ECS 36A, May 30, 2024

Announcements

- Undoubtedly you feel overwhelmed with all we have covered
- ***This is normal; <u>do not panic!</u>***
  - My word, you're trying to learn an entire programming language in 10 weeks!
  - Best way to cement your understanding: write a lot of programs . . . *but*
  - I suspect some (all?) of you have other obligations, like other classes or jobs to support yourselves in school
  - C is very different than Python, and introduces new concepts like pointers

Announcements

- Final study guide and sample final are posted
    - Answers to the sample final go up this weekend, on Canvas
- Due to *lots* of requests, I am extending homework 3's due date to tomorrow at 11:59pm
- Homework 4 will be posted today; it has 2 problems
    - Submit the program for the second one to Canvas, *not* to Gradescope
    - Due date is June 6, last day of classes
    - *Late* due date is June 11, the day before the final exam
- Extra credit 3 will also go up
    - Same as for problem 2 and due dates, above

Sorting

- Function is:

```
void qsort(void *base, size_t nmemb, size_t size,
          int (*compar)(const void *, const void *));
```

- Here *compar* is function that takes 2 pointers to elements of the array *base*, with *nmemb* members of size *size*
- *compar* returns negative if first is less than second; 0 if the two are equal; and positive if the first is greater than the second
- You supply compar

Example *compar*

```
int cmp(const void *x, const void *y)
{
    int *px, *py;
    px = (int *) x;
    py = (int *) y;

    return(*x - *y);
}
```

Calling *qsort*

```
int arr[100]; /* rray of integers to be sorted */
int narr;     /* number of integers in arr */
/*  . . .  put random numbers into arr */
/* now sort them */
qsort(arr, narr, sizeof(int), (int (*)(const void *, const void *) ) cmp);
```

Oops . . .

Remember *qsort*? Here is its call:

```
qsort(base, nelts, sizeof(double),
      (int (*)(const void *, const void *)) cmp);
```

I used this for *cmp*:

```
int cmp(const void *x, const void *y){
      double *px, *py;
      px = (double *)x;
      py = (double *)y;
      return(*px - *py);
}
```

What is wrong with this?

Oops . . .

It’s the `*px - *py` — if it returns something less than 1.0, the function returns 0 (equal), even if there is a difference of (say) 0.5 or –0.5

```
int cmp(const void *x, const void *y){
        double *px, *py;
        px = (double *)x;
        py = (double *)y;
        if (*px > *py) return(1);
        else if (*px < *py) return(-1);
        return(0);
}
```

The lines in red replace the return in the earlier version

System Calls

- Direct interface between the applications and the operating system
- They vary among operating systems
  - We will deal with Linux system calls
- We'll look primarily at the file system calls

Opening a File: Basic Ideas

- Files represented by an integer
  - 0 refers to the standard output
  - 1 refers to the standard output
  - 2 refers to the standard error
- To get a file descriptor from a file stream:

  ```
  int fileno(FILE *fp)
  ```

  returns the file descriptor associated with file pointer *fp*
- You can now mix system and stdio calls *provided you use the file descriptor in the same way you use the file pointer*
  - For example, if the file *fp* points to is open for reading, using the file descriptor to write to it will give you an error

Opening a File: Basic Ideas

- To get a file pointer from a file descriptor:

  ```
  FILE *fdopen(int fd, char *mode)
  ```

  creates a file pointer (and corresponding structure) to file descriptor *fd*, which was opened as mode indicates
- You have to set *mode* the same way as you opened it
- The system maintains a rw-pointer at the spot (the *file offset*) where the next read or write will take place
  - Unless there is an *fseek* or *fsetpos*, which moves the rw-pointer

Opening a File: Details

```
int open(const char *name, int flags)
```

- Opens the file name in the way flags indicate:
  - **O_RDONLY**: open file for reading
  - **O_WRONLY**: open file for writing
  - **O_RDWR**: open file for reading and writing
- Other flags augment these
  - **O_APPEND**: with **O_WRONLY** and **O_RDWR**, append rather than overwrite
  - **O_CREAT**: create the file if it does not exist
  - **O_EXCL**: with **O_CREAT**, fail if the file exists

Detour: File Permissions

- File protection, or *mode*, is 12 bits long; for us, the first 3 bits are irrelevant
- The other 9 bits are arranged in groups of 3:

r w x r w x r w x

- First 3 refer to *owner* (also called *user*)
- Second 3 refer to *group*
- Last 3 refer to everyone else (sometimes called *other* or *world*)
- Each r (read), w (write), x (execute) is a bit; 1 means allowed, 0 means not allowed

Detour: *umask*

- *Umask* is a shell variable designed to *mask* file creation permissions
- Each bit of *umask* turns off the corresponding bit in the permissions when a file is created
  - It's a safety mechanism so the file creator doesn't accidentally give others access they should not have
- Example: file is created with permission 666 (anyone can read or write it)
  - Not a good idea!
- Set *umask* to 022 (group and other write bits set here)
- Result: the file is created with permission 644 (anyone can read it, but only the owner can write to it)

Creating a File

- When creating a file, a third argument specifies permissions:

```
int open(const char *name, int flags, mode_t mode)
```

- The file permissions are set to

mode&~umask

- Example: if *umask* is 077, and mode is 0644 (owner can write, everyone can read), the file is created with protection mode

```
0644&~077 = 0644&0700 = 0600
```

so only the owner can read or write the file

Reading

```
ssize_t read(int fd, void *buf, size_t count)
```

- Read *count* bytes from file descriptor *fd* and save them in the area *buf* points to
  - You have to allocate *buf* or create an array or variable to give the address of
  - On success, returns the number of bytes read; this is never more than *count* but may be less
  - If it returns 0, you've reached the end of file
  - If it returns –1, an error occurred, and *errno* is set to indicate the error

Writing

```
ssize_t write(int fd, void *buf, size_t count)
```

- Writes *count* bytes from the address *buf* contains to file descriptor *fd*
  - *buf* is the address of what you want written
  - *count* is the number of bytes to write; it does *not* stop at the NUL ('\0') byte
  - On success, returns the number of bytes written; this is never more than *count* but may be less
  - If it returns 0, nothing was written
  - If it returns –1, an error occurred, and *errno* is set to indicate the error

Seeking

```
off_t lseek(int fd, off_t offset, int whence)
```

- Move the rw-pointer associated with the file descriptor *fd* to offset according to whence
  - whence is **SEEK_SET** (beginning), **SEEK_CUR** (current position), **SEEK_END** (end of file)
  - It returns new rw-pointer offset in bytes from beginning of file
- Error handling
  - If it returns –1, an error may have occurred, and if so *errno* is set to indicate the error
  - Note: off_t is unsigned long int, so that could be a valid value of a very big file

Detecting Error in *lseek*

- If you are moving rw-pointer to a given position *x*, this works:

```
    if (lseek(fd, x, SEEK_SET) != x) . . .
```

- Otherwise, do this:

```
errno = 0;          /* clear any existing error code */
if (lseek(fd, offset, whence) == -1 && errno != 0){
   /* . . . Handle error . . . */
```

Get File Status

```
    int stat(const char *name, struct stat *buf)
    int fstat(int fd, struct stat *buf)
```

- Get the status of the file *name* or the file associated with the descriptor *fd*

Representation of Status

```
struct stat {
        dev_t st_dev;               /* ID of device containing file */
        ino_t st_ino;               /* inode number */
        mode_t st_mode;             /* protection */
        nlink_t st_nlink;           /* number of hard links */
        uid_t st_uid;               /* user ID of owner */
        gid_t st_gid;               /* group ID of owner */
        dev_t st_rdev;              /* device ID (if special file) */
        off_t st_size;              /* total size, in bytes */
        blksize_t st_blksize;       /* blocksize for file system I/O */
        blkcnt_t st_blocks;         /* number of 512B blocks allocated */
        time_t st_atime;            /* time of last access */
        time_t st_mtime;            /* time of last modification */
        time_t st_ctime;            /* time of last status change */
};
```

Closing a File

```
int close(int fd)
```

- Disassociates the file associated with *fd*
- *fd* no longer is bound to any file and can be reused
- On success, it returns 0
- On failure, it returns –1 and puts the error code in *errno*

Deleting a File

```
int unlink(const char *name)
```

- Deletes file *name* from the file system
- If there are other links to it, the file's storage is still being used
  - If the file is open, that's a link
- On success, it returns 0
- On failure, it returns –1 and puts the error code in *errno*

Review of Recursion

- Because it's tricky the first time you see it (or the second, or the ...)
- The following slides show the recursive computation of Fibonacci numbers
- A ***huge*** thank you to Karen Liu, a former TA for ECS 36A — she did these slides!

