CS240: Programming in C

Lecture 6: Recursive Functions. C Pre-processor.

PURDUE UNIVERSITY

Functions: extern and static

- Functions can be used before they are declared
- **static** for a function means the function is local only to that file
- **extern**, means that the function was declared in another file or the same file but later
- Always put prototype before definition to avoid any problems

Variables

- All variables must be declared before use
- **extern** has the same meaning as for functions
- **static** the same when declared outside functions
- **static** declared within a function ‘has memory’, i.e is initialized only the first time the function is called
- Do not use the same names for global and local variables

Static modifier: Example

```
int good_memory(void) {
   static int val = 10;
   printf("val %d\n", val++);
}

int bad_memory(void) {
   int val = 10;
   printf("val %d\n", val++);
}
```

Passing Parameters

- In C, parameters are passed to functions BY VALUE
- Functions create local copies of those variables
- Modifications are not preserved outside the functions unless the function is passed references to variables
  - int swap(int[])

```
void swap2(int vec[]) {
    int tmp;
    tmp = vec[0];
    vec[0] = vec[1];
    vec[1] = tmp;
}
int main() {
    int vec[2] = {10, 20};
    swap2(vec);
    return 0;
}
```

Recursive functions in C

- A function can call itself
  - Recursive expression of the function
  - Needs a stop condition

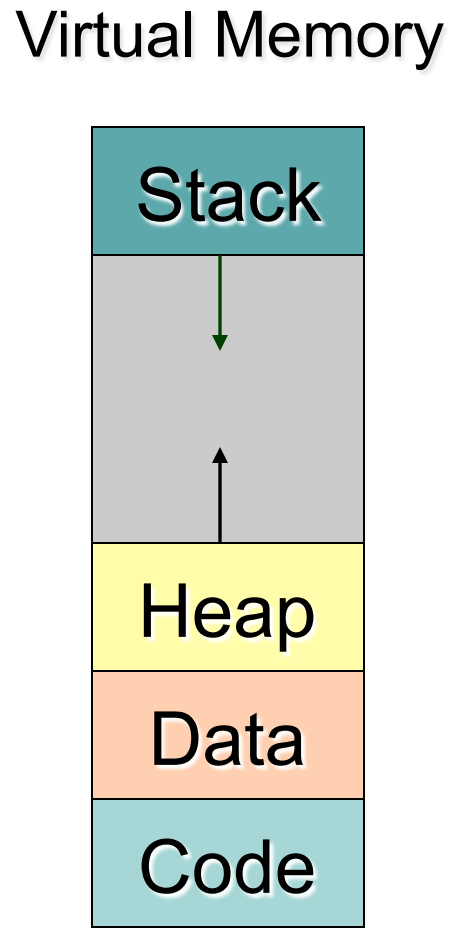
  Example: compute n!

```
int fact(n) {
     if(n<=1)
              return 1;
     else
       return n*fact(n-1);
 }
```

- Why does it work?
- It's magic!

actually… it's all about the Stack

- The operating system creates a process by assigning memory and other resources
- **Stack**: keeps track of the point to which each active subroutine should return control when it finishes executing; stores variables that are local to functions
- **Heap**: dynamic memory for variables that are created with *malloc, calloc, realloc* and disposed of with *free*
- **Data**: initialized variables including global and static variables, un-initialized variables
- **Code**: the program instructions to be executed

