Standard Library Functions

Prof. Jyotiprakash Mishra
mail@jyotiprakash.org

String Length and Copy

Program 1:

```c
#include <stdio.h>
#include <string.h>
int main() {
  char str1[] = "Hello";
  char str2[20];
  printf("Length of 'Hello': %lu\n",
    str1, strlen(str1));
  strcpy(str2, str1);
  printf("Copied: %s\n", str2);
  strncpy(str2, "World!!!", 5);
  str2[5] = '\0';
  printf("After strncpy: %s\n", str2);
  return 0;
}
```

Output:

```
Length of 'Hello': 5
Copied: Hello
After strncpy: World
```

Note:

```
strlen: returns string length
(excluding null terminator).
strcpy: copies entire string.
strncpy: copies at most n chars.
Always null-terminate after strncpy.
```

String Concatenation

Program 2:

```c
#include <stdio.h>
#include <string.h>
int main() {
  char str1[50] = "Hello";
  char str2[] = " World";
  char str3[50] = "Foo";
  printf("Before: %s\n", str1);
  strcat(str1, str2);
  printf("After strcat: %s\n", str1);
  strncat(str3, " Bar Baz", 4);
  printf("After strncat: %s\n", str3);
  return 0;
}
```

Output:

```
Before: Hello
After strcat: Hello World
After strncat: Foo Bar
```

Note:

```
strcat: appends source to
destination. Requires enough space.
strncat: appends at most n chars.
Automatically null-terminates.
```

String Comparison

Program 3:

```c
#include <stdio.h>
#include <string.h>
int main() {
  char *s1 = "apple";
  char *s2 = "banana";
  char *s3 = "apple";
  printf("strcmp(apple, banana): %d\n",
    strcmp(s1, s2));
  printf("strcmp(apple, apple): %d\n",
    strcmp(s1, s3));
  printf("strcmp(banana, apple): %d\n",
    strcmp(s2, s1));
  printf("strncmp(apple, apply, 4): %d\n",
    strncmp("apple", "apply", 4));
  return 0;
}
```

Output:

```
strcmp(apple, banana): -1
strcmp(apple, apple): 0
strcmp(banana, apple): 1
strncmp(apple, apply, 4): 0
```

Note:

```
strcmp: returns < 0 if s1 < s2,
0 if equal, > 0 if s1 > s2.
Lexicographic comparison.
strncmp: compares first n chars.
```

String Search

Program 4:

```c
#include <stdio.h>
#include <string.h>
int main() {
  char str[] = "Hello World";
  char *p;
  p = strchr(str, 'W');
  if (p) printf("Found 'W' at: %s\n", p);
  p = strrchr(str, 'l');
  if (p) printf("Last 'l' at: %s\n", p);
  p = strstr(str, "Wor");
  if (p) printf("Found 'Wor' at: %s\n", p);
  if (!strstr(str, "xyz"))
    printf("'xyz' not found\n");
  return 0;
}
```

Output:

```
Found 'W' at: World
Last 'l' at: ld
Found 'Wor' at: World
'xyz' not found
```

Note:

```
strchr: finds first occurrence of
char. strrchr: finds last occurrence.
strstr: finds substring. All return
pointer to match or NULL.
```

String Tokenization

Program 5:

```c
#include <stdio.h>
#include <string.h>
int main() {
  char str[] = "apple,banana,cherry,date";
  char *token;
  printf("Tokens:\n");
  token = strtok(str, ",");
  while (token != NULL) {
    printf("  %s\n", token);
    token = strtok(NULL, ",");
  }
  return 0;
}
```

Output:

```
Tokens:
  apple
  banana
  cherry
  date
```

Note:

```
strtok: splits string by delimiters.
First call with string, subsequent
calls with NULL. Modifies original
string (replaces delimiters with \0).
```