Environment Variables

- Used to control program execution over multiple executions
- Also used as variables when programming the shell
- Common ones:
    - **PATH** defines where to look for programs (*search path*)
    - **HOME** home directory
    - **SHELL** current shell
    - **USER** name of user

How to Get Them

- If you do not know what the variables are, or want the same program to reference more than 1 during execution, do this:

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

- *envp* is just like *argv*, but it contains a list of environment variables and their values, like this:

```
VARIABLE=VALUE
```

- Example:

```
SHELL=/bin/bash
```

How to Get Them

- If you know the variable you are interested in, do this:

```
char *getenv(char *variable)
```

- This gets you a string of the form:

```
VARIABLE=VALUE
```

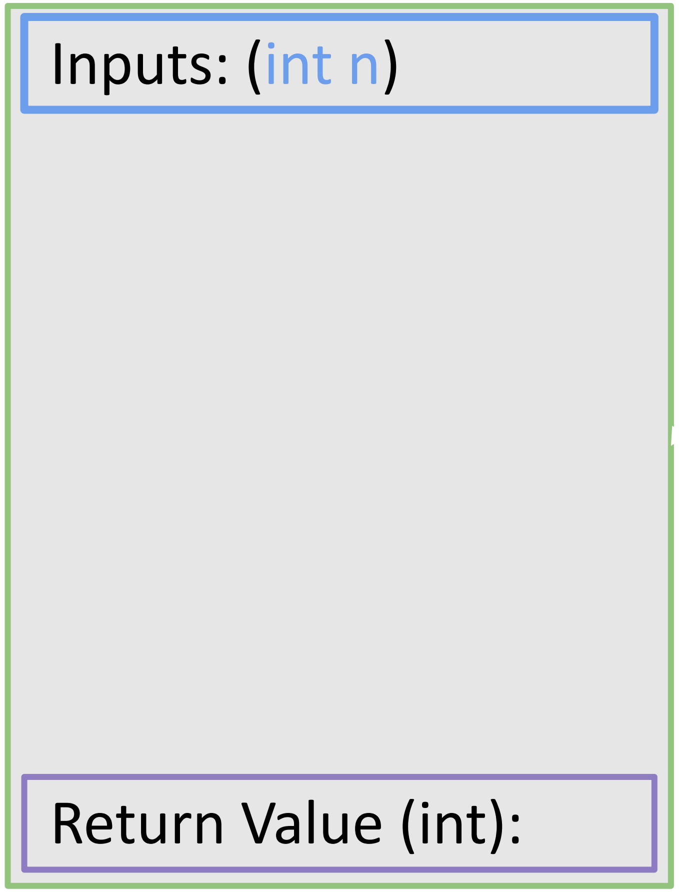
  and then look for the first =; what follows is the value

- If the variable does not exist, *getenv* returns **NULL**

Review of Recursion

- Recursion is tricky the first time you see it
  - And the second
  - And the third
  - . . .
- You *will* get used to it
- Here's a review using Fibonacci numbers
- Slides done by Karen Tu, a TA for an earlier ECS 36A class (thanks, Karen!)

Function Demonstration Notations



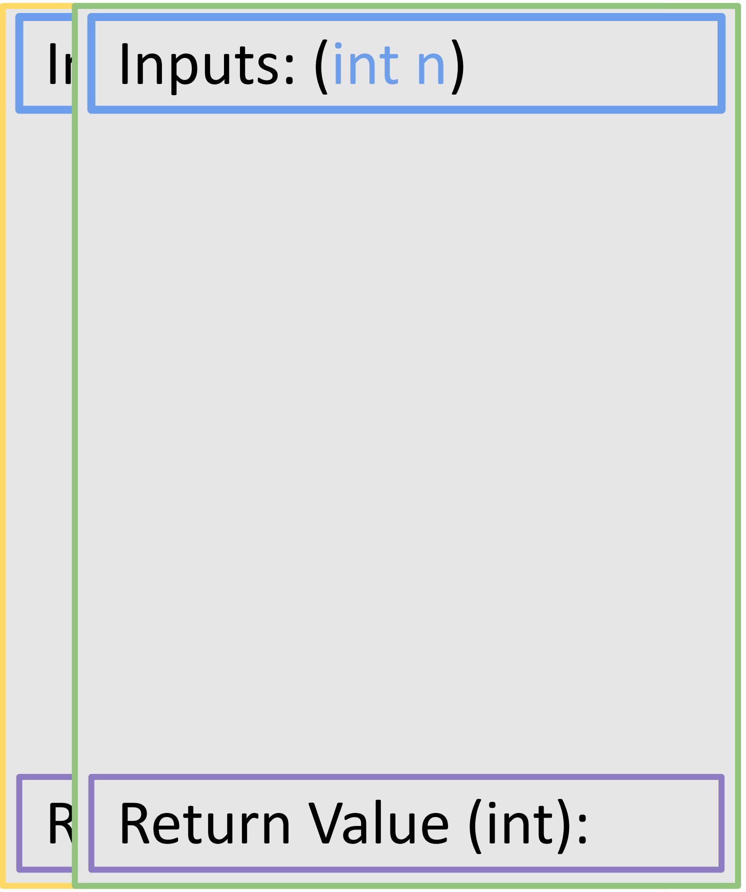
Inputs: (int n)

Return Value (int):

An area in memory allocated for a function call.

int int n

Function Demonstration Notations



Ir

Inputs: (int n)

R

Return Value (int):

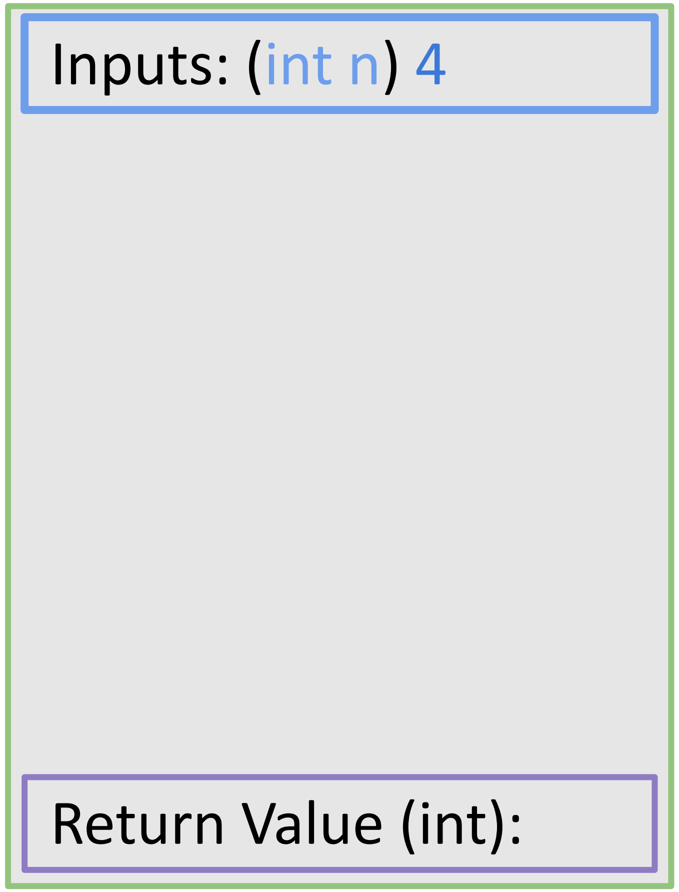
Yellow outline means the function space is allocated but the program is not running in that allocated space.

Green

Example Function Call: Recursion

```
#include <stdio.h>
#define MAXLEN 10

int ret;
```

```
int main () {
  ret = fib(4);
  printf(“%d\n”, ret);
  return 0;
}
```

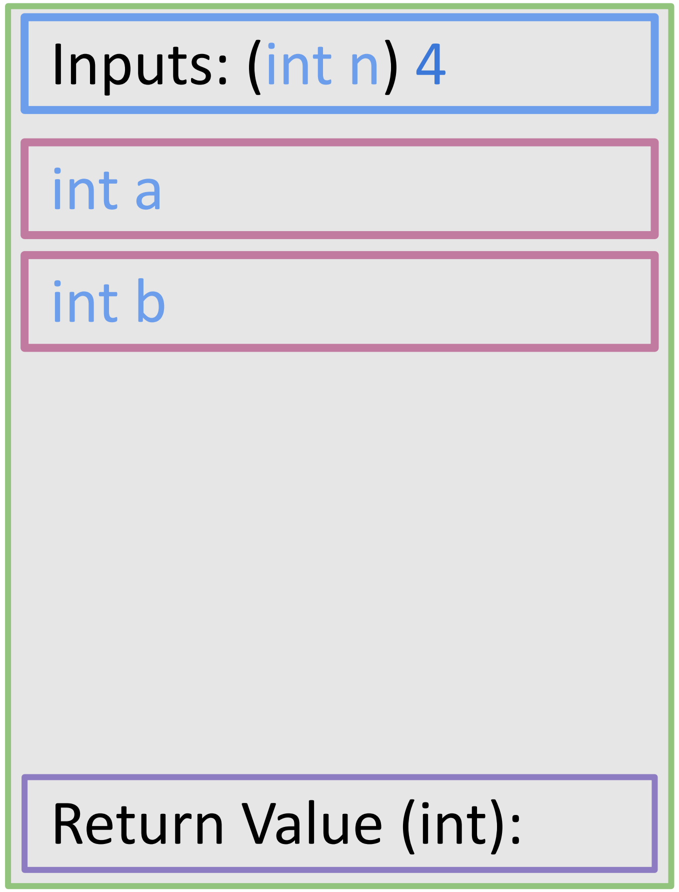


Inputs: (int n) 4

Return Value (int):

Example Function Call: Recursion

```
#include <stdio.h>
#define MAXLEN 10

int ret;

/* function declaration */
int fib(int n) {

  if (n <= 1) {
    return n;
  }
  a = fib(n-1)
  b = fib(n-2)
  return a + b;
}

int main () {
  ret = fib(4);
  printf(“%d\n”, ret);
  return 0;
}
```



Inputs: (int n) 4

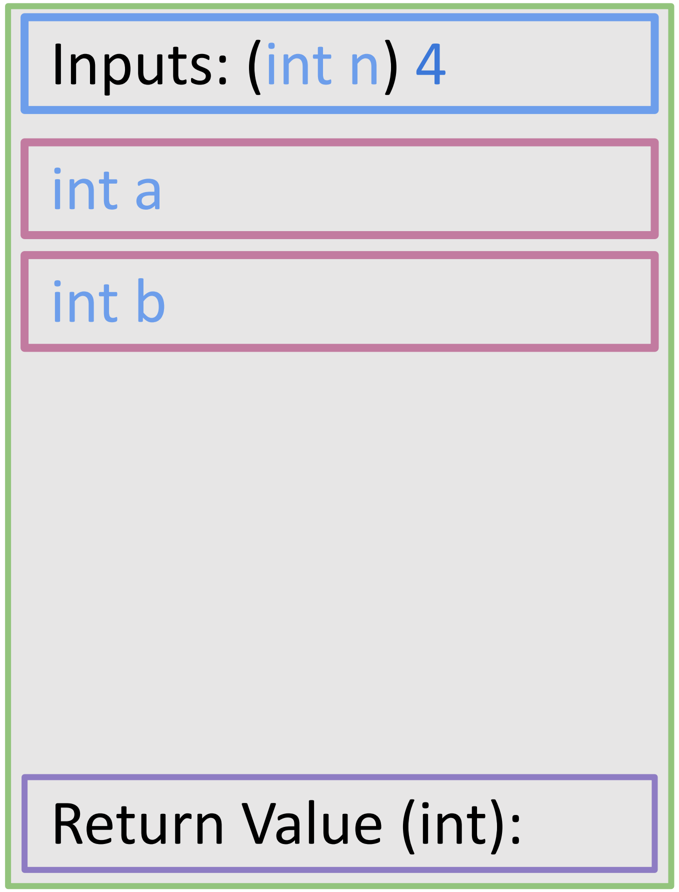
int a

int b

Return Value (int):