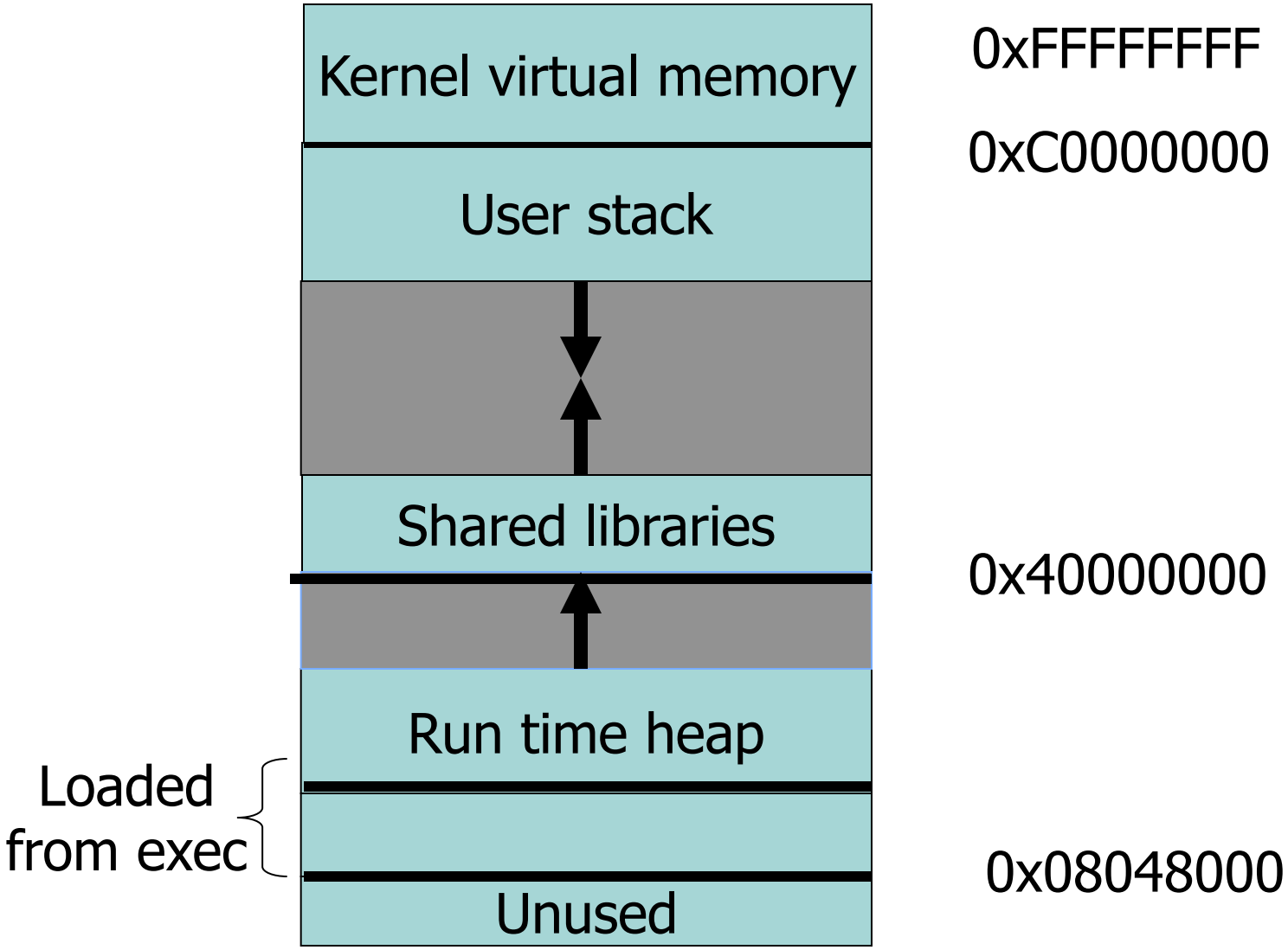


Stack

- Logically it's a LIFO structure
- Two operations: push and pop
- Grows 'down'
- Operations always happen at the top: push and pop, organized
- It provides support for recursive functions

- It stores not only the local variables but also the address of the function that needs to be executed next