String to Number Conversion

Program 6:

```c
#include <stdio.h>
#include <stdlib.h>
int main() {
  char *s1 = "12345";
  char *s2 = "3.14159";
  char *s3 = "2147483647";
  char *s4 = "FF";
  int i = atoi(s1);
  double d = atof(s2);
  long l = atol(s3);
  long hex = strtol(s4, NULL, 16);
  printf("atoi: %d\n", i);
  printf("atof: %f\n", d);
  printf("atol: %ld\n", l);
  printf("strtol(hex): %ld\n", hex);
  return 0;
}
```

Output:

```
atoi: 12345
atof: 3.141590
atol: 2147483647
strtol(hex): 255
```

Note:

```
atoi: string to int.
atof: string to double.
atol: string to long.
strtol: string to long with base
(2-36). Returns 0 on error.
```

Memory Functions

Program 7:

```c
#include <stdio.h>
#include <string.h>
int main() {
  char str[20];
  int arr[5];
  memset(str, 'A', 10);
  str[10] = '\0';
  printf("After memset: %s\n", str);
  memset(arr, 0, sizeof(arr));
  printf("Array: %d %d %d\n",
    arr[0], arr[1], arr[2]);
  int src[] = {1, 2, 3, 4, 5};
  int dst[5];
  memcpy(dst, src, sizeof(src));
  printf("After memcpy: %d %d %d\n",
    dst[0], dst[1], dst[2]);
  return 0;
}
```

Output:

```
After memset: AAAAAAAAAA
Array: 0 0 0
After memcpy: 1 2 3
```

Note:

```
memset: fills memory with byte value.
Good for zeroing arrays/structs.
memcpy: copies n bytes from source
to destination. Fast block copy.
```

Memory Move and Compare

Program 8:

```c
#include <stdio.h>
#include <string.h>
int main() {
  char str[] = "Hello World";
  memmove(str + 6, str, 5);
  str[11] = '\0';
  printf("After memmove: %s\n", str);
  int a1[] = {1, 2, 3};
  int a2[] = {1, 2, 3};
  int a3[] = {1, 2, 4};
  printf("memcmp(a1, a2): %d\n",
    memcmp(a1, a2, sizeof(a1)));
  printf("memcmp(a1, a3): %d\n",
    memcmp(a1, a3, sizeof(a1)));
  return 0;
}
```

Output:

```
After memmove: Hello Hello
memcmp(a1, a2): 0
memcmp(a1, a3): -1
```

Note:

```
memmove: like memcpy but handles
overlapping regions safely.
memcmp: compares n bytes. Returns
0 if equal, <0 or >0 otherwise.
```

Random Numbers

Program 9:

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int main() {
  int i;
  srand(time(NULL));
  printf("Random numbers 0-99:\n");
  for (i = 0; i < 5; i++) {
    printf("%d ", rand() % 100);
  }
  printf("\nRandom numbers 1-6:\n");
  for (i = 0; i < 5; i++) {
    printf("%d ", (rand() % 6) + 1);
  }
  printf("\n");
  return 0;
}
```

Output:

```
Random numbers 0-99:
42 17 89 33 61
Random numbers 1-6:
3 5 1 4 6
```

Note:

```
srand: seeds random number generator.
Use time(NULL) for different sequence
each run. rand: generates random int.
Use modulo for range.
```

Absolute and Division

Program 10:

```c
#include <stdio.h>
#include <stdlib.h>
int main() {
  int n1 = -42, n2 = 17;
  printf("abs(%d) = %d\n", n1, abs(n1));
  printf("abs(%d) = %d\n", n2, abs(n2));
  div_t result = div(17, 5);
  printf("17 / 5: quot=%d, rem=%d\n",
    result.quot, result.rem);
  result = div(-17, 5);
  printf("-17 / 5: quot=%d, rem=%d\n",
    result.quot, result.rem);
  return 0;
}
```

Output:

```
abs(-42) = 42
abs(17) = 17
17 / 5: quot=3, rem=2
-17 / 5: quot=-3, rem=-2
```

Note:

```
abs: absolute value of integer.
div: divides and returns struct
with quotient and remainder.
Atomic operation. Also: labs, ldiv
for long.
```

Sorting with qsort

Program 11:

```c
#include <stdio.h>
#include <stdlib.h>
int compare(const void *a,
  const void *b) {
  return (*(int*)a - *(int*)b);
}
int main() {
  int arr[] = {5, 2, 8, 1, 9, 3};
  int n = 6, i;
  printf("Before: ");
  for (i = 0; i < n; i++)
    printf("%d ", arr[i]);
  qsort(arr, n, sizeof(int), compare);
  printf("\nAfter:  ");
  for (i = 0; i < n; i++)
    printf("%d ", arr[i]);
  printf("\n");
  return 0;
}
```

Output:

```
Before: 5 2 8 1 9 3
After:  1 2 3 5 8 9
```

Note:

```
qsort: generic quicksort. Takes
array, count, element size, and
comparison function. Sorts in-place.
Works with any type.
```

Binary Search

Program 12:

```c
#include <stdio.h>
#include <stdlib.h>
int compare(const void *a,
  const void *b) {
  return (*(int*)a - *(int*)b);
}
int main() {
  int arr[] = {1, 3, 5, 7, 9, 11, 13};
  int n = 7, key = 7, *found;
  found = bsearch(&key, arr, n,
    sizeof(int), compare);
  if (found)
    printf("Found %d\n", *found);
  key = 6;
  found = bsearch(&key, arr, n,
    sizeof(int), compare);
  if (!found)
    printf("%d not found\n", key);
  return 0;
}
```

Output:

```
Found 7
6 not found
```

Note:

```
bsearch: binary search in sorted
array. Returns pointer to match or
NULL. Array MUST be sorted first.
O(log n) complexity.
```

Math Functions - Basic

Program 13:

```c
#include <stdio.h>
#include <math.h>
int main() {
  double x = 16.0, y = -3.7;
  printf("sqrt(%.0f) = %.2f\n",
    x, sqrt(x));
  printf("pow(2, 8) = %.0f\n",
    pow(2, 8));
  printf("fabs(%.1f) = %.1f\n",
    y, fabs(y));
  printf("ceil(%.1f) = %.0f\n",
    y, ceil(y));
  printf("floor(%.1f) = %.0f\n",
    y, floor(y));
  return 0;
}
```

Output:

```
sqrt(16) = 4.00
pow(2, 8) = 256
fabs(-3.7) = 3.7
ceil(-3.7) = -3
floor(-3.7) = -4
```

Compile:

```
gcc prog.c -lm
```

Note:

```
Math functions need -lm flag.
sqrt, pow, fabs, ceil, floor.
```

Math Functions - Trigonometry

Program 14:

```c
#include <stdio.h>
#include <math.h>
#define PI 3.14159265
int main() {
  double angle = 45.0;
  double radians = angle * PI / 180.0;
  printf("sin(45deg) = %.4f\n",
    sin(radians));
  printf("cos(45deg) = %.4f\n",
    cos(radians));
  printf("tan(45deg) = %.4f\n",
    tan(radians));
  printf("asin(0.5) = %.2f rad\n",
    asin(0.5));
  return 0;
}
```

Output:

```
sin(45deg) = 0.7071
cos(45deg) = 0.7071
tan(45deg) = 1.0000
asin(0.5) = 0.52 rad
```

Note:

```
Trig functions use radians.
sin, cos, tan for angles.
asin, acos, atan for inverse.
Convert degrees: rad = deg * PI/180.
```

Math Functions - Logarithm

Program 15:

```c
#include <stdio.h>
#include <math.h>
int main() {
  double x = 2.718282;
  double y = 100.0;
  printf("log(e) = %.4f\n", log(x));
  printf("log10(100) = %.1f\n",
    log10(y));
  printf("exp(1) = %.6f\n", exp(1));
  printf("exp(2) = %.6f\n", exp(2));
  printf("log2(8) = %.1f\n", log2(8));
  return 0;
}
```

Output:

```
log(e) = 1.0000
log10(100) = 2.0
exp(1) = 2.718282
exp(2) = 7.389056
log2(8) = 3.0
```

Note:

```
log: natural logarithm (base e).
log10: base-10 logarithm.
log2: base-2 logarithm.
exp: e raised to power.
```

Math Functions - Rounding

Program 16:

```c
#include <stdio.h>
#include <math.h>
int main() {
  double nums[] = {2.3, 2.5, 2.7,
    -2.3, -2.5, -2.7};
  int i;
  for (i = 0; i < 6; i++) {
    printf("%.1f: floor=%.0f ceil=%.0f"
      " round=%.0f\n",
      nums[i], floor(nums[i]),
      ceil(nums[i]), round(nums[i]));
  }
  return 0;
}
```

Output:

```
2.3: floor=2 ceil=3 round=2
2.5: floor=2 ceil=3 round=2
2.7: floor=2 ceil=3 round=3
-2.3: floor=-3 ceil=-2 round=-2
-2.5: floor=-3 ceil=-2 round=-2
-2.7: floor=-3 ceil=-2 round=-3
```

Note:

```
floor: rounds down toward -infinity.
ceil: rounds up toward +infinity.
round: rounds to nearest integer.
```

Environment Variables

Program 17:

```c
#include <stdio.h>
#include <stdlib.h>
int main() {
  char *home = getenv("HOME");
  char *path = getenv("PATH");
  char *user = getenv("USER");
  if (home)
    printf("HOME: %s\n", home);
  if (user)
    printf("USER: %s\n", user);
  if (!getenv("MYVAR"))
    printf("MYVAR not set\n");
  return 0;
}
```

Output:

```
HOME: /Users/username
USER: username
MYVAR not set
```

Note:

```
getenv: retrieves environment
variable value. Returns NULL if
not found. Common vars: HOME, USER,
PATH, PWD.
```

System Command Execution

Program 18:

```c
#include <stdio.h>
#include <stdlib.h>
int main() {
  int ret;
  printf("Listing files:\n");
  ret = system("ls -l *.c 2>/dev/null");
  if (ret == 0) {
    printf("Command succeeded\n");
  } else {
    printf("Command failed or no files\n");
  }
  printf("\nDate:\n");
  system("date");
  return 0;
}
```

Output:

```
Listing files:
-rw-r--r-- 1 user user 234 Jan 1 prog.c
Command succeeded

Date:
Fri Jan 17 00:00:00 UTC 2026
```

Note:

```
system: executes shell command.
Returns exit status. Security risk
if command contains user input.
```

Program Termination

Program 19:

```c
#include <stdio.h>
#include <stdlib.h>
void cleanup1() {
  printf("Cleanup 1 called\n");
}
void cleanup2() {
  printf("Cleanup 2 called\n");
}
int main() {
  atexit(cleanup1);
  atexit(cleanup2);
  printf("Program running\n");
  printf("Exiting normally\n");
  return 0;
}
```

Output:

```
Program running
Exiting normally
Cleanup 2 called
Cleanup 1 called
```

Note:

```
atexit: registers function to call
at program exit. Called in reverse
order of registration. Up to 32
functions. Use for cleanup.
```

Character Classification

Program 20:

```c
#include <stdio.h>
#include <ctype.h>
int main() {
  char chars[] = {'a', 'Z', '5', ' ',
    '\n', '!'};
  int i;
  for (i = 0; i < 6; i++) {
    char c = chars[i];
    printf("'%c': ", c == '\n' ? 'n' : c);
    if (isalpha(c)) printf("alpha ");
    if (isdigit(c)) printf("digit ");
    if (isspace(c)) printf("space ");
    if (isupper(c)) printf("upper ");
    if (islower(c)) printf("lower ");
    printf("\n");
  }
  return 0;
}
```

Output:

```
'a': alpha lower
'Z': alpha upper
'5': digit
' ': space
'n': space
'!':
```

Note:

```
ctype.h character tests:
isalpha, isdigit, isspace, isupper,
islower, isalnum, ispunct.
Also: toupper, tolower for conversion.
```