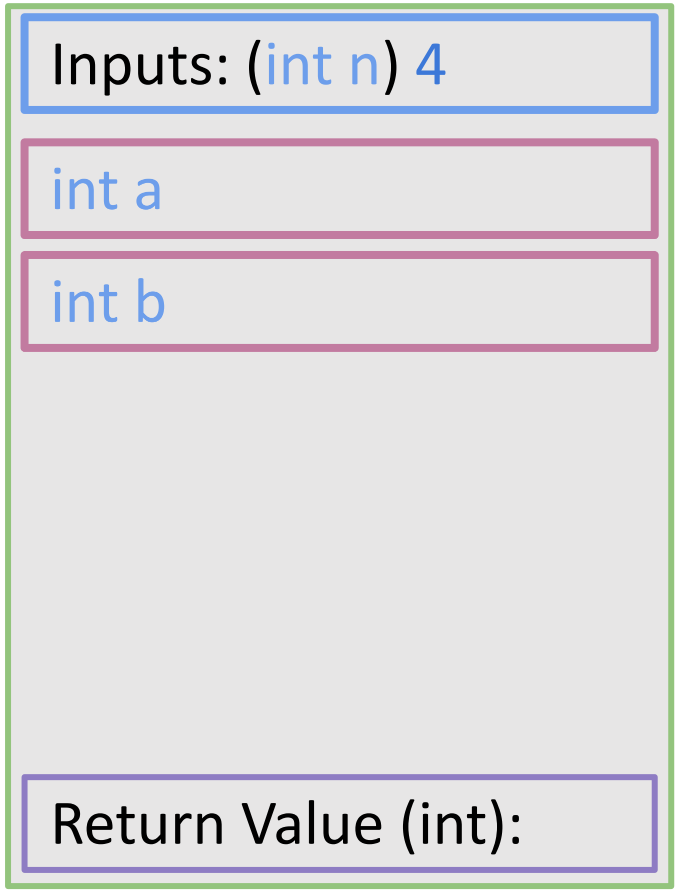
Example Function Call: Recursion

```
#include <stdio.h>
#define MAXLEN 10

int ret;

/* function declaration */
int fib(int n) {
  int a, b;
                        4
    return n;

  a = fib(n-1)
  b = fib(n-2)
  return a + b;


int main () {
  ret = fib(4);
  printf(“%d\n”, ret);
  return 0;
}
```



Example Function Call: Recursion

```
#include <stdio.h>
#define MAXLEN 10

int ret;

/* function declaration */
int fib(int n) {
  int a, b;
  if (n <= 1) {  /* n = 4 > 1 */
    return n;
  }

  b = fib(n-2)
  return a + b;
}

int main () {
  ret = fib(4);
  printf(“%d\n”, ret);
  return 0;
}
```



Inputs: (int n) 4

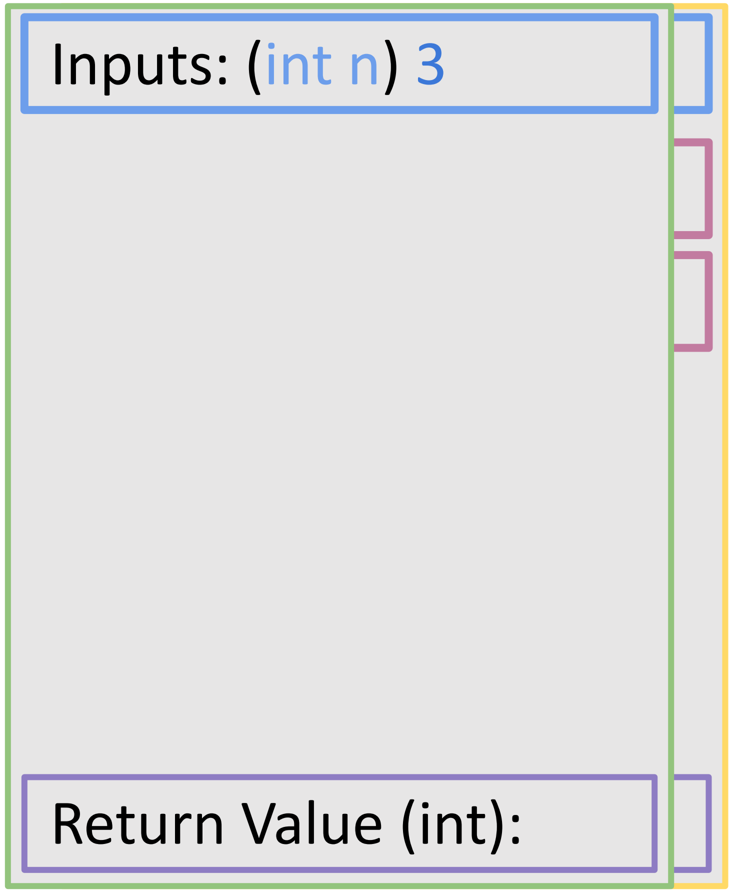
int a

int b

Return Value (int):

Example Function Call: Recursion

```
#include <stdio.h>
#define MAXLEN 10

int ret;

/* function declaration */
int fib(int n) {
  int a, b;
  if (n <= 1) {  /* n = 4 > 1 */
    return n;
  }
                           3
  b = fib(n-2)
  return a + b;
}

int main () {
  ret = fib(4);
  printf("%d\n", ret);
  return 0;
}
```

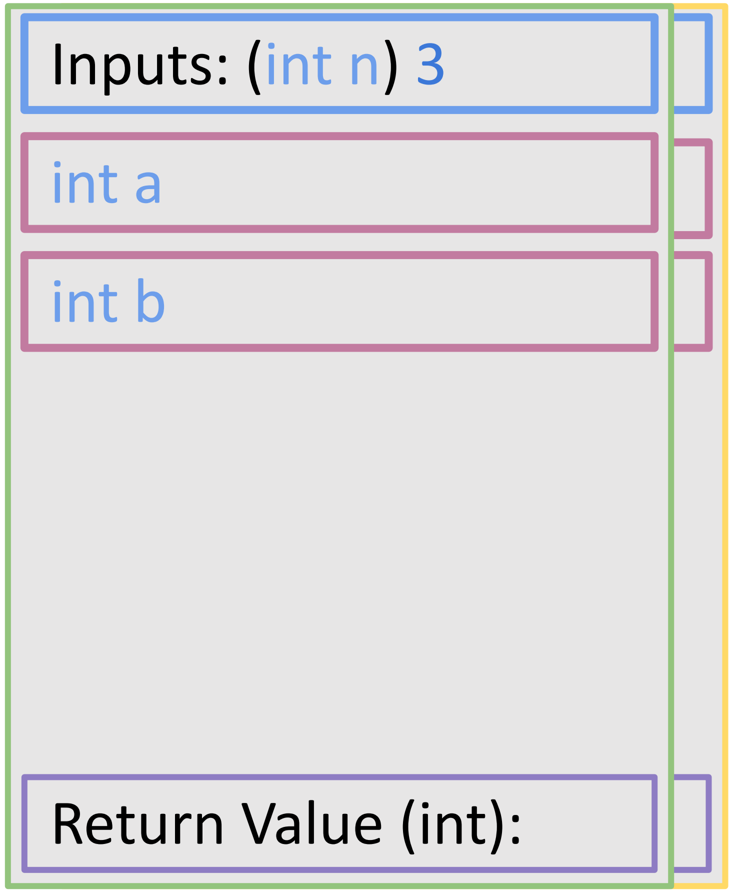


Inputs: (int n) 3

Return Value (int):

Example Function Call: Recursion

```
#include <stdio.h>
#define MAXLEN 10

int ret;

/* function declaration */
int fib(int n) {

  if (n <= 1) {
    return n;
  }
  a = fib(n-1)
  b = fib(n-2)
  return a + b;


int main () {
  ret = fib(4);
  printf("%d\n", ret);
  return 0;
}
```