Example: Linux Process Memory Layout

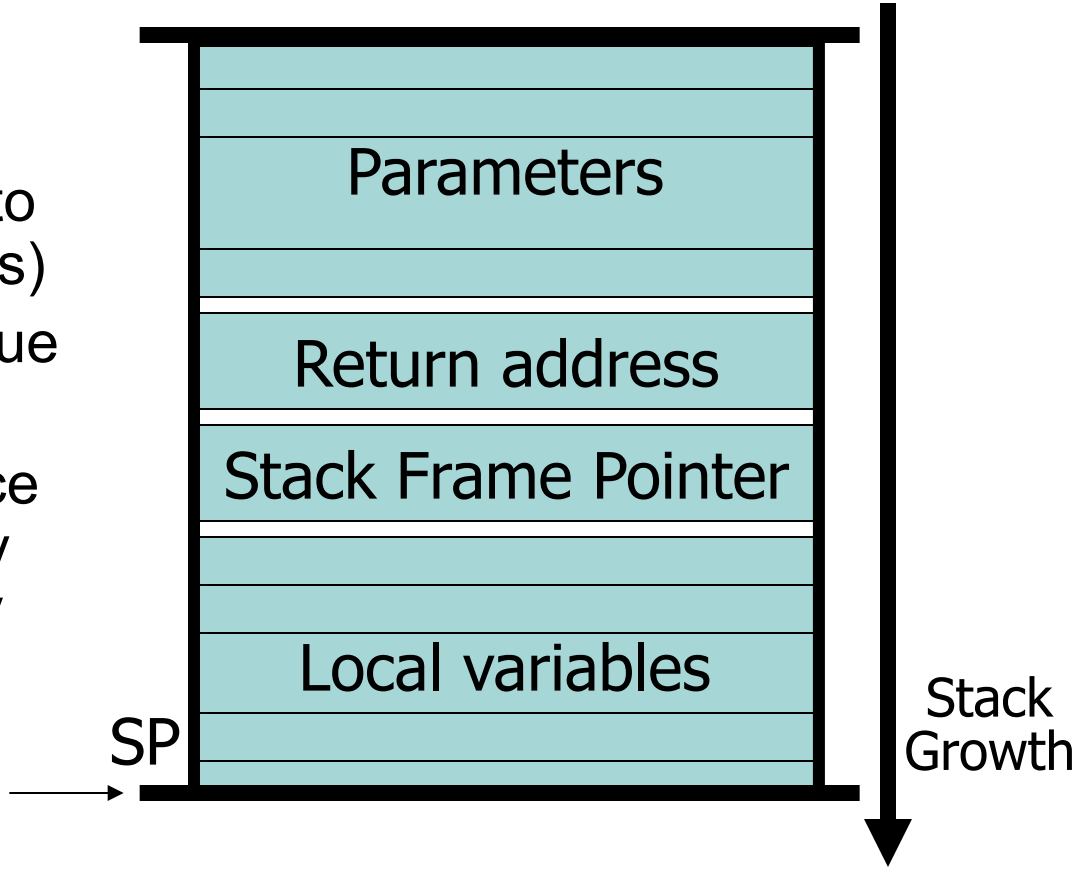


C Program execution

- PC (program counter or instruction pointer) points to next machine instruction to be executed
- Procedure call:
  - Prepare parameters
  - Save state (SP (stack pointer) and PC) and allocate on stack local variables
  - Jumps to the beginning of procedure being called
- Procedure return:
  - Recover state (SP and PC (this is return address)) from stack and adjust stack
  - Execution continues from return address

Stack frame

- Parameters for the procedure
- Save current PC onto stack (return address)
- Save current SP value onto stack
- Allocates stack space for local variables by decrementing SP by appropriate amount



Example: N!

