FUNCTIONS

PROGRAMMING LANGUAGES

BY

Z. CIHAN TAYSI

OUTLINE

- Passing arguments
  - pass by reference, pass by value
- Declarations and calls
  - definition, allusion, function call
- Examples
- Recursion
- The main function
- Function pointers

PASSING ARGUMENTS

- Because C passes arguments by value, a function can assign values to the formal arguments without affecting the actual arguments

- If you do want a function to change the value of an object, you must pass a pointer to the object and then make an assignment through the dereferenced pointer.
  - ***remember scanf function !!!***

DECLARATIONS AND CALLS

- Definition
  - Actually defines what the function does, as well as number and type of arguments
- Function Allusion
  - Declares a function that is defined somewhere else
- Function Call
  - Invokes a function, causing program execution to jump to the next invoked function. When the function returns, execution resumes at the point just after the call

FUNCTION ALLUSION

- A few examples

```
void simpleFunction1(  void ); // prototype of last example
simpleFunction1();

extern float simpleFunction2();
int factorial( int );
void sortArray(int *, int);
float *mergeSort(float  *, int, float *, int, int *);
```

FUNCTION DEFINITION

- A very simple example
  - no arguments
  - no return

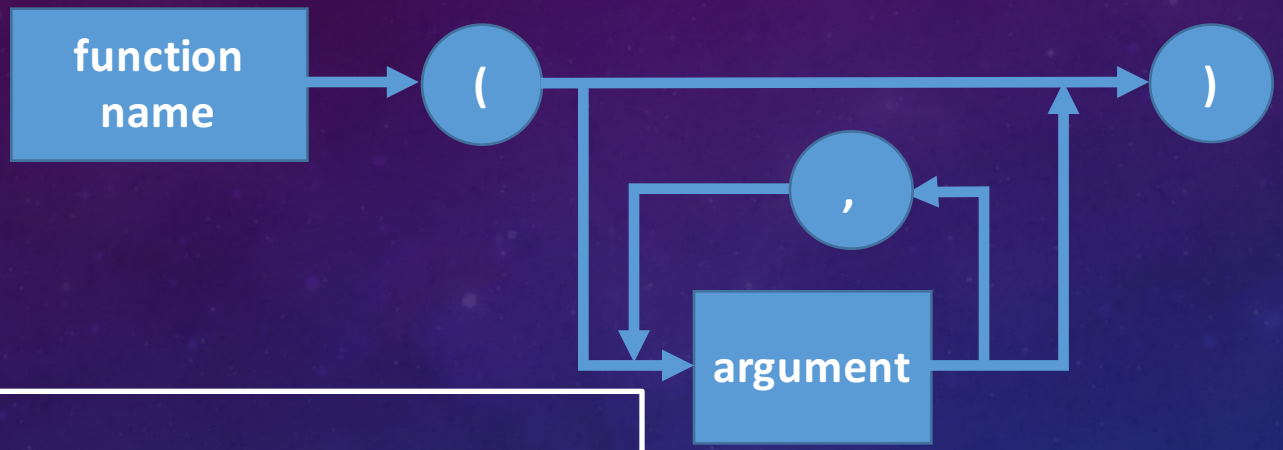
- A relatively complex example
  - a function to calculate factorial n

```
void simpleFunction1  ( void )  {

        printf("\nThis  is simpleFunction1\n");
}

int factorial( int n)  {
        int i,f=1;
        for(i=2;i<=n;i++)
                f = f *  i;
        return f;
}
```

FUNCTION CALL




```
printf("Hello World\n");
printf("Result is %d\n", factorial(10));
scanf("%s", str);
x = factorial(n) / factorial(m);
```

```
printf("%d : %d : %s : %d\n", i, j, line, rc);
matT = transpose(mat, rows, cols);
```

ORDER OF FUNCTIONS

- In order to use a function you must define it beforehand.
  - In order to use your own function in the ***main() function***, you should define it ***before the main()*** in the same file

- It is also possible to use function allusion (function prototype)
  - You can write the prototype of your function before the ***main() function*** and use it anywhere (main() or any other function of yours)

