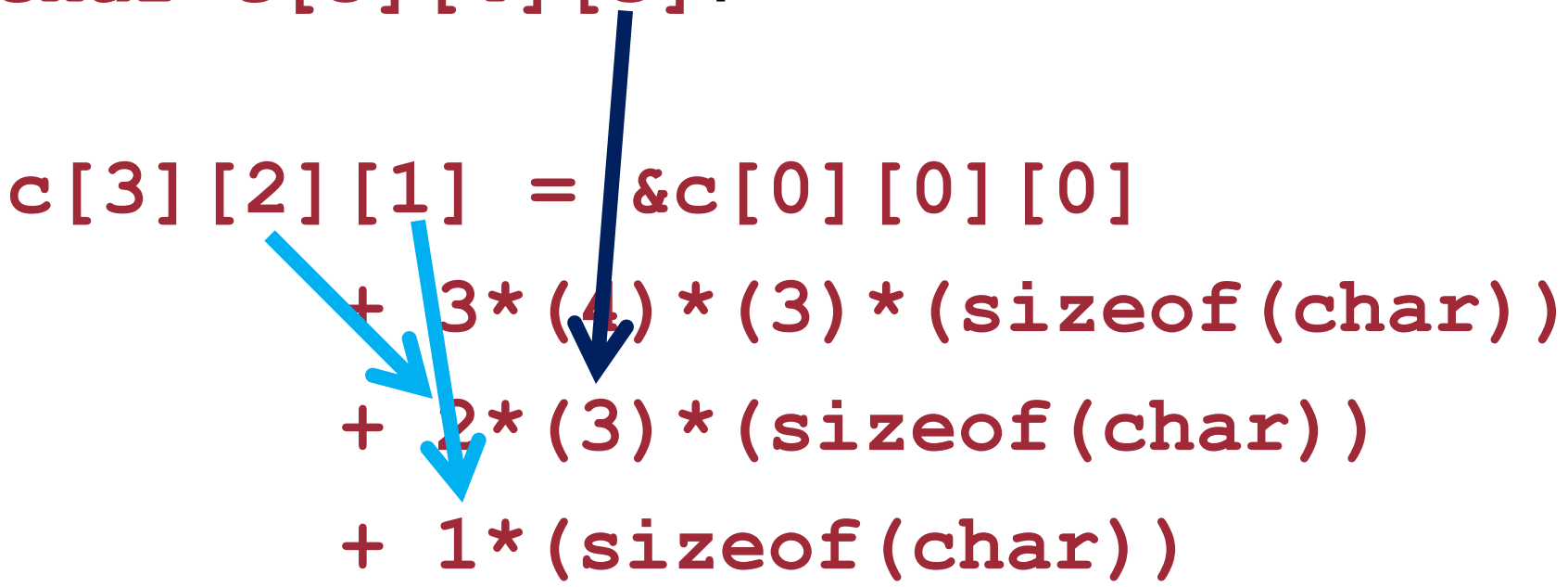
```
char c[5][4][3]?
```



Multi-Dimensional Array Program

How does the compiler translate the construct `c[3][2][1]`, from the declaration

```
char c[5][4][3]?
```



Memory Map

```
int main(void) {
  char c[4][3], n, m, p, q;
  p = '#'; q = '$';
  for (n=0; n<4; n++)
    for (m=0; m<3; m++)
      c[n][m] = 65 + n*3 + m;
}
```

Address	Label	Value
400	c[0][0]	65
401	c[0][1]	66
402	c[0][2]	67
403	c[1][0]	68
404	c[1][1]	69
405	c[1][2]	70
406	c[2][0]	71
407	c[2][1]	72
408	c[2][2]	73
409	c[3][0]	74
410	c[3][1]	75
411	c[3][2]	76

Address	Label	Value
412	n	0 to 4
413	m	0 to 3
414	p	#
415	q	$

What is `c[0][5]`?

What is `c[4][3]`?

What is `c[2]`?

