CS101C: Introduction to Programming (Using C)

Autumn 2025

Nikhil Hegde
Achyut Mani Tripathi

Week9: Recap of Functions, Recursion, Review of Midsem Exam Paper

Recap: Functions

```
int main(){
      int a=10;
      int b=20;
      swap(a, b);
      printf(“a=%d b=%d”,a,b); // prints a=10, b=20
}
```

Call-by-value

Recap: Functions

```
void swap(int a, int b){
      int tmp = a;
      a = b;
      b=tmp;
      return;
}
```

Recap: Functions

```
int main(){
      int a=10;
      int b=20;
      swap2(&a, &b);
      printf("a=%d b=%d",a,b); // prints a=20, b=10
}
```

Call-by-reference

Recap: Functions

```
void swap2(int *a, int *b){
                int tmp = *a;
                *a = *b;
                *b=tmp;
                return;
}
```

Recap: Functions Declaration vs. Function Definition

- `void swap2(int *a, int *b); //declaration`
- `//definition follows`

```
void swap2(int *a, int *b){
                int tmp = *a;
                *a = *b;
                *b=tmp;
                return;
            }
```

Recap: Functions Declaration vs. Function Definition

- Why you need a declaration?

So that you do not have to define the function before the function call site.

  - You can define a function after the call site in the same .c file
  - You can define a function in another .c file!

Recursion

- Function calling itself!

```
int factorial(int n)
{
   if(n == 0)
      return 1;
   else
      return n * factorial(n-1);
}
```

Recursion

- Better to think of recursion as a problem solving technique rather than a programming principle.
- A common pattern in problem solving:
  1. Break the problem into smaller problems
  2. Apply the same function to solve the smaller problems
  3. Use the solutions created in previous step to solve original problem

Recursion

- Is the pattern never ending?

  No.

- Repeating the process creates smaller and smaller problems. Eventually, the problem becomes *trivial* to solve.

trivial problem = base case

Example - Factorial

- n! is just n * (n-1)!

Break the problem into smaller version of the same problem (step 1)

```
int factorial(int n)
{
   if(n == 0)
      return 1;
   else
      return n * factorial(n-1);
}
```

Example - Factorial

- call `factorial` again to solve the smaller problem

Solve the smaller problem by calling the same function (step 2)

```
int factorial(int n)
{
   if(n == 0)
      return 1;
   else
      return n * factorial(n-1);
}
```

Example - Factorial

- Multiply the result of previous step (calling `factorial(n-1)`) by n to find `factorial(n)`

Use the solution of the smaller problem to solve the original problem (step 3)

```
int factorial(int n)
{
   if(n == 0)
      return 1;
   else
      return n * factorial(n-1);
}
```

Example - Factorial

- The *base case* is simple: we know that `factorial(0) = 1`

```
int factorial(int n)
{
   if(n == 0)
      return 1;
   else
      return n * factorial(n-1);
}
```

Why recursive codes?

- Intuitive
  - Easier way to think of a solution
- Sometimes, the only way to effectively solve a problem!

Why recursive codes work?

- Think *inductively:*
  - Assume that the recursive function already works, but.. only on smaller problems than the original problem
  - Write recursive function for the original problem assuming it works
  - Write correct base case

Why recursive codes work?

- `Factorial` example*:*
  - Assume that `factorial(n-1)` works
  - If we have `(n-1)!` computing `n!` is easy: just multiply by n
  - Make sure that there exists a working base case: provide answer to the smallest argument passed to `factorial`

Divide-and-Conquer – A common recursive pattern

- Computing sum of array elements – toy example

```
int sum(int * arr, int nels)
{
   if (nels == 1)
      return arr[0];
   int sum1 = sum(arr, nels/2);
   int sum2 = sum(&arr[nels/2], (nels + 1)/2);
   return sum1 + sum2;
}
```

Divide-and-Conquer – A common recursive pattern

- Computing sum of array elements – toy example

```
int sum(int * arr, int nels)
{
   if (nels == 1)
      return arr[0];
   int sum1 = sum(arr, nels/2);
   int sum2 = sum(&arr[nels/2], (nels + 1)/2);
   return sum1 + sum2;
}
```

Divide-and-Conquer

- A problem can be broken into two or more smaller problems of similar or related type
- More realistic examples:

  Quicksort, Merge sort, finding closest pair of points

Recursion – observations

- Can have multiple base cases
  - Fibonacci series
- Tail recursion
  - `Factorial`

Midsem Exam Revision

Return type of the built-in function that computes the length of a string in C is:

strlen

<cstring>

```
size_t strlen ( const char * str );
```

Get string length

Returns the length of the C string ***str***.

The length of a C string is determined by the terminating null-character: A ***C string*** is as long as the number of characters between the beginning of the string and the terminating null character (without including the terminating null character itself).