- Observation: n! = n*(n-1)! and 1! = 1

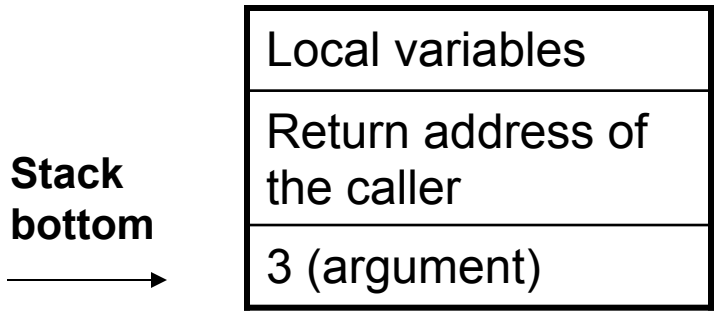
```
int fact(n) {
  if(n<=1)
     return 1;
  else
     return n*fact(n-1);
}
```

Zooming in …

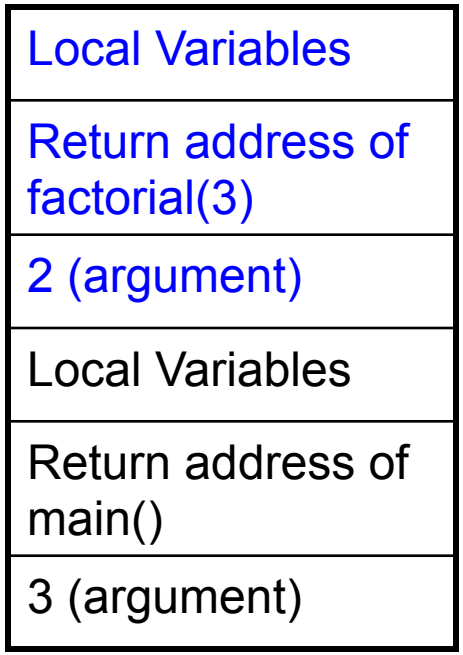
```
int factorial(int i) {
        if(i<=1) return 1;
        else return i*factorial(i-1);
}
```

factorial(3)=?

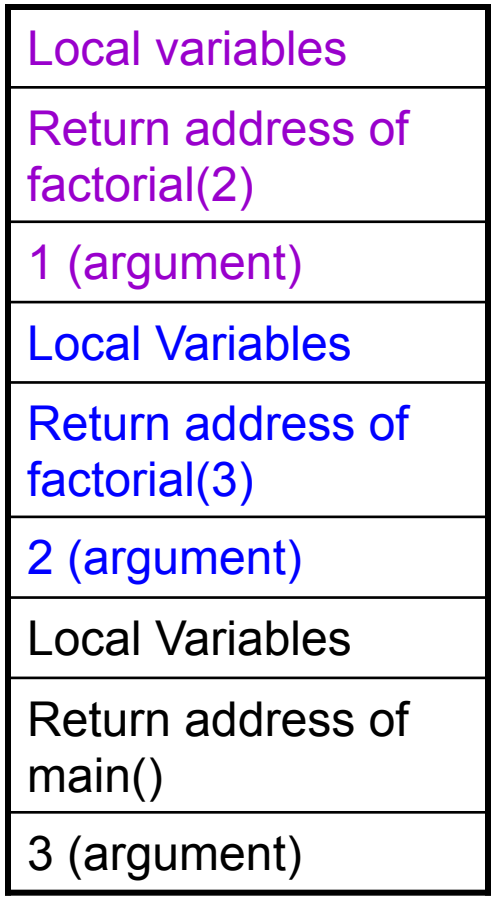
1. Call factorial(3)



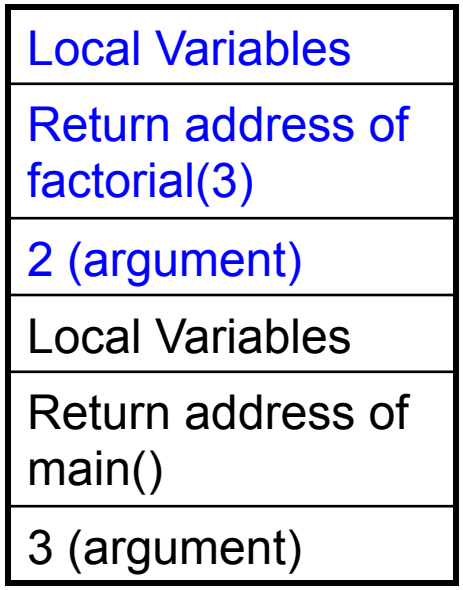
2. Call factorial(2) in factorial(3)