Multidimensional Arrays as Pointers

```
int main(void) {
  char c[4][3], n, m, p, q;
  m = '@'; p = '#'; q = '$';
  for (n=0; n<12; n++) {
    *(*c + n) = 65 + n;
  }
}
```

mdarrays3.c

An address because multidimensional

Bugs:

```
c[n] = 65 + n  // compiler error
*c[n] = 65 + n  // writes first entry of each row
```

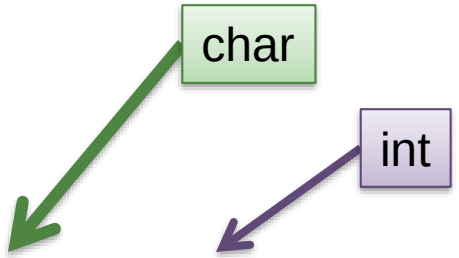
These are equivalent:

```
*(*c + n)        = 65 + n;
*(c[0] + n)      = 65 + n;
*(&c[0][0] + n) = 65 + n;
```

Strings

Strings are simply one-dimensional arrays of characters.

C uses a sentinel to mark end of a sequence. That is, the `'\0'` character determines the end of a string.




```
char a[10];

for (i=0; i<10; i++) a[i] = 'A' + i;

// addr [0 ][1 ][2 ][3 ][4 ][5 ][6 ][7 ][8 ][9 ]
// data [65][66][67][68][69][70][71][72][73][74]
//      [A ][B ][C ][D ][E ][F ][G ][H ][I ][J ]
```

Anything wrong with the above code?

A Warning about Strings

Strings in C are delimited by a `'\0'` character placed at the end of a string. (Some other languages store strings differently such as using the first byte of the string to tell how long the string is.)

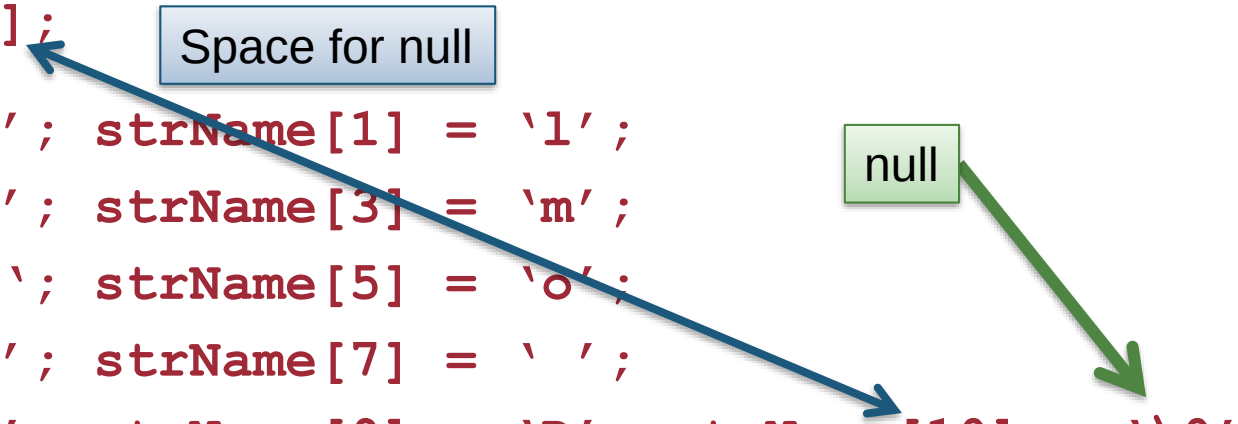
Therefore, it is important to always leave space for the terminator character when allocating space for a string.

It is also important to make sure you append the `'\0'` character to a string when writing your own string operations.

The '\0' Character

For example, if you are saving 10-character names in strings, dimension the string as a 11-charcter array.

```
char strName[11];

strName[0] = 'C'; strName[1] = 'l';
strName[2] = 'e'; strName[3] = 'm';
strName[4] = 's'; strName[5] = 'o';
strName[6] = 'n'; strName[7] = ' ';
strName[8] = 'I'; strName[9] = 'D'; strName[10] = '\0'
```



We use null in three different ways

- `'\0'` refers to ASCII characters
- `NULL` refers to pointers
- `0` refers to integers