Inputs: (int n) 3

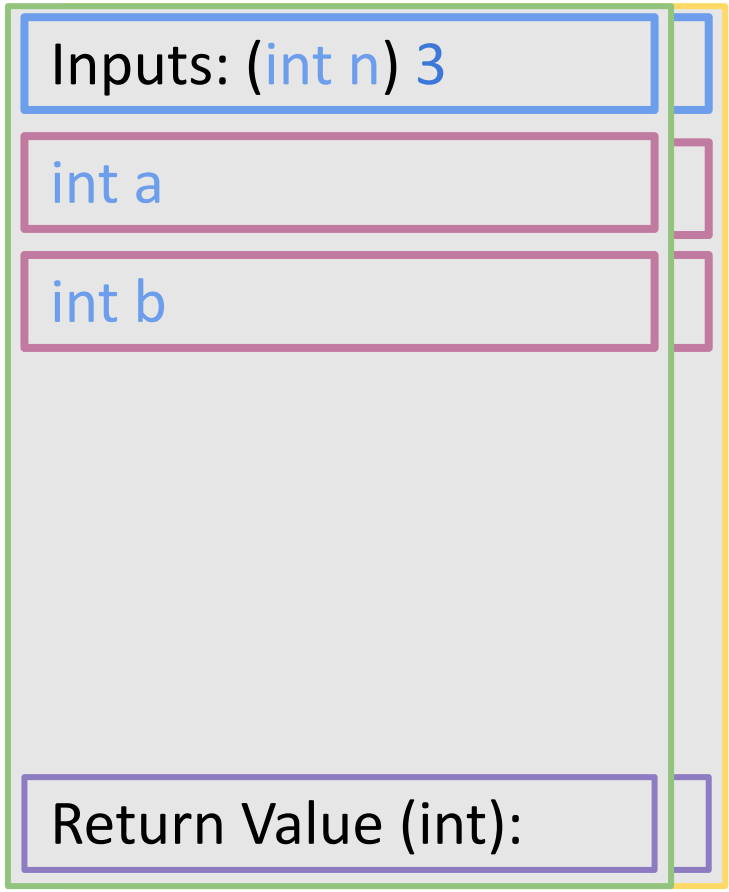
int a

int b

Return Value (int):

Example Function Call: Recursion

```
#include <stdio.h>
#define MAXLEN 10

int ret;

/* function declaration */
int fib(int n) {
  int a, b;
                        3
    return n;
  }
  a = fib(n-1)
  b = fib(n-2)
  return a + b;


int main () {
  ret = fib(4);
  printf(“%d\n”, ret);
  return 0;
}
```



Inputs: (int n) 3

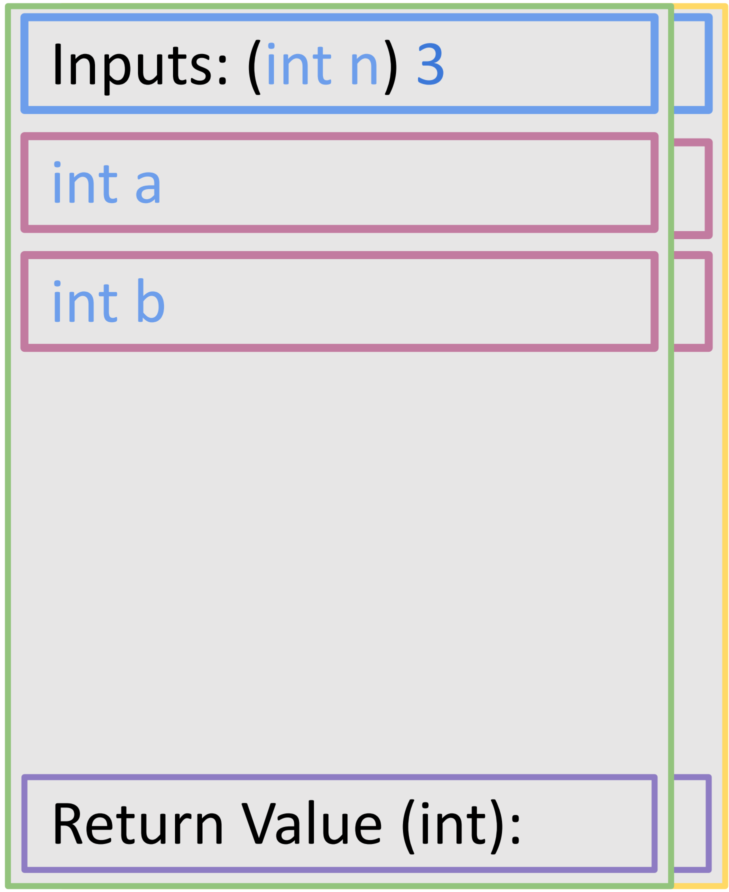
int a

int b

Return Value (int):

Example Function Call: Recursion

```
#include <stdio.h>
#define MAXLEN 10

int ret;

/* function declaration */
int fib(int n) {
  int a, b;
  if (n <= 1) {  /* n = 3 > 1 */
    return n;
  }

  b = fib(n-2)
  return a + b;
}

int main () {
  ret = fib(4);
  printf(“%d\n”, ret);
  return 0;
}
```



Inputs: (int n) 3

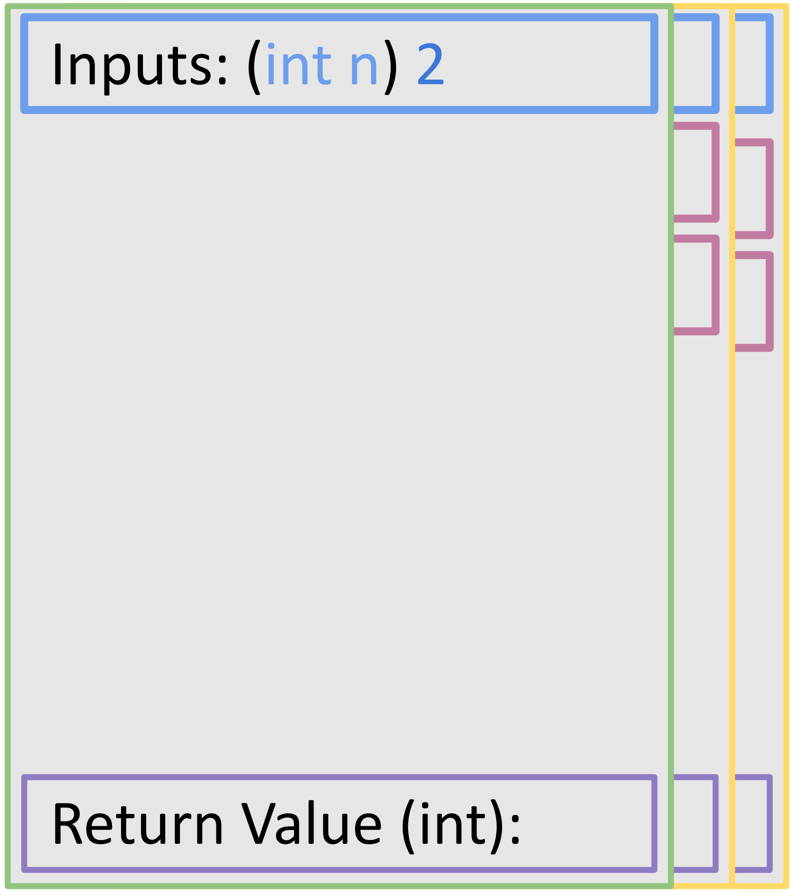
int a

int b

Return Value (int):

Example Function Call: Recursion

```
#include <stdio.h>
#define MAXLEN 10

int ret;

/* function declaration */
int fib(int n) {
  int a, b;
  if (n <= 1) {  /* n = 3 > 1 */
    return n;
  }
                                     2

  b = fib(n-2)
  return a + b;
}

int main () {
  ret = fib(4);
  printf("%d\n", ret);
  return 0;
}
```

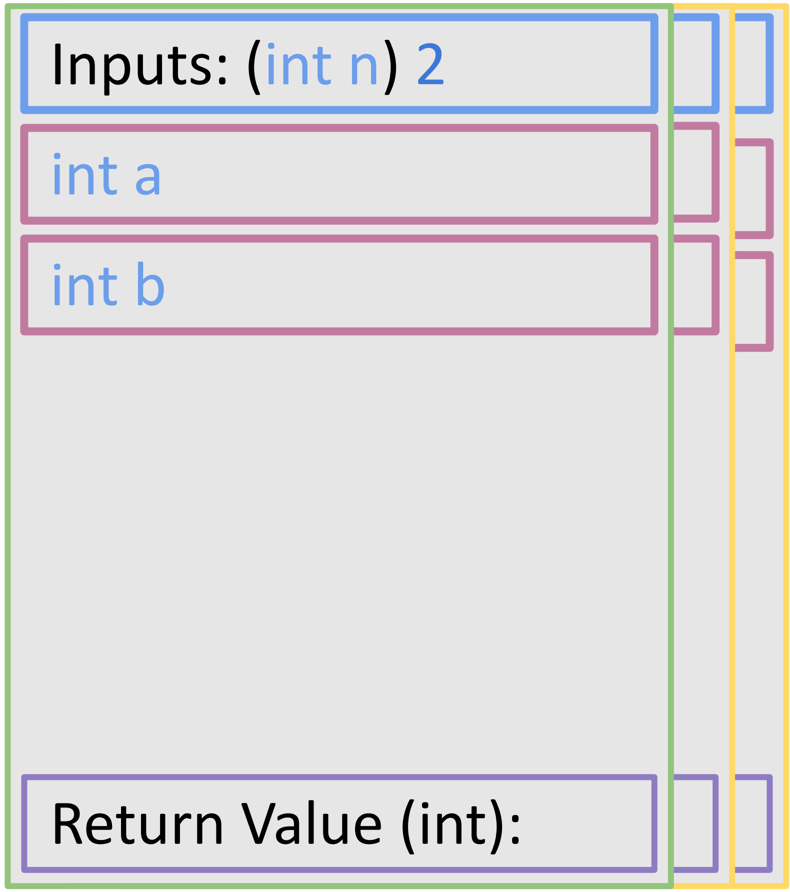


Inputs: (int n) 2

Return Value (int):

Example Function Call: Recursion

```
#include <stdio.h>
#define MAXLEN 10

int ret;

/* function declaration */
int fib(int n) {

  if (n <= 1) {
    return n;
  }
  a = fib(n-1)
  b = fib(n-2)
  return a + b;
}

int main () {
  ret = fib(4);
  printf(“%d\n”, ret);
  return 0;
}
```



