CIS 330:

Lecture 4:
Build Systems, Tar, Character Strings
(Finish Lecture 3)
More array relationships

```c
int A[10];
int *B;

B=&(A[0])  \rightarrow  B = A
B=&(A[5])  \rightarrow  B = A+5
```
Pointers to pointers

- Remember: pointer points to a location in memory
  - We’ve been considering cases where locations in memory are arrays of integers
  - But locations in memory could be pointer themselves
Simple pointers to pointers example

```c
#include <stdlib.h>
int main()
{
    int **X = malloc(sizeof(int *)*4);
    X[0] = malloc(sizeof(int)*6);
    X[1] = malloc(sizeof(int)*4);
    X[2] = malloc(sizeof(int)*8);
    X[3] = malloc(sizeof(int)*10);
}
```

```bash
C02LN00GFD58:330 hank$ gcc ptrptr.c
C02LN00GFD58:330 hank$ ./a.out
```
What’s the difference between these two programs?

Answer: X is on the heap on the left, and on the stack on the right. But they are both pointers-to-pointers.
What's the difference between these two programs?

C02LN00GFD58:330 hank$ cat ptrptr.c
#include <stdlib.h>
int main()
{
  int **X = malloc(sizeof(int *)*4);
  X[0] = malloc(sizeof(int)*6);
  X[1] = malloc(sizeof(int)*4);
  X[2] = malloc(sizeof(int)*8);
  X[3] = malloc(sizeof(int)*10);
}
C02LN00GFD58:330 hank$ gcc ptrptr.c
C02LN00GFD58:330 hank$ ./a.out

C02LN00GFD58:330 hank$ cat ptrptr3.c
#include <stdlib.h>
int main()
{
  int **X = malloc(sizeof(int *)*4);
  int num = 6+4+8+10;
  int *allMem = malloc(sizeof(int)*num);
  X[0] = allMem;
  X[1] = X[0]+6;
}
C02LN00GFD58:330 hank$ gcc ptrptr3.c
C02LN00GFD58:330 hank$ ./a.out

Answer: program on left makes one allocation for each pointer, program on right makes one allocation for whole program & each pointer points at locations within that allocation.
Call by value / call by reference

• Refers to how parameters are passed to a function.
  – Call by value: send the value of the variable as a function parameter
    • Side effects in that function don’t affect the variable in the calling function
  – Call by reference: send a reference (pointer) as a function parameter
    • Side effects in that function affect the variable in the calling function
Call by Value

```c
#include <stdio.h>

void foo(int x)
{
    x = x+1;
}

int main()
{
    int x = 2;
    foo(x);
    printf("X is %d\n", x);
}
```

```
C02LN00GFD58:330 hank$ cat cbv.c
C02LN00GFD58:330 hank$ gcc cbv.c
C02LN00GFD58:330 hank$ ./a.out
X is 2
```
Call by value

```c
int doubler(int A)
{
    int stack_varA;
    stack_varA = 2*A;
    return stack_varA;
}

int main()
{
    int stack_varC;
    int stack_varD = 3;
    stack_varC = doubler(stack_varD);
}
```

Return copies into location specified by calling function

Code

Data

Stack

- stack_varC
- stack_varD
- A (= 3)

<info for how to get back to main>

Free

Heap

<int location for RV>

stack_varA
Call by reference

```c
#include <stdio.h>

void foo(int *x) {
    *x = *x + 1;
}

int main() {
    int x = 2;
    foo(&x);
    printf("X is \%d\n", x);
}
```

```bash
hank$ cat cbr.c
hank$ gcc cbr.c
hank$ ./a.out
X is 3
```
Call by reference

```c
int doubler(int*A)
{
    int stack_varA;
    stack_varA = 2*A;
    return stack_varA;
}

int main()
{
    int stack_varC;
    int stack_varD = 3; &
    stack_varC = doubler(stack_varD);
}
```

Return copies into location specified by calling function

Code
- Data
- Stack
  - stack_varC
  - stack_varD
  - <info for how to get back to main>
  - A (= address)
  - <Location for RV>
  - stack_varA

Free

Heap
(grade 2A)
Outline

• Review
• Project 1B Overview
• Build
• Project 1C Overview
• Tar
• Character Strings
Useful vi commands