This should not be confused with the size of the array that holds the string. For example:

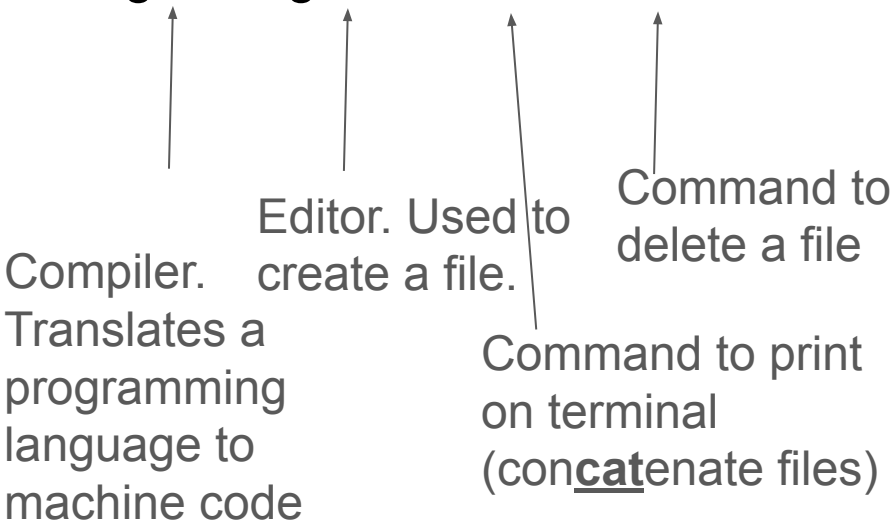
```
char mystr[100]="test string";
```

defines an array of characters with a size of 100 `char`s, but the C string with which ***mystr*** has been initialized has a length of only 11 characters. Therefore, while `sizeof(mystr)` evaluates to `100`, `strlen(mystr)` returns `11`.

Midsem Exam Revision

Select the correct answer after matching: 1. Command to create a file. 2. Command to translate a human-readable file to binary. 3. Command to print a file on the terminal. 4. Command to delete a file

A. gcc B. gedit c. cat d. rm e. del



Answer option a or e will be awarded marks.

Midsem Exam Revision

```
nikhilh@DESKTOP-PMT9DGE:~/cs101/midsemexam$ cat 3.c
#include<stdio.h>
int main(){
        int x=10;
        printf("%d ",x++);
        printf("%d ",++x);
        printf("%d ",x++);
}
nikhilh@DESKTOP-PMT9DGE:~/cs101/midsemexam$ gcc 3.c
nikhilh@DESKTOP-PMT9DGE:~/cs101/midsemexam$ ./a.out
10 12 12 nikhilh@DESKTOP-PMT9DGE:~/cs101/midsemexam$
```

Midsem Exam Revision

The following code snippet: char* x=”CSE”; printf(“%c”,1[x]); when executed, would:

```
nikhilh@DESKTOP-PMT9DGE:~/cs101/midsemexam$ cat 4.c
#include<stdio.h>
int main(){
        char* x="CSE";
        printf("%c",1[x]);
}
nikhilh@DESKTOP-PMT9DGE:~/cs101/midsemexam$ gcc 4.c
nikhilh@DESKTOP-PMT9DGE:~/cs101/midsemexam$ ./a.out
Snikhilh@DESKTOP-PMT9DGE:~/cs101/midsemexam$
```

Midsem Exam Revision

The following code snippet: int x=5; printf(“%zu”,sizeof(x)); when executed, would print:

```
nikhilh@DESKTOP-PMT9DGE:~/cs101/midsemexam$ cat 5.c
#include<stdio.h>
int main(){
 int x=5; printf("%zu",sizeof(x));
}
nikhilh@DESKTOP-PMT9DGE:~/cs101/midsemexam$ gcc 5.c
nikhilh@DESKTOP-PMT9DGE:~/cs101/midsemexam$ ./a.out
4nikhilh@DESKTOP-PMT9DGE:~/cs101/midsemexam$
```

Midsem Exam Revision

Assuming that integer variable y has been defined and initialized, the correct way of initializing a pointer in C is:

```
int *x=&y;

int* x=&y;

int * x=&y;
```

(not the blankspaces. They don’t matter here.)

Midsem Exam Revision

Assuming that integer variables a=10 and b=0, the result of the expression a&&b is:

```
nikhilh@DESKTOP-PMT9DGE:~/cs101/midsemexam$ cat 7.c
#include<stdio.h>
int main(){
 int a=10, b=0;
 printf("%d",a&&b);
}
nikhilh@DESKTOP-PMT9DGE:~/cs101/midsemexam$ gcc 7.c
nikhilh@DESKTOP-PMT9DGE:~/cs101/midsemexam$ ./a.out
0nikhilh@DESKTOP-PMT9DGE:~/cs101/midsemexam$
```

Midsem Exam Revision

The condition in if(a=100) has a value: 100

```
nikhilh@DESKTOP-PMT9DGE:~/cs101/midsemexam$ cat 8.c
#include<stdio.h>
int main(){
 int a=10;
 printf("%d",a=100);
}
nikhilh@DESKTOP-PMT9DGE:~/cs101/midsemexam$ gcc 8.c
nikhilh@DESKTOP-PMT9DGE:~/cs101/midsemexam$ ./a.out
100nikhilh@DESKTOP-PMT9DGE:~/cs101/midsemexam$
```