Inputs: (int n) 2

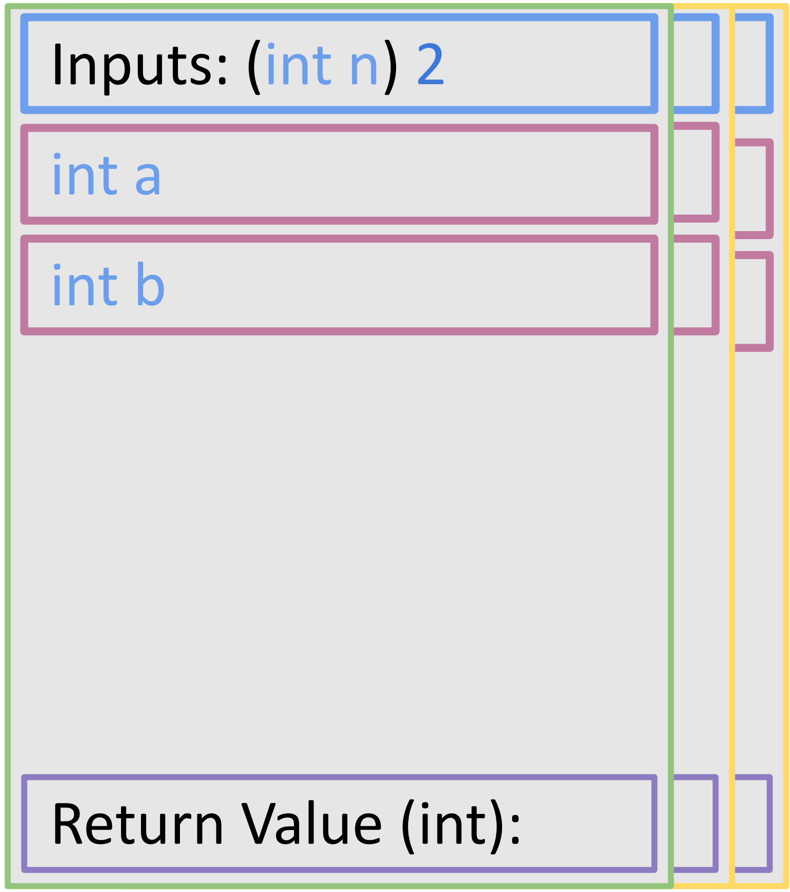
int a

int b

Return Value (int):

Example Function Call: Recursion

```
#include <stdio.h>
#define MAXLEN 10

int ret;

/* function declaration */
int fib(int n) {
  int a, b;
                       2
    return n;
  }
  a = fib(n-1)
  b = fib(n-2)
  return a + b;
}

int main () {
  ret = fib(4);
  printf(“%d\n”, ret);
  return 0;
}
```



Inputs: (int n) 2

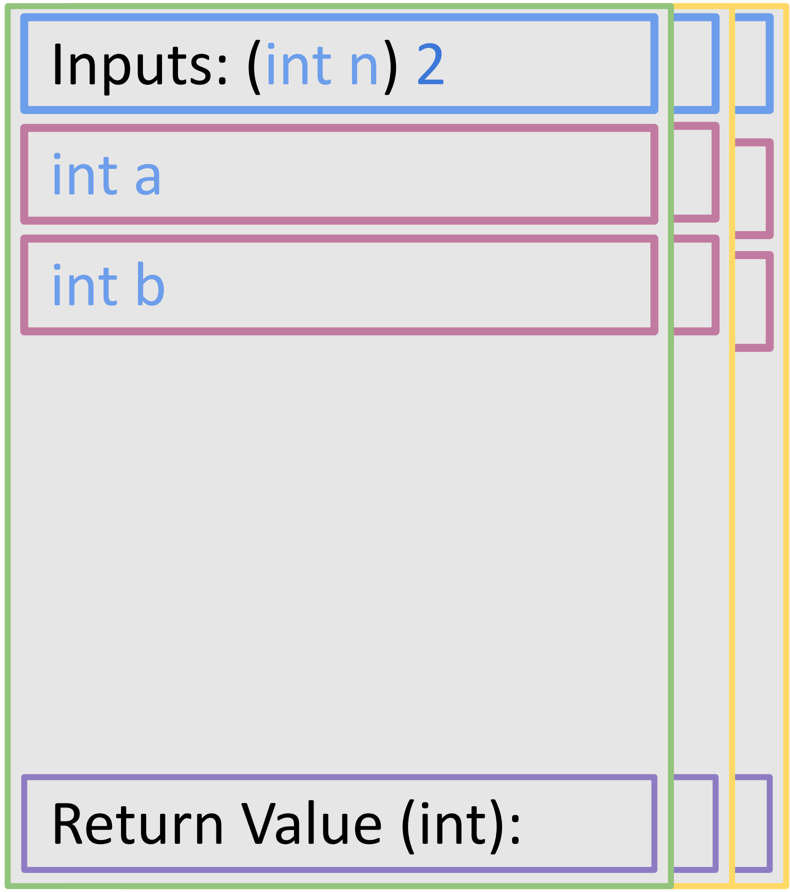
int a

int b

Return Value (int):

Example Function Call: Recursion

```
#include <stdio.h>
#define MAXLEN 10

int ret;

/* function declaration */
int fib(int n) {
  int a, b;
  if (n <= 1) { /* n = 2 > 1 */
    return n;
  }

  b = fib(n-2)
  return a + b;
}

int main () {
  ret = fib(4);
  printf(“%d\n”, ret);
  return 0;
}
```



Inputs: (int n) 2

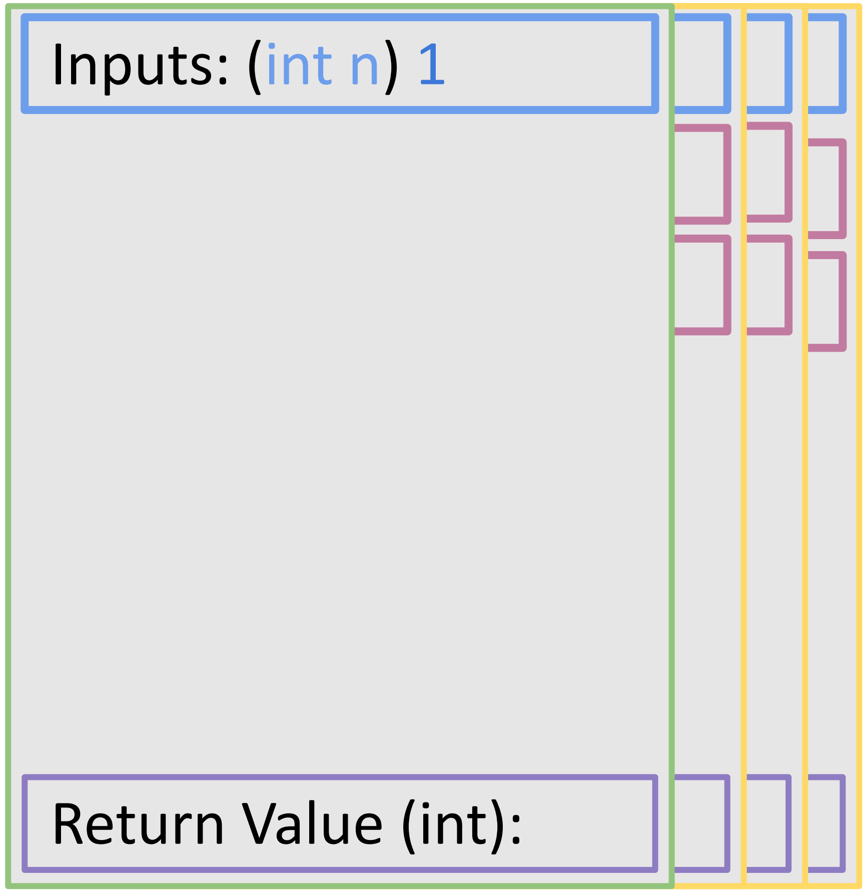
int a

int b

Return Value (int):

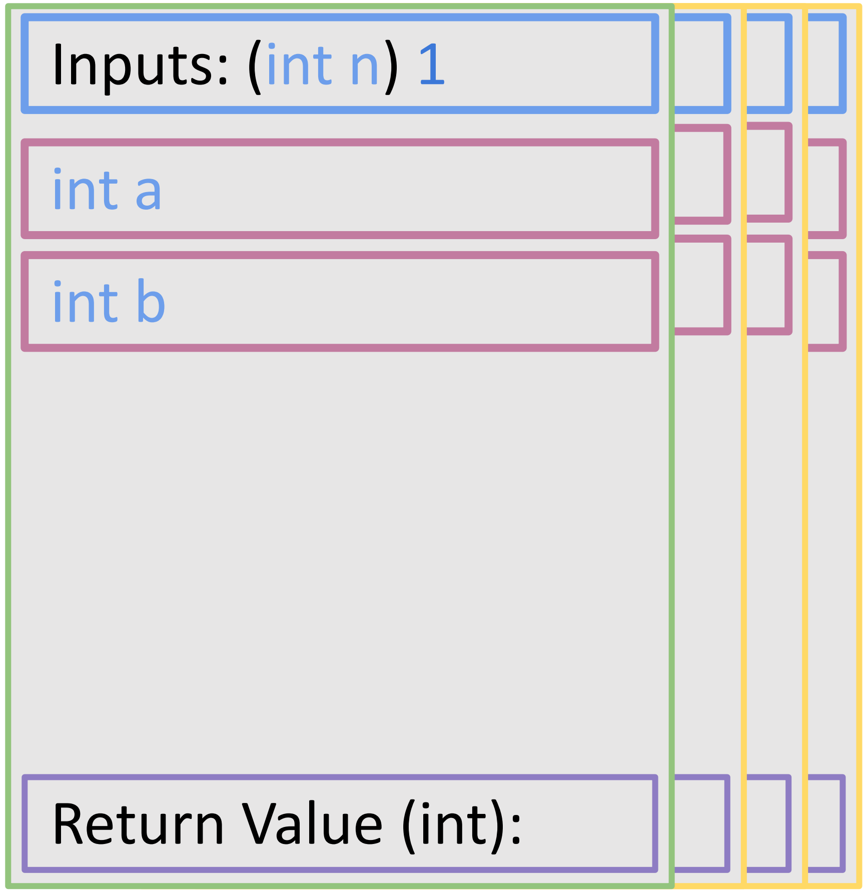
Example Function Call: Recursion

```
#include <stdio.h>
#define MAXLEN 10

int ret;

/* function declaration */
int fib(int n) {
  int a, b;
  if (n <= 1) { /* n = 2 > 1 */
    return n;
  }
                          1

  b = fib(n-2)
  return a + b;
}

int main () {
  ret = fib(4);
  printf(“%d\n”, ret);
  return 0;
}
```