• yy: yank the current line and put it in a buffer
  – 2yy: yank the current line and the line below it
• p: paste the contents of the buffer
• Navigation
  – “:100” go to line 100 in the file
  – ‘/’: search forwards, ‘?’: search backwards
• Arrows can be used to navigate the cursor position (while in command mode)
  – So do h, j, k, and l

We will discuss more tips for “vi” throughout the quarter. They will mostly be student-driven (Q&A time each class)
Permissions: System Calls

• System calls: a request from a program to the OS to do something on its behalf
  – … including accessing files and directories

• System calls:
  – Typically exposed through functions in C library
  – Unix utilities (cd, ls, touch) are programs that call these functions

Permissions in Unix are enforced via system calls.
Executable files

• An executable file: a file that you can invoke from the command line
  – Scripts
  – Binary programs

• The concept of whether a file is executable is linked with file permissions
Translating permissions to binary

<table>
<thead>
<tr>
<th>#</th>
<th>Permission</th>
<th>rwx</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>full</td>
<td>111</td>
</tr>
<tr>
<td>6</td>
<td>read and write</td>
<td>110</td>
</tr>
<tr>
<td>5</td>
<td>read and execute</td>
<td>101</td>
</tr>
<tr>
<td>4</td>
<td>read only</td>
<td>100</td>
</tr>
<tr>
<td>3</td>
<td>write and execute</td>
<td>011</td>
</tr>
<tr>
<td>2</td>
<td>write only</td>
<td>010</td>
</tr>
<tr>
<td>1</td>
<td>execute only</td>
<td>001</td>
</tr>
<tr>
<td>0</td>
<td>none</td>
<td>000</td>
</tr>
</tbody>
</table>

Which of these modes make sense? Which don’t?

We can have separate values (0-7) for user, group, and other.
Unix command: chmod

• chmod: change file mode

• chmod 750 <filename>
  – User gets 7 (rwx)
  – Group gets 5 (rx)
  – Other gets 0 (no access)

Lots of options to chmod (usage shown here is most common)
ls -l

- Long listing of files

- Permissions
- Links (*)
- Owner
- Group
- File size
- Date of last change
- Filename

How to interpret this?
Permissions and Directories

• You can only enter a directory if you have “execute” permissions to the directory.

• Quiz: a directory has permissions “400”. What can you do with this directory?

Answer: it depends on what permissions a system call requires.
Directories with read, but no execute

Last login: Thu Apr 3 08:14:33 on ttys007
C02LN00GFD58:~ hank$ mkdir CIS330
C02LN00GFD58:~ hank$ touch CIS330/a
C02LN00GFD58:~ hank$ chmod 400 CIS330
C02LN00GFD58:~ hank$ ls CIS330
a
C02LN00GFD58:~ hank$ cd CIS330
-bash: cd: CIS330: Permission denied
C02LN00GFD58:~ hank$ cat CIS330/a
cat: CIS330/a: Permission denied
Outline

• Review
• Project 1B Overview
• Build
• Project 1C Overview
• Tar
• Character Strings
Unix scripts

• Scripts
  – Use an editor (vi/emacs/other) to create a file that contains a bunch of Unix commands
  – Give the file execute permissions
  – Run it like you would any program!!
Unix scripts

• Arguments
  – Assume you have a script named “myscript”
  – If you invoke it as “myscript foo bar”
  – Then
    • $# == 2
    • $1 == foo
    • $2 == bar
Project 1B

• Summary: write a script that will create a specific directory structure, with files in the directories, and specific permissions.
Project 1B

The directory structure should be:

- Root dir
  - Dir1 Permissions: 770
    - File1 Permissions=400
      - Dir3 Permissions: 000
        - Dir4 Permissions: 750
          - File4 Permissions=666
          - File3 Permissions=200
  - Dir2 Permissions: 775
    - File2 Permissions=640

Key
- Files:
  - Name of file Permissions
- Directories:
  - Name of directory Permissions:
Outline

• Review
• Project 1B Overview
• Build
• Project 1C Overview
• Tar
• Character Strings
Build: The Actors

- File types
  - Source code
  - Object code
  - Executable code

- Programs
  - Compiler
  - Linker
Analogy

Source Code

Object Code

Executable Code
Compilers, Object Code, and Linkers

• Compilers transform source code to object code
  – Confusing: most compilers also secretly have access to linkers and apply the linker for you.