Strings Program

strings.c

```
int main(void)
{   char s[20];
    char i;

    s[0] = 'C';   s[1] = 'l';   s[2] = 'e';   s[3] = 'm';
    s[4] = 's';   s[5] = 'o';   s[6] = 'n';   s[7] = ' ';
    s[8] = 'T';   s[9] = 'i';   s[10] = 'g';   s[11] = 'e';
    s[12] = 'r';   s[13] = 's';   s[14] = '!';

    for (i=0; i<20; i++) printf("%c", s[i]);
    printf("\n"); getchar();

    s[15] = '\0';

    for (i=0; i<20; i++) printf("%c", s[i]);
    printf("\n"); getchar();

    i = 0;
    while (s[i] != '\0') { printf("%c", s[i++]);
    printf("\n"); getchar();

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

    printf("%s\n", &s[8]); getchar

    printf("%s\n", s[0]); getchar();
}
```

Correct null

Stop when get to null

s is a pointer

Not a pointer

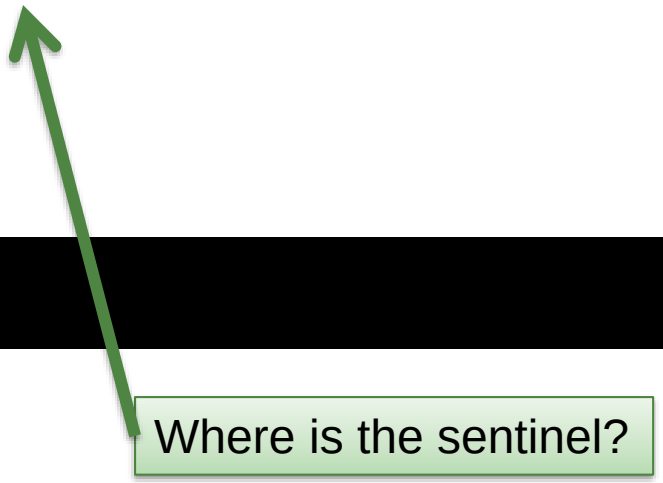
```
Clemson Tigers! $☺
Clemson Tigers! $☺
Clemson Tigers!
Clemson Tigers!
Tigers!
<Seg Fault>
```

Arrays of Strings

Multi-dimensional arrays of characters can be used to store a list of text.

```
char str[10][10] =
{ "APPLE", "BALL", "CAT", "DOG", "EARTH",
  "FLOWER", "GIRL", "HAT", "ICE", "JET"
};
printf("%s\n", str[7]);
```

```
HAT
```



Array of Strings Program

```
char str[10][10] =
{ "APPLE", "BALL", "CAT", "DOG"
  "FLOWER", "GIRL", "HAT", "ICE
};

int main(void)
{  int i;

   for (i=0; i<10*10; i++) prin
(&str[0][0]+i), *(&str[0][0]+i)

   getchar();

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

   printf("%s\n", str[0]);

   printf("%s\n", &str[0][0

   printf("%s\n", str[3]);

   printf("%s\n", &str[4][0]);

   printf("%s\n", &str[4][2]);
   printf("%s\n", &str[1][20]);
}
```

stringa

```
4202496 - A
4202497 - P
4202498 - P
4202499 - L
4202500 - E
4202501 -
4202502 -
4202503 -
4202504 -
```

```
…
APPLE
APPLE
APPLE
DOG
EARTH
RTH
DOG
```

```
4202514 -
4202515 -
4202516 - C
4202517 - A
4202518 - T
...
```

Array of String

stringarray2.c

```
char strACCTeam[12][21] =
{ "Boston College", "Clemson", "Duke", "Flori
 "Maryland", "Miami", "North Carolina", "Nort
 "Virginia Tech", "Wake Forest"
};

char *strACCTeamCompact[] =
{ "Boston College", "Clemson", "Duke", "Flori
 "Maryland", "Miami", "North Carolina", "Nort
 "Virginia Tech", "Wake Forest"
};

int main(void)
{  int i;

   for (i=0; i<ACC_TEAMS; i++) printf("%s\n",
   printf("\n");

   for (i=0; i<ACC_TEAMS; i++) printf("%s\n",
   printf("\n");

   printf("The greatest team is %s!\n", strAC
   getchar();

   for (i=0; i<12*21; i++)
     printf("%c   %c\n", *(&strACCTeam[0][0]+
}
```

String Functions

The C library `string.h` contains many useful string and memory manipulation functions.

```
// Copy a string to another
char *strcpy(char *dest, const char *src);
char *strncpy(char *dest, const char *src, size_t maxlen);

// Calculate the length of string
size_t strlen(const char *s);
```

stringlen.c

```
// Compare two strings
int strcmp(const char *s1, const char *s2);
int strncmp(const char *s1, const char *s2, size_t maxlen);
```

stringcmp.c

Memory Map

main

my_strcmp(str1,str2);

Address	Label	Value
400-406	*Tigers	Tigers
407-416	*Game	Gamecocks
417-426	*Squ	Squirrels
427-506	str1	abc\n
507-586	str2	def\n
587-590	ans	?