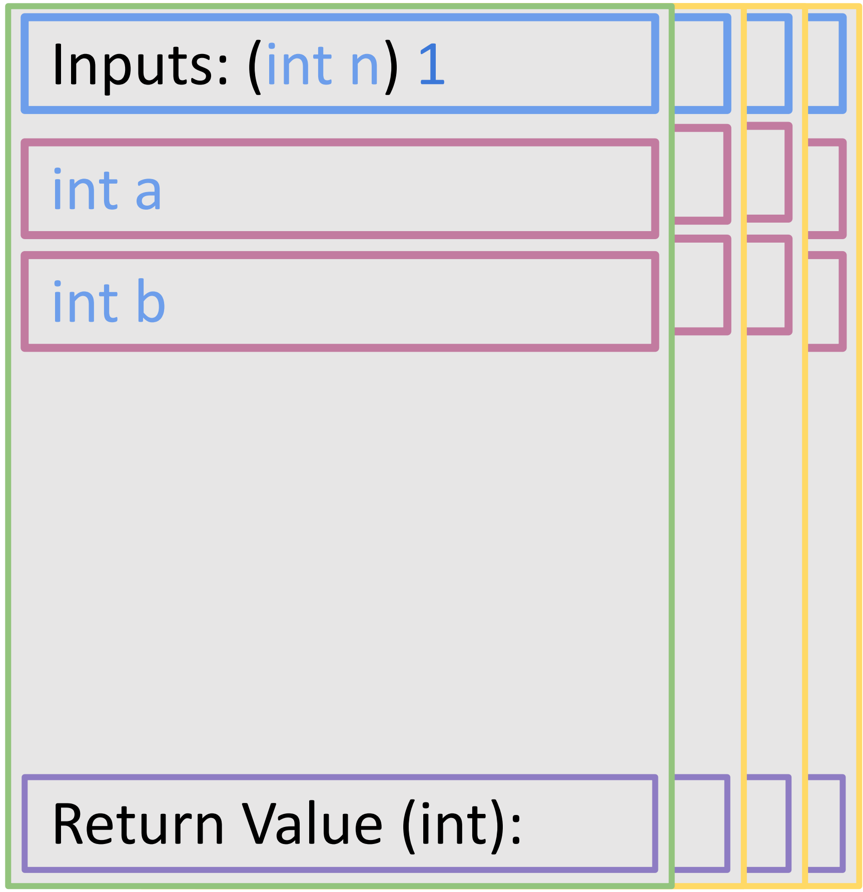
Example Function Call: Recursion

```
#include <stdio.h>
#define MAXLEN 10

int ret;

/* function declaration */
int fib(int n) {

  if (n <= 1) {
    return n;
  }
  a = fib(n-1)
  b = fib(n-2)
  return a + b;
}

int main () {
  ret = fib(4);
  printf(“%d\n”, ret);
  return 0;
}
```



Example Function Call: Recursion

```
#include <stdio.h>
#define MAXLEN 10

int ret;

/* function declaration */
int fib(int n) {
  int a, b;
                /* n = 1 <= 1*/
    return n;
  }
  a = fib(n-1)
  b = fib(n-2)
  return a + b;
}

int main () {
  ret = fib(4);
  printf(“%d\n”, ret);
  return 0;
}
```


Inputs: (int n) 1

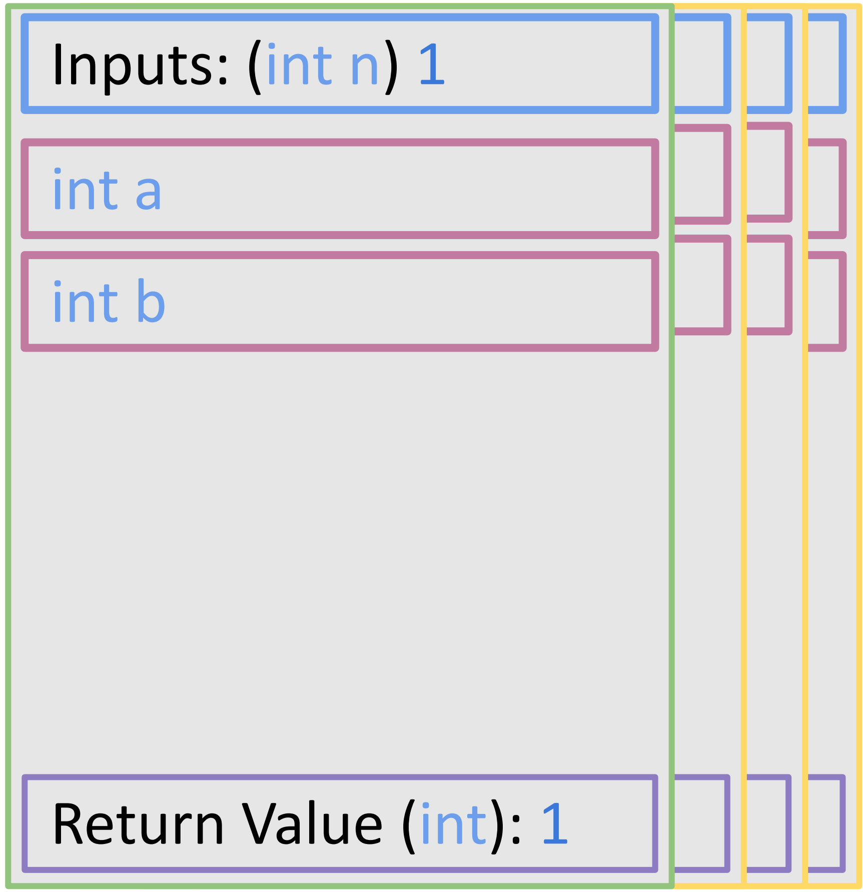
int a

int b

Return Value (int):

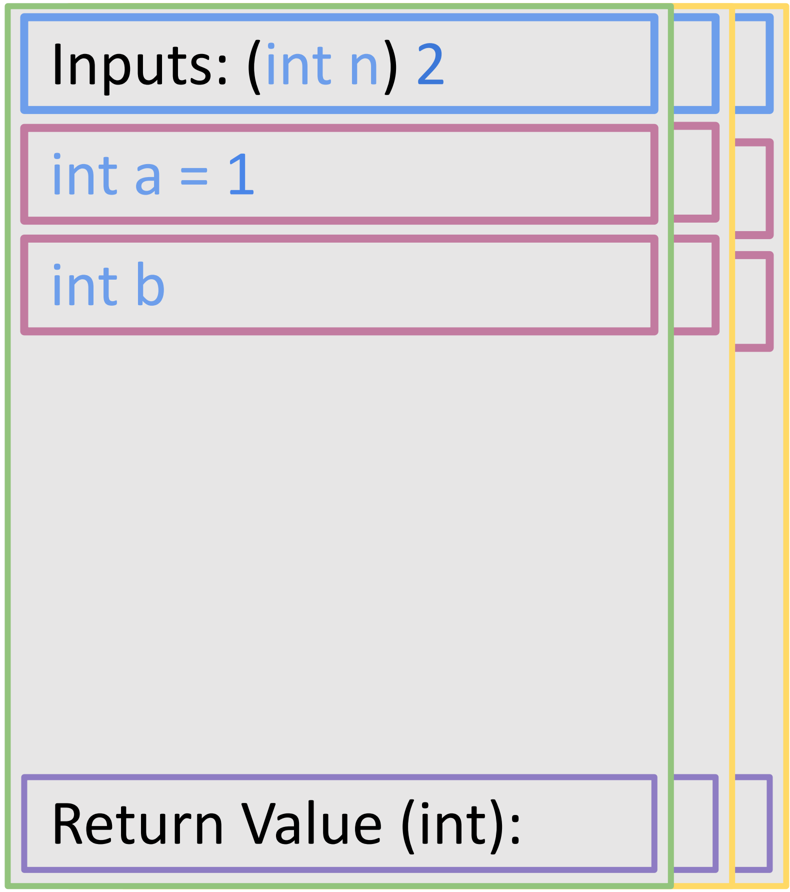
Example Function Call: Recursion

```
#include <stdio.h>
#define MAXLEN 10

int ret;

/* function declaration */
int fib(int n) {
  int a, b;
                /* n = 1 <= 1*/



  a = fib(n-1)
  b = fib(n-2)
  return a + b;
}

int main () {
  ret = fib(4);
  printf("%d\n", ret);
  return 0;
}
```



Example Function Call: Recursion

```
#include <stdio.h>
#define MAXLEN 10

int ret;

/* function declaration */
int fib(int n) {
  int a, b;
  if (n <= 1) { /* n = 2 > 1 */
    return n;
  }

  b = fib(n-2)
  return a + b;
}

int main () {
  ret = fib(4);
  printf(“%d\n”, ret);
  return 0;
}
```