• **Object code**: statements in machine code
  – not executable
  – intended to be part of a program

• Linker: turns object code into executable programs
GNU Compilers

• GNU compilers: open source
  – gcc: GNU compiler for C
  – g++: GNU compiler for C++

C++ is superset of C. With very few exceptions, every C program should compile with a C++ compiler.
C++ comments

• “//” : everything following on this line is a comment and should be ignored

• Examples:

  // we set pi below
  float pi = 3.14159; // approximation of pi

Can you think of a valid C syntax that will not compile in C++?

float radians=degrees/*approx. of pi*//3.14159;
A comment on case (i.e., uppercase vs lowercase)

• Case is important in Unix
  – But Mac is tolerant

• gcc t.c
  → invokes C compiler

• gcc t.C
  → invokes C++ compiler
Our first gcc program

C02LN00GFD58:CIS330 hank$ cat t.c
#include <stdio.h>
int main()
{
    printf("hello world!\n");
}
C02LN00GFD58:CIS330 hank$ gcc t.c
C02LN00GFD58:CIS330 hank$ ./a.out
hello world!
C02LN00GFD58:CIS330 hank$
Our first gcc program: named output

```
C02LN00GFD58:CIS330 hank$ cat t.c
#include <stdio.h>
int main()
{
    printf("hello world!\n");
}
C02LN00GFD58:CIS330 hank$ gcc t.c
C02LN00GFD58:CIS330 hank$ ./a.out
hello world!
C02LN00GFD58:CIS330 hank$ gcc -o helloworld t.c
C02LN00GFD58:CIS330 hank$ ./helloworld
hello world!
C02LN00GFD58:CIS330 hank$ ls -l helloworld
-rwxr-xr-x 1 hank staff 8496 Apr 3 15:15 helloworld
C02LN00GFD58:CIS330 hank$
```

"-o" sets name of output
Output name is different
Output has execute permissions
gcc flags: debug and optimization

• “gcc –g”: debug symbols
  – Debug symbols place information in the object files so that debuggers (gdb) can:
    • set breakpoints
    • provide context information when there is a crash

• “gcc –O2”: optimization
  – Add optimizations ... never fails

• “gcc –O3”: provide more optimizations
  – Add optimizations ... sometimes fails

• “gcc –O3 –g”
  – Debugging symbols slow down execution ... and sometimes compiler won’t do it anyways...
Debug Symbols

• live code

```c
int main()
{
    int sum = 0;
    int i;
    for (i = 0; i < 10; i++)
        sum += i;
    return sum;
}
```

• `gcc -S t.c`  # look at t.s
• `gcc -S -g t.c`  # look at t.s

• (-S flag: compile to assembly instead of object code)
Object Code Symbols

• Symbols associate names with variables and functions in object code.

• Necessary for:
  – debugging
  – large programs
Large code development

Why could this be a good idea?
Multi-file development: example

```
fawcett:330 child$ cat t1.c
int doubler(int x)
{
    return 2*x;
}
fawcett:330 child$ cat t2.c
int main()
{
    return doubler(5);
}
fawcett:330 child$ gcc -c t1.c
fawcett:330 child$ gcc -c t2.c
fawcett:330 child$ gcc -o both t2.o t1.o
fawcett:330 child$ ./both
fawcett:330 child$ echo $?
10
```

*cat is a Unix command that prints the contents of a file*

*$? is a shell construct that has the return value of the last executed program*
Multi-file development: example

fawcett:330 childs$ cat t1.c
int doubler(int x)
{
    return 2*x;
}
fawcett:330 childs$ cat t2.c
int main()
{
    return doubler(5);
}
fawcett:330 childs$ gcc -c t1.c
fawcett:330 childs$ gcc -c t2.c
fawcett:330 childs$ gcc -o both t2.o t1.o
fawcett:330 childs$ ./both
fawcett:330 childs$ echo $?
Multi-file development: example

```c
fawcett:330 childds$ cat t1.c
int doubler(int x)
{
    return 2*x;
}
fawcett:330 childds$ cat t2.c
int main()
{
    return doubler(5);
}
fawcett:330 childds$ gcc -c t1.c
fawcett:330 childds$ gcc -c t2.c
fawcett:330 childds$ gcc -o both t1.o t2.o
fawcett:330 childds$ ./both
fawcett:330 childds$ echo $? 10
```