Address	Label	Value
900-903	s	
904-907	t	
908-911	i	0
912-915	answer	0

What is `s[0]`?

```
int my_strcmp(char s[], char t[]) {
    int i = 0, answer = 0;
    while (answer == 0) {
        if (s[i] < t[i]) answer = -1;
        else if (s[i] > t[i]) answer = 1;
        if (s[i] == '\0' || t[i] == '\0') break;
        i++;
    }
    return answer;
}
```

stringcmp.c

String Functions

The C library `string.h` contains many useful string and memory manipulation functions.

```
// Copy a string to another
char *strcpy(char *dest, const char *src);
char *strncpy(char *dest, const char *src, size_t maxlen);
```

Think-Pair-Share

More String Functions

```
// Add to the end of (Concatenate) a string
char *strcat(char *dest, const char *src);
char *strncat(char *dest, const char *src, size_t maxlen);
```

stringcpycat.c

```
// Find the first instance of a char within a string
char *strchr(const char *s, int c);
```

stringchr.c

```
// Scans a string for the occurrence of a substring
char *strstr(const char *s1, const char *s2);
```

stringstr.c

```
// Returns the length of the initial portion of s1
// which is also present in s2.
```

stringspn.c

```
size_t strspn(const char *s1, const char *s2);
// Convert a string to upper or lower case
char *strupr(char *s);
char *strlwr(char *s);
```

stringuprlwr.c

Character Functions

```
// Not String functions, but help to translate
// characters to upper and lower case.
int tolower(int ch);
int toupper(int ch);
```

ToUpper.c

```
if ('A' <= c && c <= 'Z' || 'a' <= c && c <= 'z')
```

```
// Checks to see if character is a letter.
int isalpha(int ch);
```

IsAlpha.c

```
#include <ctype.h>
isalnum() // checks for an alphanumeric character, i.e. (isalpha(c) || isdigit(c))
isblank() // checks for a blank character; that is, a space or a tab.
iscntrl() // checks for a control character.
isdigit() // checks for a digit (0 through 9).
isgraph() // checks for any printable character except space.
islower() // checks for a lower-case character.
isprint() // checks for any printable character including space.
ispunct() // checks for printable character which is not a space or alphanumeric
isspace() // checks for white-space characters (' ', '\f', '\n', '\r', '\t', '\v')
isupper() // checks for an uppercase letter.
isxdigit() // checks for a hexadecimal digits, i.e. one of
```


http://linux.die.net/man/3/isalpha

String Conversions

```
int atoi(char s[])
{
    int i, n, sign;
    for (i = 0; isspace(s[i]); i++)   // skip white space
        ;
    sign = (s[i] == '-') ? -1 : 1;
    if (s[i] == '+' || s[i] ==  '-')  // skip sign
        i++;
    for (n = 0; isdigit(s[i]); i++)
        n = 10 * n + (s[i] - '0');
    return sign * n;
}
```

Why use `s[]`?

`'0' <= s[i] && s[i] <= '9'`

Type conversion

stringatoi.c

```
// Convert a string to an integer or float
int atoi(const char *str);
long atol(const char *str);
long long atoll(const char *str);
double atof(const char *str);
```

Program Structure
- Skip white space, if any
- Get sign, if any
- Get integer part and convert it

Memory Map

stringatoi.c

main | **atoi(str1);**

Address	Label	Value
400	str1	' '
401		'-'
402		'2'
403		'3'
404		'a'
405		'\0'
…		

Address	Label	Value
900-903	s	
904-907	i	
908-911	n	
912-915	sign	

sprintf()

The C library function `sprintf()` is identical to `printf()` except that it "prints" the string to a memory location instead of the console. It allows the programmer to easily store formatted text into memory.

```
int sprintf(char *str, const char *format,...);
```

For this prototype, `sprintf()` copies the string `format` to the memory location pointed to by the address `str`.

