CS101C: Introduction to Programming (Using C)

Autumn 2025

Nikhil Hegde
Achyut Mani Tripathi

Week10: Recursion (continued), Structures

Recursion – a real life example

"This is an increasingly common occurrence in our political **discourse**."
Washington Post Jun 25, 2019

discourse:
a formal discussion of a subject in speech or writing, as a dissertation, **treatise**, sermon, etc.

treatise:
a formal and systematic **exposition** in writing of the principles of a subject, generally longer andmore detailed than an essay.

exposition:
the act of **expounding**, setting forth, or explaining.

```
void LookUpDictionary(string n) {
   array<string> retVal = GetMeaning(n)
   foreach element in retVal:
      if meaning of element is known
         continue;
      else
         LookUpDictionary(element);
}
```

Recap: Example - Factorial

- n! = n x (n-1) x (n-2) x . . . x 3 x 2 x 1

(n-1)! = (n-1) x (n-2) x . . . x 3 x 2 x 1

therefore,

n! = n x (n-1)!

is this complete?

- plug 0 to n and the equation breaks.

therefore,

$$n! = \begin{cases} n \times (n-1)! & \text{when } n \geq 1 \\ 1 & \text{when } n = 0 \end{cases}$$

// factorial of negative numbers not defined.

Example - Factorial

$$n! = \begin{cases} n \times (n-1)! & \text{when } n \geq 1 \\ 1 & \text{when } n = 0 \end{cases}$$

// factorial of negative numbers not defined.

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

Example - Factorial

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

Today’s class (8/10/2025)

- Recursion (Recap)
- Structures

Exercise

```
1 int ex1(char* str)
2 {
3  if(*str == ‘\0’)
4     return 0;
5  else
6     return 1 + ex1(str+1);
7 }
```

what does the function ex1 do?

Structures in C

- Used to group dissimilar variables into logical entities
- Example: Name, branch, and CPI of a student.
- Name is a character array, branch is an integer, and CPI is a floating-point number. But all these 3 values (name, branch, and CPI) are attributes of one particular student.

```
struct student {
    char name[100];
    int branch;
    float cpi;
};
```

Structures in C

```
struct student {
    char name[100];
    int branch;
    float cpi;
};
```

- Highlights declaration of a structure and the name by which the structure will be known.

Structures in C

```
struct student {
    char name[100];
    int branch;
    float cpi;
};
```

- Individual members of the structure ‘student’
- Each variable holds the value corresponding to each attribute of the logical entity we are dealing with in the program

Structures in C

```
struct student {
    char name[100];
    int branch;
    float cpi;
};

struct student student1, student2;
struct student students[10];
```

In the example shown:

- `student1`, and `student2` are variables of type `struct student`.
- `struct student` can be thought of as a new user-defined type.
- `struct student` can also be used just like built-in types (`int`, `char`, and `float`) when passing it to functions and returning from functions.
- **Each variable will have its associated `name`, `branch` and `cpi`**
- `students[10]` is an array of 10 elements of type `struct student`

Structures in C

```
struct student {
    char name[100];
    int branch;
    float cpi;
};
struct student student1, student2;

strcpy(student1.name, “Tom”);
student1.branch = 10;
student1.cpi = 10.0;
```

In the example shown:

- The member variables of `student1` are initialized (or assigned new values)
- Each member variable of the structure are assigned values in separate statements.

Structures in C

```
struct location {
    float latitude;
    float longitude;
};

struct user {
    char name[100];
    struct location loc;
};

struct user user1;
user1.loc.latitude = 23.7;
```

One structure can be inside another structure.

In the example shown:

- A variable of type `struct location` is inside another variable of type `struct user`.
- **The member variable inside the variable `loc` (that is inside `user1`) is accessed by using multiple `.` (dots).**

Structures in C

```
struct item {
    char code;
    int quantity;
    float price;
};
struct item soap, brush;

soap = {'s', 2, 12.5};
brush = {'b', 1, 25.0};
```

In the example shown:

- The member variables of `soap`, `brush` (of type `struct item`) are initialized in a single statement.
- **NOTE this style of initialization does not work for `char` arrays (or strings).**

Structures: Demo

```c
#include<stdio.h>
#include<string.h>
//below is the definition of a structure. note the keyword struct, curly braces, and semicolon.
struct student{
    char name[100];
    int branch;
    float cpi;
};

//another struct definition.
struct personalinfo{
    int phone;
    char gender;
    float age;
};

//another struct definition to demonstrate nested structures.
struct studentv2{
    char name[100];
    int branch;
    float cpi;
    struct personalinfo info;
};
```

```c
int main(){
    //definition of a variable of type student
    struct student s1;
    //initialization (Writing to) of the fields of the struct student.
    //note the dot notation to access the fields.
    s1.branch=1;
    s1.cpi=2.5;
    //s1.name="Sam"; is this allowed? No. Why? "Sam" is a string constant and its type is char *.
    //on the lhs we have s1.name, which is an array of characters and its type is char []. type mismatch between lhs and rhs.
    //so, one way to initialize the name field of s1 variable is:
    strcpy(s1.name, "Rama");
    printf("student name:%s student branch:%d student cpi:%f\n",s1.name, s1.branch, s1.cpi);
    //initialize the fields.
    struct personalinfo i1= {12345678,'M',45};
    printf("contact number:%d gender:%c age:%f\n",i1.phone, i1.gender, i1.age);
    //define a variable of type struct studentv2
    struct studentv2 s2;
    //write values into the info field of s2:
    s2.info=i1;
    //write other values into s2's fields.
}
```

see struct_demo.c shared in code examples

Today’s class (10/10/2025)

- Array of Structures
- Address of structure variables and structure fields
- Passing structures to functions

Structures: demo

```c
//below function modifies the cpi field of a struct student variable. This function is an example of call-by-reference.
void cpimodifier(struct student* s){
    (*s).cpi=5.0; //note that we can use s->cpi=5.0 as well
    return;
}

int main(){
    //below is an example of an array of structure: students is an array of struct student of size 2.
    struct student students[2];

    //below code accepts user input and writes them into the fields of struct student array elements. You can uncomment this code and enter input from terminal.
    /*printf("Enter student details (name, branch, cpi)\n");
    for(int i=0;i<2;i++){
        //note. you can also write this line as: scanf("%s",&students[i].name); However, you get a warning.
        //The reason is that the type of students[i].name is "char [96]". When you say &students[i].name, you are
        //trying to get the address and so the address must be stored in a pointer variable of type char (*)[96].
        //However, %s in scanf expects a "char *" argument.
        //If you pass the argument as students[i].name, then because array name is synonymous with the address of the first element of the array, no warning is seen.
        //we satisfy
        scanf("%s",students[i].name);
        scanf("%d",&students[i].branch);
        scanf("%f",&students[i].cpi);
    }*/
    //below code writes to the cpi field of the struct student variable (first element of the array)
    students[0].cpi=10.0;
    printf("gpa of student[0] (before calling gpamodifier): %f\n",students[0].cpi);

    //print the size of the struct student. Uncomment this line to see the size.
    //printf("size of struct student: %zu\n",sizeof(struct student));

    //call the function cpimodifier and pass the first element of the array students. The first element is of type struct student. We need to pass the address of this.
    //hence, we pass &students[0]
    cpimodifier(&students[0]);
```

see struct_demo2.c shared in code examples