Linker order matters for some linkers (not Macs). Some linkers need the .o with “main” first and then extract the symbols they need as they go. Other linkers make multiple passes.
Libraries

• Library: collection of “implementations” (functions!) with a well defined interface

• Interface comes through “header” files.

• In C, header files contain functions and variables.
  – Accessed through “#include <file.h>”
Libraries

• Why are libraries a good thing?

• Answers:
  – separation
    • I.e., divide and conquer
      – increases productivity
    • I.e., simplicity
    • I.e., prevents tendrils between modules that shouldn’t exist
  – encapsulation (hides details of the implementation)
    • “A little knowledge is a dangerous thing”...

• Products
  – I can sell you a library and don’t have to give you the source code.
Libraries

• Why are libraries a bad thing?

• Answers:
  – separation
    • I.e., makes connections between modules harder
      – (were the library interfaces chosen correctly?)
  – complexity
    • need to incorporate libraries into code compilation
Includes and Libraries

• gcc support for libraries
  – “-I”: path to headers for library
    • when you say “#include <file.h>, then it looks for file.h in the directories -I points at
  – “-L”: path to library location
  – “-lname”: link in library libname
Library types

• Two types:
  – static and shared
• Static: all information is taken from library and put into final binary at link time.
  – library is never needed again
• Shared: at link time, library is checked for needed information.
  – library is loaded when program runs

More about shared and static later ... for today, assume static
Making a static library

```
C02LN00GFD58:multiplier hank$ cat multiplier.h # here's the header file
int doubler(int);
int tripler(int);
C02LN00GFD58:multiplier hank$ cat doubler.c # here's one of the c files
int doubler(int x) {return 2*x;}
C02LN00GFD58:multiplier hank$ cat tripler.c # here's the other c files
int tripler(int x) {return 3*x;}
C02LN00GFD58:multiplier hank$ gcc -c doubler.c # make an object file
C02LN00GFD58:multiplier hank$ ls doubler.o # we now have a .o
  doubler.o
C02LN00GFD58:multiplier hank$ gcc -c tripler.c
C02LN00GFD58:multiplier hank$ ar r multiplier.a doubler.o tripler.o
C02LN00GFD58:multiplier hank$
```

(should have called this libmultiplier.a)

Note the ‘#’ is the comment character
What's in the file?

```
C02LN00GFD58:multiplier hank$ nm multiplier.a

multiplier.a(doubler.o):
00000000000000038 s EH_frame0
0000000000000000 T _doubler
00000000000000050 S _doubler.eh

multiplier.a(tripler.o):
00000000000000030 s EH_frame0
0000000000000000 T _tripler
00000000000000048 S _tripler.eh
C02LN00GFD58:multiplier hank$
```
Typical library installations

• Convention
  – Header files are placed in “include” directory
  – Library files are placed in “lib” directory

• Many standard libraries are installed in /usr
  – /usr/include
  – /usr/lib

• Compilers automatically look in /usr/include and /usr/lib (and other places)
Installing the library

```
C02LN00GFD58:multiplier hank$ mkdir ~/multiplier
C02LN00GFD58:multiplier hank$ mkdir ~/multiplier/include
C02LN00GFD58:multiplier hank$ cp multiplier.h ~/multiplier/include/
C02LN00GFD58:multiplier hank$ mkdir ~/multiplier/lib
C02LN00GFD58:multiplier hank$ cp doubler.c multiplier.a tripler.c
doubler.o multiplier.h tripler.o
C02LN00GFD58:multiplier hank$ cp multiplier.a ~/multiplier/
C02LN00GFD58:multiplier hank$ mv multiplier.a libmultiplier.a
C02LN00GFD58:multiplier hank$ cp libmultiplier.a ~/multiplier/lib/
C02LN00GFD58:multiplier hank$ 
```