Inputs: (int n) 2

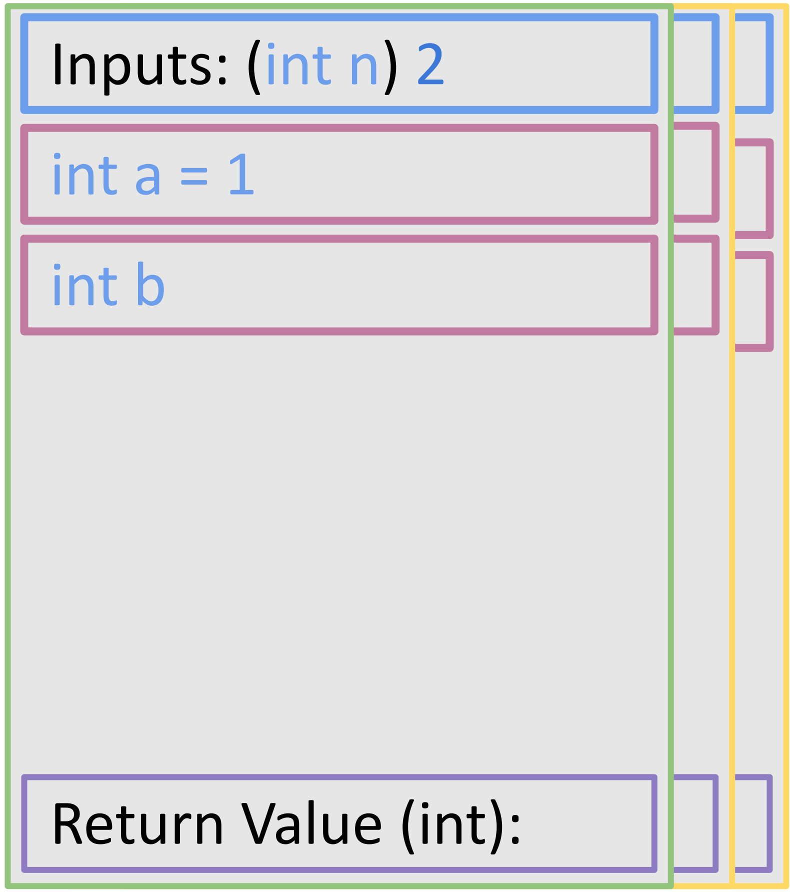
int a = 1

int b

Return Value (int):

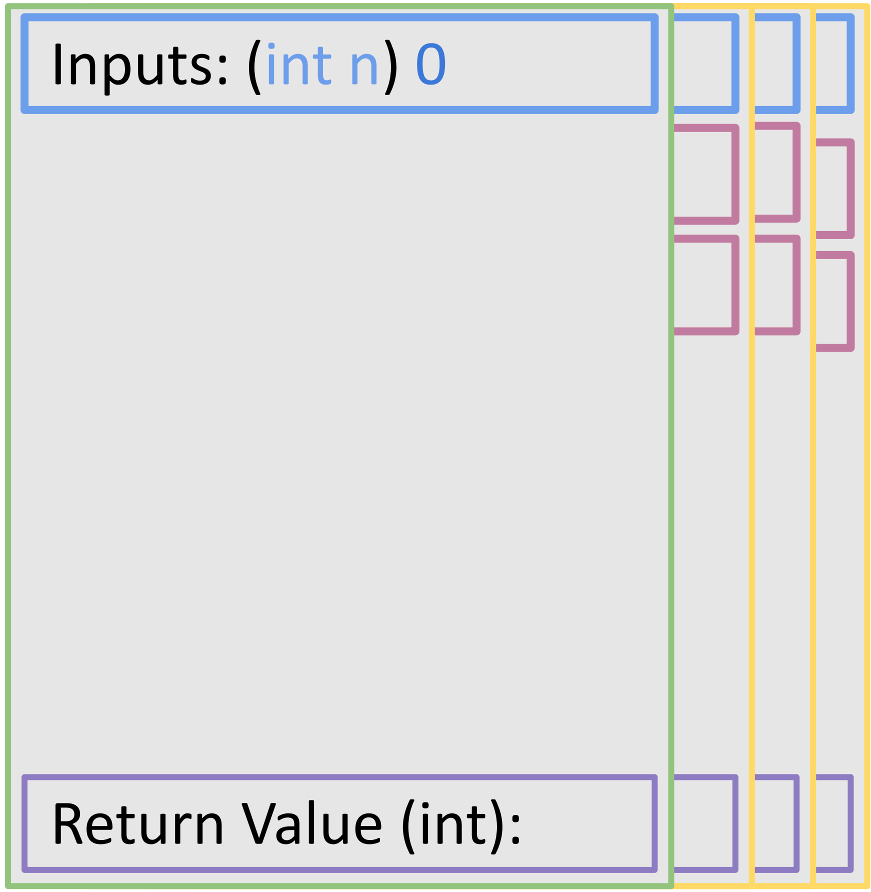
Example Function Call: Recursion

```
#include <stdio.h>
#define MAXLEN 10

int ret;

/* function declaration */
int fib(int n) {
  int a, b;
  if (n <= 1) { /* n = 2 > 1 */
    return n;
  }
  a = fib(n-1) /* n - 1 = 1 */

  return a + b;
}

int main () {
  ret = fib(4);
  printf(“%d\n”, ret);
  return 0;
}
```



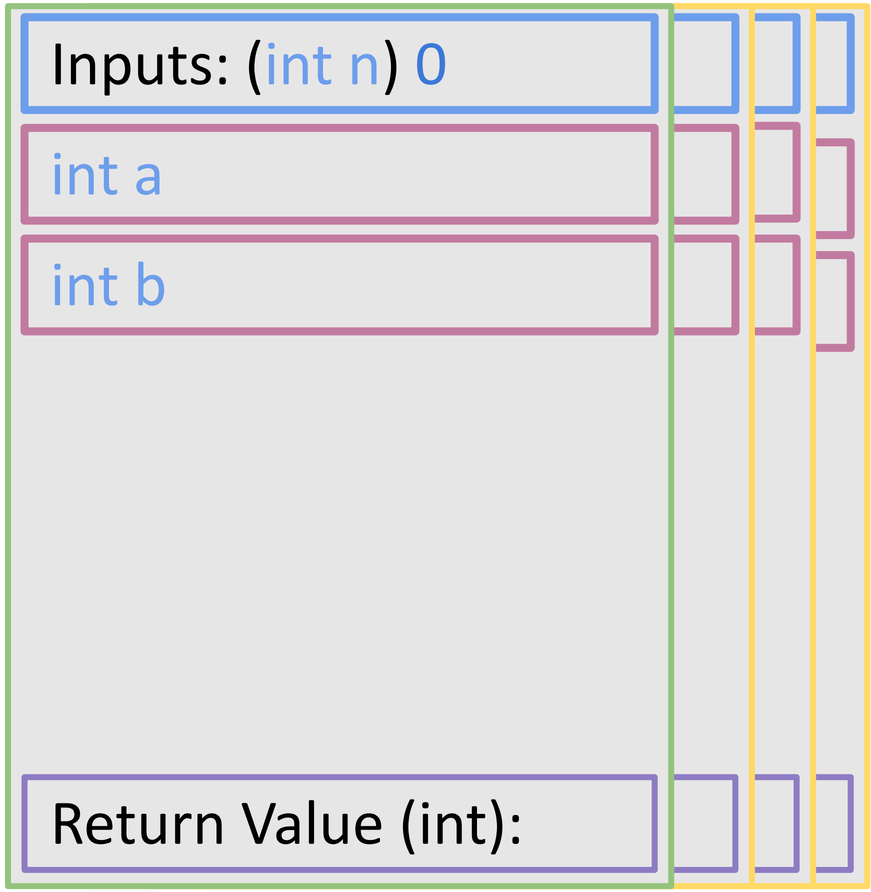
Example Function Call: Recursion

```
#include <stdio.h>
#define MAXLEN 10

int ret;

/* function declaration */
int fib(int n) {
  int a, b;
  if (n <= 1) { /* n = 2 > 1 */
    return n;
  }
  a = fib(n-1) /* n - 1 = 1 */
                          0

  return a + b;
}

int main () {
  ret = fib(4);
  printf("%d\n", ret);
  return 0;
}
```



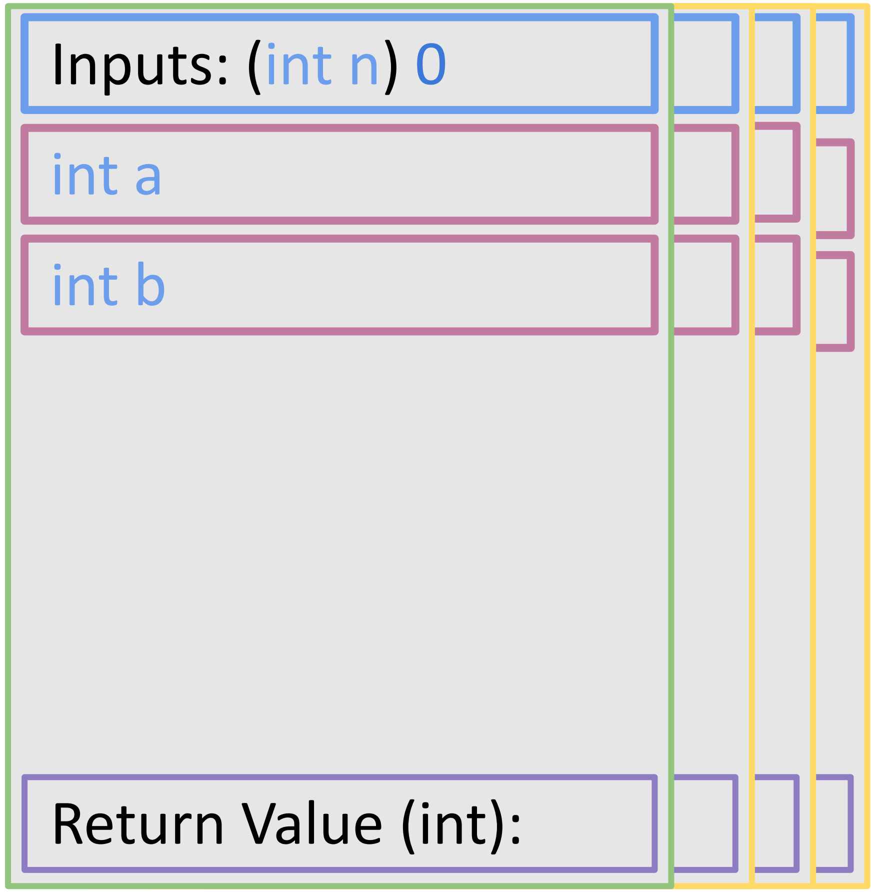
Example Function Call: Recursion

```
#include <stdio.h>
#define MAXLEN 10

int ret;

/* function declaration */
int fib(int n) {

  if (n <= 1) {
    return n;
  }
  a = fib(n-1)
  b = fib(n-2)
  return a + b;
}

int main () {
  ret = fib(4);
  printf(“%d\n”, ret);
  return 0;
}
```