Midsem Exam Revision

Assuming that integer variables a=64 and b=2, the result of the expression a>>b is:

```
nikhilh@DESKTOP-PMT9DGE:~/cs101/midsemexam$ cat 9.c
#include<stdio.h>
int main(){
 int a=64, b=2;
 printf("%d",a>>b);
}
nikhilh@DESKTOP-PMT9DGE:~/cs101/midsemexam$ gcc 9.c
nikhilh@DESKTOP-PMT9DGE:~/cs101/midsemexam$ ./a.out
16nikhilh@DESKTOP-PMT9DGE:~/cs101/midsemexam$
```

Midsem Exam Revision

The binary equivalent of 0xC5E is: 1100 0101 1110

Midsem Exam Revision

Assuming that integer variables a=10, b=3 the result of the expression (a=b+2, a*2) is:

```
nikhilh@DESKTOP-PMT9DGE:~/cs101/midsemexam$ cat 11.c
#include<stdio.h>
int main(){
 int a=10, b=3;
 int c=(a=b+2, a*2);
 printf("%d",c);
}
nikhilh@DESKTOP-PMT9DGE:~/cs101/midsemexam$ gcc 11.c
nikhilh@DESKTOP-PMT9DGE:~/cs101/midsemexam$ ./a.out
10nikhilh@DESKTOP-PMT9DGE:~/cs101/midsemexam$
```

Midsem Exam Revision

Assuming that integer variables a=10,b=20,c=5, and d=0 the result of the expression (d+=a?b:c) is:

```
nikhilh@DESKTOP-PMT9DGE:~/cs101/midsemexam$ cat 12.c
#include<stdio.h>
int main(){
 int a=10, b=20, c=5, d=0;
 d+=a?b:c;
 printf("%d",d);
}
nikhilh@DESKTOP-PMT9DGE:~/cs101/midsemexam$ gcc 12.c
nikhilh@DESKTOP-PMT9DGE:~/cs101/midsemexam$ ./a.out
20nikhilh@DESKTOP-PMT9DGE:~/cs101/midsemexam$
```

Midsem Exam Revision

```
nikhilh@DESKTOP-PMT9DGE:~/cs101/midsemexam$ cat 13.c
#include<stdio.h>
int main(){
        int a=15;
        if(a<15)
                if(a<5)
                        printf("a is less than 5\n");
                else if(a<8)
                        printf("a is >=5 but < 8\n");
                else if(a<12)
                        printf("a is >=8 but < 12\n");
        else
                printf("a is >=15 ");
        printf("BYE\n");
}
nikhilh@DESKTOP-PMT9DGE:~/cs101/midsemexam$ gcc 13.c
nikhilh@DESKTOP-PMT9DGE:~/cs101/midsemexam$ ./a.out
BYE
```

Midsem Exam Revision

The following code snippet, when executed, would print

```
nikhilh@DESKTOP-PMT9DGE:~/cs101/midsemexam$ cat 14.c
#include<stdio.h>
int main(){
        int i = 10, j = 20;
        while (i<25,j<25){
                i++;
                j++;
        }
        printf("%d %d", i, j);
}
nikhilh@DESKTOP-PMT9DGE:~/cs101/midsemexam$ gcc 14.c
nikhilh@DESKTOP-PMT9DGE:~/cs101/midsemexam$ ./a.out
15 25nikhilh@DESKTOP-PMT9DGE:~/cs101/midsemexam$
```

Midsem Exam Revision

In the following code snippet, which line would be a syntax error?

1. int x1[]={‘C’,’S’,’E’,0};
2. int *x2=x1;
3. int x3[3]={0x43, 0x53, 0x45};
4. int x4[4]={0x43, 0x53, 0x45, 0};

None. All are valid.

- 1 is initializing an integer array using initializer list. The initialized values are equivalent ASCII values of characters and 0.
- 2 is initializing a pointer. 3 (and 4) same as 1 but initializing hexadecimal values (and 0)

Midsem Exam Revision

The following code snippet, when executed, would print

```
nikhilh@DESKTOP-PMT9DGE:~/cs101/midsemexam$ cat 19.c
#include<stdio.h>
int main(){
        char str[] = "%d %c", arr[] = "CS101CExam";
        printf(str, 0[arr], 2[arr + 3]);
        return 0;
}
nikhilh@DESKTOP-PMT9DGE:~/cs101/midsemexam$ gcc 19.c
nikhilh@DESKTOP-PMT9DGE:~/cs101/midsemexam$ ./a.out
67 Cnikhilh@DESKTOP-PMT9DGE:~/cs101/midsemexam$
```

Midsem Exam Revision

The following code snippet, when executed, would print

```
nikhilh@DESKTOP-PMT9DGE:~/cs101/midsemexam$ cat 20.c
#include<stdio.h>
#include<string.h>
int main(){
        char str[] = "I love mess food";
        printf("%zu", strlen(str));
}
nikhilh@DESKTOP-PMT9DGE:~/cs101/midsemexam$ gcc 20.c
nikhilh@DESKTOP-PMT9DGE:~/cs101/midsemexam$ ./a.out
16nikhilh@DESKTOP-PMT9DGE:~/cs101/midsemexam$
```