“mv”: unix command for renaming a file
Example: compiling with a library

C02LN00GFD58:CIS330 hank$ cat t.c
#include <multiplier.h>
#include <stdio.h>
int main()
{
    printf("Twice 6 is %d, triple 6 is %d\n", doubler(6), tripler(6));
}
C02LN00GFD58:CIS330 hank$ gcc -o mult_example t.c -I/Users/hank-multiplier/include -L/Users/hank-multiplier/lib -lmultiplier
C02LN00GFD58:CIS330 hank$ ./mult_example
Twice 6 is 12, triple 6 is 18
C02LN00GFD58:CIS330 hank$

• gcc support for libraries
  – “-I”: path to headers for library
  – “-L”: path to library location
  – “-lname”: link in library libname
• There is a Unix command called “make”
• make takes an input file called a “Makefile”
• A Makefile allows you to specify rules
  – “if timestamp of A, B, or C is newer than D, then carry out this action” (to make a new version of D)
• make’s functionality is broader than just compiling things, but it is mostly used for computation

Basic idea: all details for compilation are captured in a configuration file ... you just invoke “make” from a shell
Makefiles

- Reasons Makefiles are great:
  - Difficult to type all the compilation commands at a prompt
  - Typical develop cycle requires frequent compilation
  - When sharing code, an expert developer can encapsulate the details of the compilation, and a new developer doesn’t need to know the details ... just “make”
Makefile syntax

• Makefiles are set up as a series of rules
• Rules have the format:
  target: dependencies
  [tab] system command
Makefile example: multiplier lib

C02LN00GFD58:code hank$
cat Makefile
lib: doubler.o tripler.o
  ar r libmultiplier.a doubler.o tripler.o
  cp libmultiplier.a ~/multiplier/lib
  cp multiplier.h ~/multiplier/include

doubler.o: doubler.c
  gcc -c doubler.c

tripler.o: tripler.c
  gcc -c tripler.c
C02LN00GFD58:code hank$
C02LN00GFD58:code hank$
C02LN00GFD58:code hank$
m make
ar r libmultiplier.a doubler.o tripler.o
cp libmultiplier.a ~/multiplier/lib
cp multiplier.h ~/multiplier/include
C02LN00GFD58:code hank$
C02LN00GFD58:code hank$
touch doubler.c
C02LN00GFD58:code hank$
m make
gcc -c doubler.c
ar r libmultiplier.a doubler.o tripler.o
cp libmultiplier.a ~/multiplier/lib
cp multiplier.h ~/multiplier/include
C02LN00GFD58:code hank$
Fancy makefile example: multiplier lib

```
CC=gcc
CFLAGS=-g
INSTALL_DIR=~/.multiplier

AR=ar
AR_FLAGS=r

SOURCES=doubler.c tripler.c
OBJECTS=\$(SOURCES:.c=.o)

lib: \$(OBJECTS)
  \$(AR) \$(AR_FLAGS) libmultiplier.a \$(OBJECTS)
  cp libmultiplier.a \$(INSTALL_DIR)/lib
  cp multiplier.h \$(INSTALL_DIR)/include

.c.o:
  \$(CC) \$(CFLAGS) -c <
```

C02LN00GFD58:code hank$ touch doubler.c
C02LN00GFD58:code hank$ make
gcc -g -c doubler.c
ar r libmultiplier.a doubler.o tripler.o
cp libmultiplier.a ~/multiplier/lib
cp multiplier.h ~/multiplier/include
C02LN00GFD58:code hank$
Configuration management tools

• Problem:
  – Unix platforms vary
    • Where is libX installed?
    • Is OpenGL supported?