The format string `format` can optionally contain embedded *format tags* that are substituted by the values specified in subsequent argument(s) and formatted as requested.

sprintf() Example

sprintf.c

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

int main(void)
{ char str[120];

  sprintf(str, "The value of Pi is %f,
           and the inverse of Pi is %f.\n", M_PI, M_1_PI);
   puts(str);
}
```

```
The value of Pi is 3.141593, and the inverse of Pi is 0.318310.
```

sprintf() Example 2

sprintf2.c

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

int main(void)
{ char strInput[256], strFormat[256];

  puts("Enter the type you wish to use: i d u X f lf - \n");
  gets(strInput);

  sprintf(strFormat, "The value = %%%s", strInput);

  printf("The Format is \"%s\"\n", strFormat);
  printf(strFormat, -15);
}
```

sprintf() Example 3

sprintf3.c

```
FILE *fout;

int main(void)
{ int month, day, year;
  char strFile[256];

  printf("Enter Month Day Year: ");
  scanf("%d %d %d",&month, &day, &year);

  sprintf(strFile, "TestData-%d-%d-%d.txt",month,day,year);
  if ((fout = fopen(strFile, "wt")) == NULL)
  { printf("Cannot open output file."); exit(1);
  }

  fprintf(fout,"Test Data for %d/%d/%d\n", month,day,year);

  fclose(fout);
}
```

String Manipulation

The string library only contains routinely needed functions. Programmers still must be able to write their own string manipulation functions to perform specialized tasks.

stringmanip.c

```
i = j = 0;

while (str1[i] != '\0') // Do 'til end of string
{ if (strncmp(&str1[i], str3, strlen(str3)) == 0)
  { strcpy(&str2[j], str3);
    j += strlen(str3);
    i += strlen(str3);
    strcpy(&str2[j], str4);
    j += strlen(str4);
  }
  else str2[j++] = str1[i++];
 }
str2[j] = '\0';
```

Command Line Arguments

C provides a method for passing arguments with the command name of the process.

```
int main(int argc, char *argv[])
```

`argc` is the number of arguments passed.

(Note: the command itself counts as one.)

`argv[]` is a string array of each argument entered.

Command Line Arguments Example

```
#include <stdio.h>

int main(int argc, char *argv[])
{ int i;

  for (i=0; i<argc; i++)
  { printf("Argument %d is: %s\n", i, argv[i]);
  }
}
```

Command1.c

What is output for: ./Command1 my “three strings”

Command Line Arguments Example 2

Command2.c

```
/* … */
switch (argc)
{ case 1:
  case 2:
    printf("Invalid arguments entered.\n");
    printf("The command has the form:  command cheer n\n
             where cheer is a string and n is the number of times to repeat.");
    return (0);

  case 3:
    n = atoi(argv[2]);
    for (i=0; i<4; i++)
    { if (strcmp(argv[1], Cheer[i][0]) == 0)
      { for (j=0; j<n; j++) { printf("%s\n", Cheer[i][1]);   }
        return (0);
      }
    }
    printf("Don't Know that Cheer...\n"); break;
}
```

Command Line Arguments Example 3

Command3.c

```
const char strUsage[] = "Usage: [add n m] where n and m are binary numbers.";

int main(int argc, char *argv[])
{  int i, n, m;

   switch (argc)
   { case 1:
       printf("Error 1 - No operands.  %s\n", strUsage); exit(0);

     case 2:
       if (strcmp(argv[1], "-help") == 0)
       { puts(strUsage);
       } else printf("Error 2 - No second operand. %s\n", strUsage);
       exit(0);

     case 3:
       i = 0; n = 0;
       while (argv[1][i] != '\0')
       { if (argv[1][i] == '1')
         { n <<= 1;  n |= 1;
         }
         else if (argv[1][i] == '0')
         { n <<= 1;
         }
         else
         { printf("Error 3 - First operand not a binary number. %s\n", strUsage);
           exit(0);
         }
         i++;
       }
```

Pop Quiz

- Write a program that accepts up to six arguments at the command line prompt.
  - The program should print the first character of any arguments at position 0, 2, or 4, and the second character of any arguments at positions 1, 3, or 5.
  - The characters printed should be separated by spaces
  - The program should inform the user of the correct program usage if fewer than two or more than six arguments are provided.
  - Assume each argument contains at least two characters.