EXAMPLE 1

EXAMPLE 2

EXAMPLE 3

FUNCTION ARGUMENTS

PASSING ARRAYS AS FUNCTION PARAMETER

- Several ways to do it…
- Do NOT forget
  - No boundary checking !
  - remember your motivation to create a function
- Using actual array size
  - void myFunction(int ar[5])
- Using array and a size parameter
  - void myFunction(int ar[], int size)
- Using a pointer and an integer
  - void myFunction(int *ar, int size)

EXAMPLE

- Create a sort function for one dimensional arrays
- Use any type of sorting algorithm

HOWTO RETURN AN ARRAY FROM A FUNCTION

- We don't return an array from functions, rather we return a pointer holding the base address of the array to be returned.
- We must, make sure that the array exists after the function ends!
  - you can **NOT** return local arrays!
- **SOLUTION** : dynamic memory allocation

EXAMPLE

- Write a function that return compresses a sparse matrix
- The function should take a matrix as a parameter
- The function should return a new matrix 3 x n or n x 3

RECURSION

- A recursive function is one that calls itself.
  - An example is given on the right side
- It is important to notice that this function will call itself forever.
  - Actually not forever, but till the computer runs out of stack memory
  - It means a runtime error
- Thus, remember to include a stop point in your recursive functions.

```
void recurse ()  {
        static count = 1;
        printf("%d\n",  count);
        count++;
        recurse();
}
main()  {
        recurse();
}
```

RECURSION

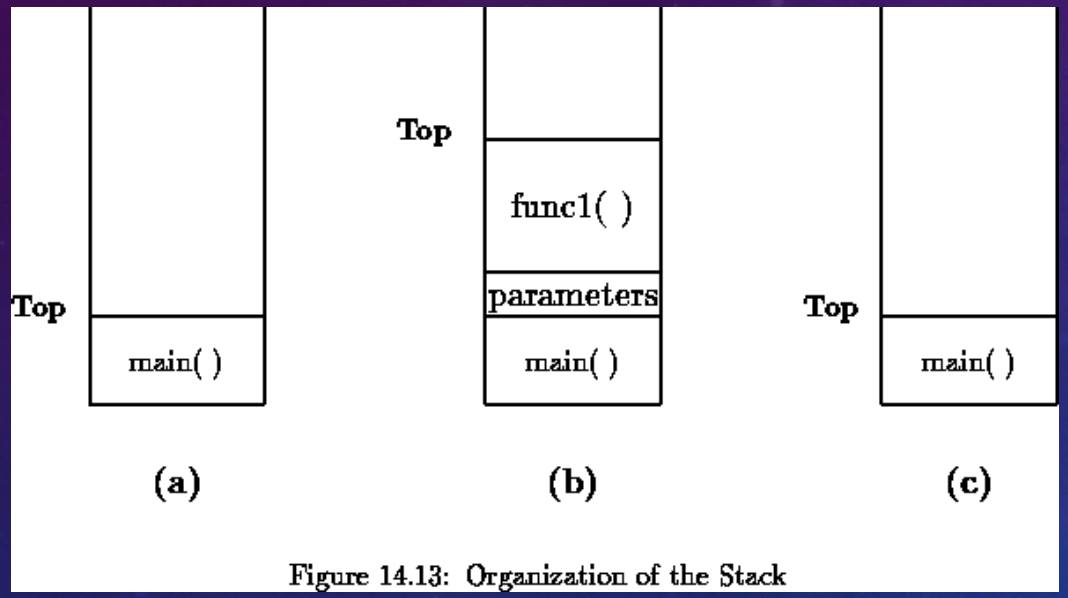


Figure 14.13: Organization of the Stack

- When a program begins executing in the function main(), space is allocated on the stack for all variables declared within main(), - **Figure 14.13(a)**
- If main() calls a function, func1(), additional storage is allocated for the variables in func1() at the top of the stack - **Figure 14.13(b)**
  - Notice that the parameters passed by main() to func1() are also stored on the stack.
- When func1() returns, storage for its local variables is deallocated, and the Top of the stack returns to tthe position - **Figure 14.13(c)**
- As can be seen, the memory allocated in the stack area is used and reused during program execution.
  - *It should be clear that memory allocated in this area will contain garbage values left over from previous usage.*