• Idea:
  – Write problem that answers these questions, then adapts build system
    • Example: put “-L/path/to/libX –lX” in the link line
    • Other fixes as well
Two popular configuration management tools

• Autoconf
  – Unix-based
  – Game plan:
    • You write scripts to test availability on system
    • Generates Makefiles based on results

• Cmake
  – Unix and Windows
  – Game plan:
    • You write .cmake files that test for package locations
    • Generates Makefiles based on results

CMake has been gaining momentum in recent years, because it is one of the best solutions for cross-platform support.
Outline

• Review
• Project 1B Overview
• Build
  • Project 1C Overview
• Tar
• Character Strings
CIS 330: Project #1C
Assigned: April 7\textsuperscript{th}, 2016
Due April 12\textsuperscript{th}, 2016
(which means submitted by 6am on April 13\textsuperscript{th}, 2016)
Worth 2\% of your grade

Assignment: Download the file “Proj1C.tar”. This file contains a C-based project. You will build a Makefile for the project, and also extend the project.
Project 1C

== Build a Makefile for math330 ==

Your Makefile should:
(1) create an include directory
(2) copy the Header file to the include directory
(3) create a lib directory
(4) compile the .c files in trig and exp as object files (.o’s)
(5) make a library
(6) install the library to the lib directory
(7) compile the “cli” program against the include and library directory

== Extend the math330 library ==

You should:
(1) add 3 new functions: arccos, arcsin, and arctan (each in their own file)
(2) Extend the “cli” program to support these functions
(3) Extend your Makefile to support the new functions
Outline

• Review
• Project 1B Overview
• Build
• Project 1C Overview
• Tar
• Character Strings
Unix command: tar

• Anyone know what tar stands for?

tar = tape archiver

IBM tape library
Unix command: tar

• Problem: you have many files and you want to...
  – move them to another machine
  – give a copy to a friend
  – etc.

• Tar: take many files and make one file
  – Originally so one file can be written to tape drive

• Serves same purpose as “.zip” files.
Unix command: tar

• `tar cvf 330.tar file1 file2 file3`
  – puts 3 files (file1, file2, file3) into a new file called 330.tar

• `scp 330.tar @ix:~`

• `ssh ix`

• `tar xvf 330.tar`

• `ls`

  file1 file2 file
Outline

• Review
• Project 1B Overview
• Build
• Project 1C Overview
• Tar
• Character Strings
There have been various extensions to ASCII ...
now more than 128 characters
Many special characters are handled outside this convention

Image source: granneman.com
signed vs unsigned chars

• signed char ("char"):  
  – valid values: -128 to 127  
  – size: 1 byte  
  – used to represent characters with ASCII  
  • values -128 to -1 are not valid

• unsigned char:  
  – valid values: 0 to 255  
  – size: 1 byte  
  – used to represent data
character strings

• A character “string” is:
  – an array of type “char”
  – that is terminated by the NULL character

• Example:
  char str[12] = “hello world”;
  – str[11] = ‘\0’ (the compiler did this automatically)

• The C library has multiple functions for handling strings
Character strings example

```
128-223-223-72-wireless:330 hank$ cat string.c
#include <stdio.h>

int main()
{
    char str[12] = "hello world";
    char *str2 = str+6;

    printf("str is "\%s" and str2 is "\%s"\n", str, str2);

    str[5] = '\0';

    printf("Now str is "\%s" and str2 is "\%s"\n", str, str2);
}
128-223-223-72-wireless:330 hank$ gcc string.c
128-223-223-72-wireless:330 hank$ ./a.out
str is "hello world" and str2 is "world"
Now str is "hello" and str2 is "world"
```
Useful C library string functions

- `strcpy`: string copy
- `strncpy`: string copy, but just first N characters
- `strlen`: length of a string

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

int main()
{
    char str[12] = "hello world";
    char str2[6], str3[7];
    strcpy(str2, str+strlen("hello "));
    strncpy(str3, str, strlen("hello "));
    printf("%s,%s\n", str2, str3);
}
```
```bash
128-223-223-72-wireless:330 hank$ cat strcpy.c
128-223-223-72-wireless:330 hank$ gcc strcpy.c
128-223-223-72-wireless:330 hank$ ./a.out
world,hello
```
Useful C library string functions

- `strcpy`: string copy
- `strncpy`: string copy, but just first N characters
- `strlen`: length of a string

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

int main()
{
    char str[12] = "hello world";
    char str2[7], str3[6];
    strcpy(str2, str + strlen("hello "));
    strncpy(str3, str, strlen("hello "));
    printf("%s,%s\n", str2, str3);
}
```

```
128-223-223-72-wireless:330 hank$ gcc strcpy.c
128-223-223-72-wireless:330 hank$ ./a.out
world,hello world
```
# More useful C library string functions