Pos:	0	1	2	3	4	5
myprog	ab	cd	ef	gh	ij	

Output:

```
m b c f g j
```

Command Line Arguments Example 3

How are constant strings stored in a program?

`HidingString.exe`

Note on Using [string] Functions

Is this code efficient?

strlen2.c

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

int main(void)
{ int k, n;
  char a[]="The quick brown fox jumps over the lazy yellow dog";

  /* …… */

  for (k=0; k<strlen(a); k++)
  { /* …… */
  }
  fn = strlen(a);
  for (k=0; k<fn; k++)
  {
```

```
Time0 = 1329938461 : 130
Time1 = 1329938462 : 271
Difference = 1141

Time0 = 1329938460 : 966
Time1 = 1329938461 : 130
Difference = 164
```

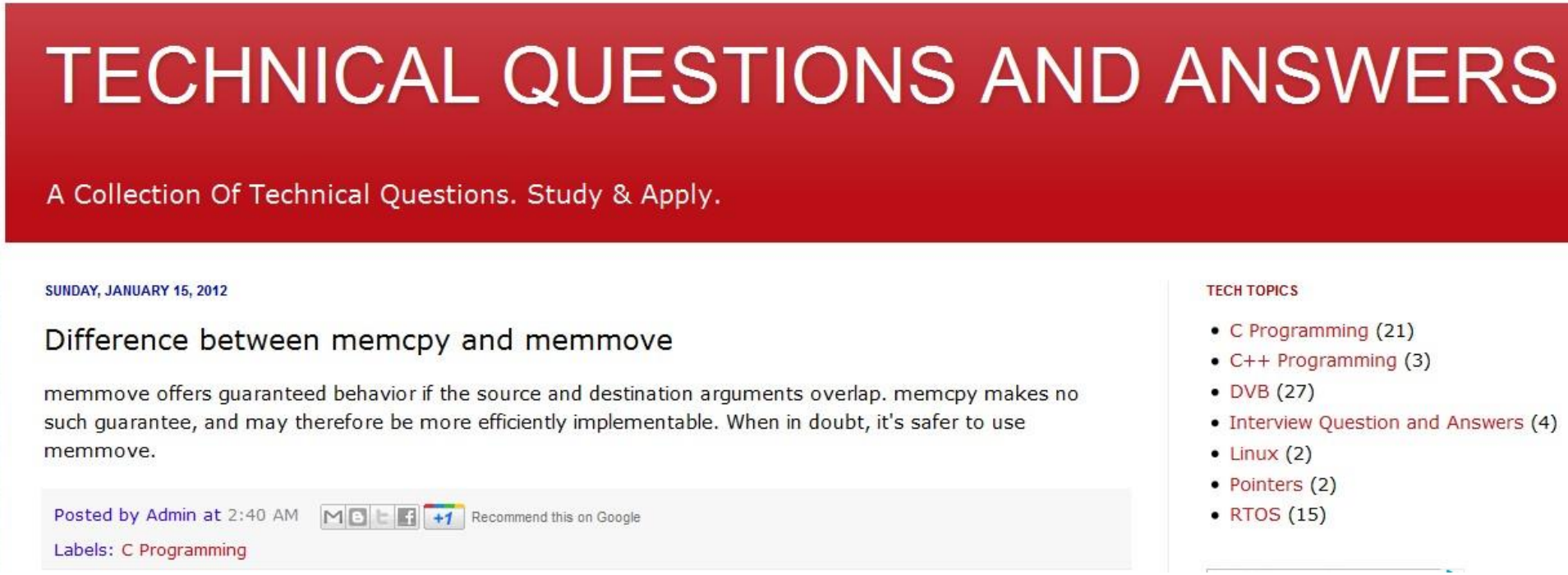
Other Memory Functions

```
// Set a block of memory to a certain character value
void *memset(void *ptr, int value, size_t num);

// Copies a block of memory from one place to another
void *memcpy(void *dest, const void *src, size_t num);

// Copies a block of memory from one place to another
void *memmove(void *dest, const void *src, size_t num);
```

MemFuncs.c

memmove() vs. memcpy()

```
// Copies a block of memory from one place to another
void *memcpy(void *dest, const void *src, size_t num);

// Copies a block of memory from one place to another
void *memmove(void *dest, const void *src, size_t num);
```



MemFuncs2.c