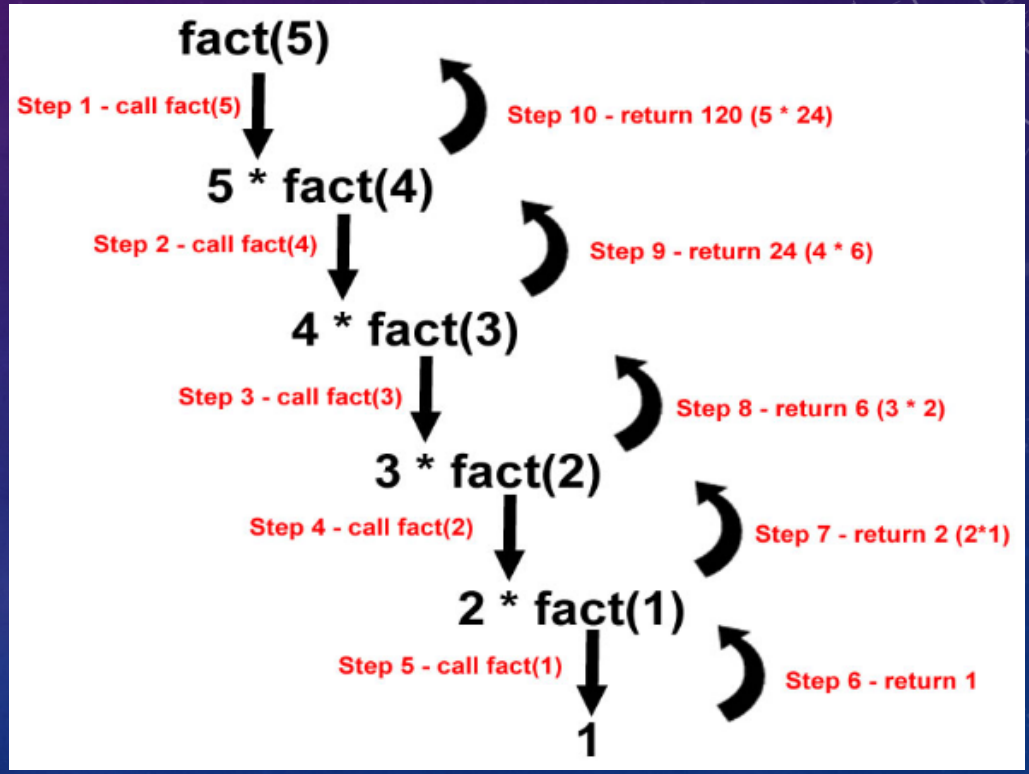
RECURSION

- A few examples to solve with recursion
  - Factorial – n!
  - Fibonacci numbers – $F_{n+1} = F_n + F_{n-1}$
  - Binary search
  - Depth-first search

```
int fact( int n )  {
        if( n <= 1 )
                return 1;
        else
                return n*fact(n-1);
}
main()  {
        printf("5! is %d\n", fact(5));
}
```

RECURSION

- A few examples to solve with recursion
  - Factorial – n!
  - Fibonacci numbers – $F_{n+1} = F_n + F_{n-1}$
  - Binary search
  - Depth-first search



MAIN() FUNCTION

- All C programs must contain a function called ***main()***, which is always the first function executed in a C program.
- When ***main()*** returns, the program is done.
- The compiler treats the main() function like any other function, except that at runtime the host environment is responsible for providing two arguments
  - **argc** – number of arguments that are presented at the command line
  - **argv** – an array of pointers to the command line arguments

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

        while(--argc > 0 )
                printf("%s\n",  *++argv);
        exit(0);
}
```

MAIN() FUNCTION

- A better way to handle command line arguments
  - getopt
  - argp
  - suboptions

```
while ((c = getopt (argc, argv, "abc:")) != -1)
  switch (c)
    {
    case 'a':
      aflag = 1;
      break;
....
    default:
      abort ();
    }
```

http://www.gnu.org/software/libc/manual/html_node/Example-of-Getopt.html#Example-of-Getopt

FUNCTION POINTERS