## Functions

<table>
<thead>
<tr>
<th>Category</th>
<th>Function</th>
<th>Description</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Copying:</td>
<td>memcpy</td>
<td>Copy block of memory</td>
<td><a href="https://cplusplus.com">function</a></td>
</tr>
<tr>
<td></td>
<td>memmove</td>
<td>Move block of memory</td>
<td><a href="https://cplusplus.com">function</a></td>
</tr>
<tr>
<td></td>
<td>strcpy</td>
<td>Copy string</td>
<td><a href="https://cplusplus.com">function</a></td>
</tr>
<tr>
<td></td>
<td>strncpy</td>
<td>Copy characters from string</td>
<td><a href="https://cplusplus.com">function</a></td>
</tr>
<tr>
<td>Concatenation:</td>
<td>strcat</td>
<td>Concatenate strings</td>
<td><a href="https://cplusplus.com">function</a></td>
</tr>
<tr>
<td></td>
<td>strncat</td>
<td>Append characters from string</td>
<td><a href="https://cplusplus.com">function</a></td>
</tr>
<tr>
<td>Comparison:</td>
<td>memcmp</td>
<td>Compare two blocks of memory</td>
<td><a href="https://cplusplus.com">function</a></td>
</tr>
<tr>
<td></td>
<td>strcmp</td>
<td>Compare two strings</td>
<td><a href="https://cplusplus.com">function</a></td>
</tr>
<tr>
<td></td>
<td>strncoll</td>
<td>Compare two strings using locale</td>
<td><a href="https://cplusplus.com">function</a></td>
</tr>
<tr>
<td></td>
<td>strncmp</td>
<td>Compare characters of two strings</td>
<td><a href="https://cplusplus.com">function</a></td>
</tr>
<tr>
<td></td>
<td>strxfrm</td>
<td>Transform string using locale</td>
<td><a href="https://cplusplus.com">function</a></td>
</tr>
<tr>
<td>Searching:</td>
<td>memchr</td>
<td>Locate character in block of memory</td>
<td><a href="https://cplusplus.com">function</a></td>
</tr>
<tr>
<td></td>
<td>strchr</td>
<td>Locate first occurrence of character in string</td>
<td><a href="https://cplusplus.com">function</a></td>
</tr>
<tr>
<td></td>
<td>strcsppn</td>
<td>Get span until character in string</td>
<td><a href="https://cplusplus.com">function</a></td>
</tr>
<tr>
<td></td>
<td>strpbrk</td>
<td>Locate characters in string</td>
<td><a href="https://cplusplus.com">function</a></td>
</tr>
<tr>
<td></td>
<td>strrchr</td>
<td>Locate last occurrence of character in string</td>
<td><a href="https://cplusplus.com">function</a></td>
</tr>
<tr>
<td></td>
<td>strspn</td>
<td>Get span of character set in string</td>
<td><a href="https://cplusplus.com">function</a></td>
</tr>
<tr>
<td></td>
<td>strstr</td>
<td>Locate substring</td>
<td><a href="https://cplusplus.com">function</a></td>
</tr>
<tr>
<td></td>
<td>strtok</td>
<td>Split string into tokens</td>
<td><a href="https://cplusplus.com">function</a></td>
</tr>
<tr>
<td>Other:</td>
<td>memset</td>
<td>Fill block of memory</td>
<td><a href="https://cplusplus.com">function</a></td>
</tr>
<tr>
<td></td>
<td>strerror</td>
<td>Get pointer to error message string</td>
<td><a href="https://cplusplus.com">function</a></td>
</tr>
<tr>
<td></td>
<td>strlen</td>
<td>Get string length</td>
<td><a href="https://cplusplus.com">function</a></td>
</tr>
<tr>
<td>Macros:</td>
<td>NULL</td>
<td>Null pointer</td>
<td><a href="https://cplusplus.com">macro</a></td>
</tr>
<tr>
<td>Types:</td>
<td>size_t</td>
<td>Unsigned integral type</td>
<td><a href="https://cplusplus.com">type</a></td>
</tr>
</tbody>
</table>

source: [cplusplus.com](https://cplusplus.com)