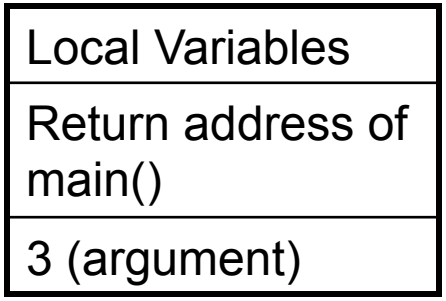
3. Call factorial(1) in factorial(2)



4. factorial(1) returns 1. The return value is stored in register. Control flow returns to factorial(2)



5. factorial(2) returns 2*1

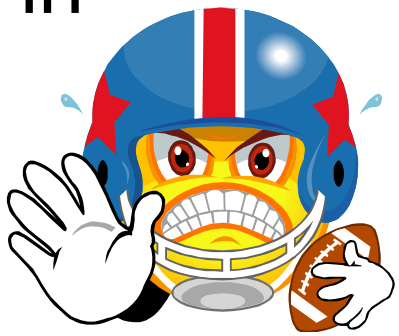


6. factorial(3) returns 3*2*1

Stack is empty

Exercises

- Write a recursive implementation to compute the GCD of two numbers
- What are the advantages of using recursion?
- Read the two recursive functions from your book and understand in each case what is the recursive relation and what is the stop condition



C Pre-processor

- Additional step before compilation
- Provides two operations
  - **`#include`**
  - **`#define`**

#include

- "" starts searching at source program location;
- <> follows implementation dependent
- rules; e.g., /usr/include and -I option in gcc specified at compilation time
- included file is usually a header (.h) file, but can also be a .c file or any other file

```
#include "filename"
#include <filename>
```

Example

- You have implemented a program package with a set of functions for other programmers to call
- You distribute the implementation of your code as a library
- You distribute the interface of your code as a header file for users of your code to #include, like <stdio.h> for the standard I/O library of the libc.a C library
- The .h file contains, say, prototypes of functions that the users will call, and external variables that the users can set to control your program's behavior.

Macro substitution

- scope is from occurrence of #define to corresponding #undef, another #define of the same name, or end of file
- simple textual substitution, NO LANGUAGE AWARENESS

```
#define name replacement-text
#undef name
```

Examples

- #define STEP 10
- #define forever for (;;)
- #define max(A, B) ((A) > (B) ? (A) : (B))

When #defines go wrong

- What's wrong with

```
#define square(x) x * x
```

- How about this

```
#define square(x) ((x) * (x))
```

Conditional pre-processing

- #ifdef
- #ifndef
- #else
- #elif
- #endif

Applications of #ifdef: portability

```
#ifdef SYSV
#define HDR "sysv.h"
#elif defined(BSD)
#define HDR "bsd.h"
#elif defined(MSDOS)
#define HDR "msdos.h"
#else
#define HDR "default.h"
#endif
#include HDR
```

Application of #ifndef: include files

- To include a include file only once

```
#idndef _MY_INCLUDE_FILE_
#define _MY_INCLUDE_FILE_
 header file

#endif /* _MY_INCLUDE_FILE_ */
```

Application of #ifdef: Print debug information

```
#ifdef DEBUG
#define DPRINTF(args) printf args
#else
#define DPRINTF(args)
#endif
```

- Specify how you want to macro to expand by specifying the DEBUG variable at compilation time in the Makefile
- gcc -D option

Readings for This Lecture

K&R Chapter 4