Example Function Call: Recursion

```
#include <stdio.h>
#define MAXLEN 10

int ret;

/* function declaration */
int fib(int n) {
  int a, b;
              { /* n = 0 <= 1 */
    return n;

  a = fib(n-1)
  b = fib(n-2)
  return a + b;
}

int main () {
  ret = fib(4);
  printf(“%d\n”, ret);
  return 0;
}
```



Inputs: (int n) 0

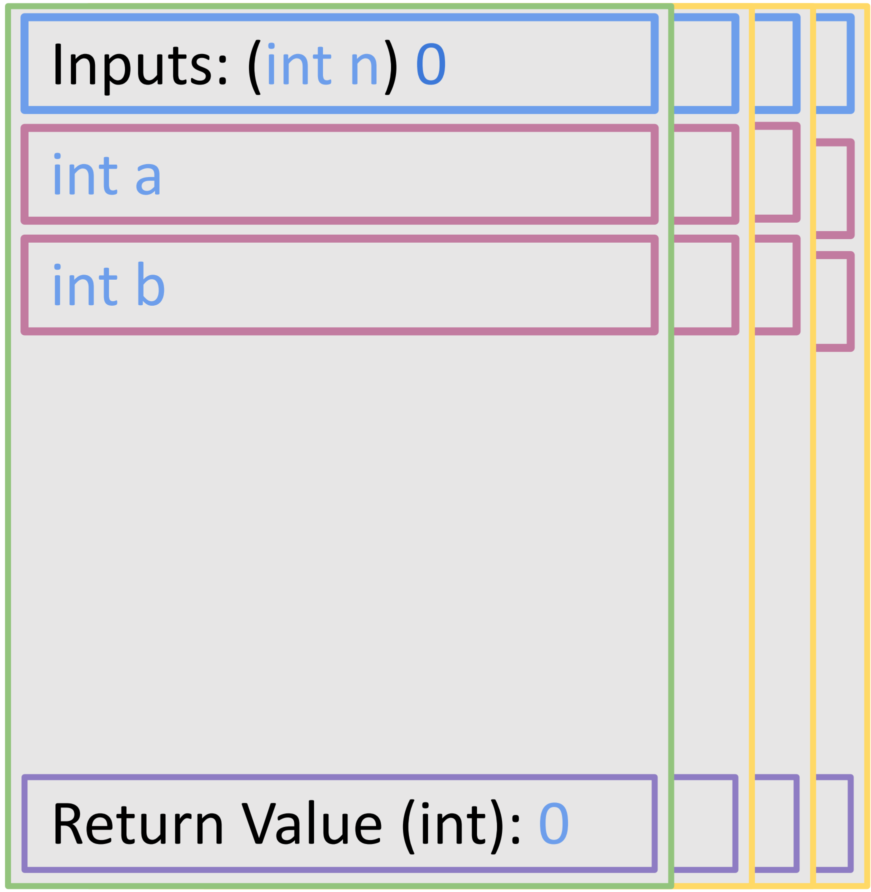
int a

int b

Return Value (int):

Example Function Call: Recursion

```
#include <stdio.h>
#define MAXLEN 10

int ret;

/* function declaration */
int fib(int n) {
  int a, b;
              { /* n = 0 <= 1 */


  a = fib(n-1)
  b = fib(n-2)
  return a + b;
}

int main () {
  ret = fib(4);
  printf("%d\n", ret);
  return 0;
}
```



Inputs: (int n) 0

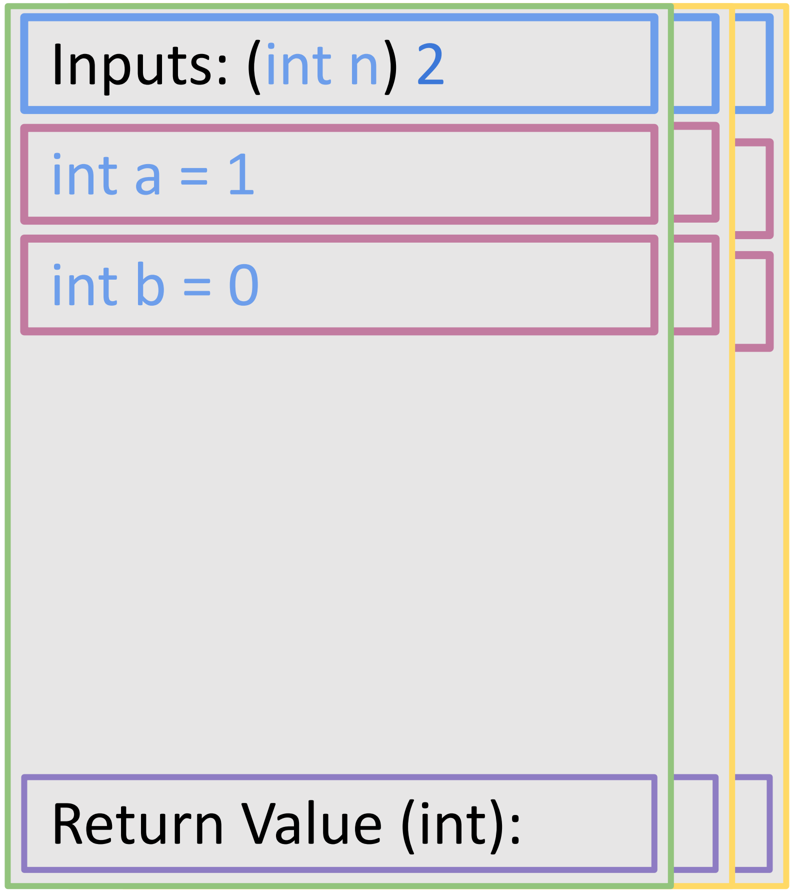
int a

int b

Return Value (int): 0

Example Function Call: Recursion

```
#include <stdio.h>
#define MAXLEN 10

int ret;

/* function declaration */
int fib(int n) {
  int a, b;
  if (n <= 1) { /* n = 2 > 1 */
    return n;
  }
  a = fib(n-1) /* n - 1 = 1 */

  return a + b;
}

int main () {
  ret = fib(4);
  printf(“%d\n”, ret);
  return 0;
}
```



Inputs: (int n) 2

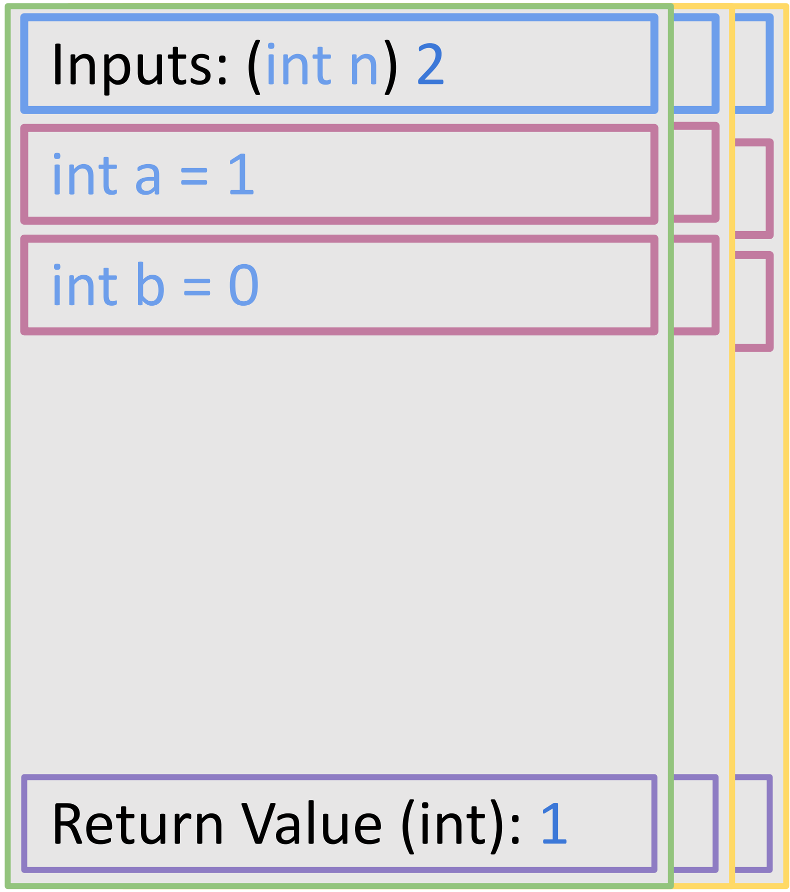
int a = 1

int b = 0

Return Value (int):

Example Function Call: Recursion

```
#include <stdio.h>
#define MAXLEN 10

int ret;

/* function declaration */
int fib(int n) {
  int a, b;
  if (n <= 1) { /* n = 2 > 1 */
    return n;
  }
  a = fib(n-1) /* n - 1 = 1 */
  b = fib(n-2) /* n - 2 = 0 */

}

int main () {
  ret = fib(4);
  printf(“%d\n”, ret);
  return 0;
}
```



Inputs: (int n) 2

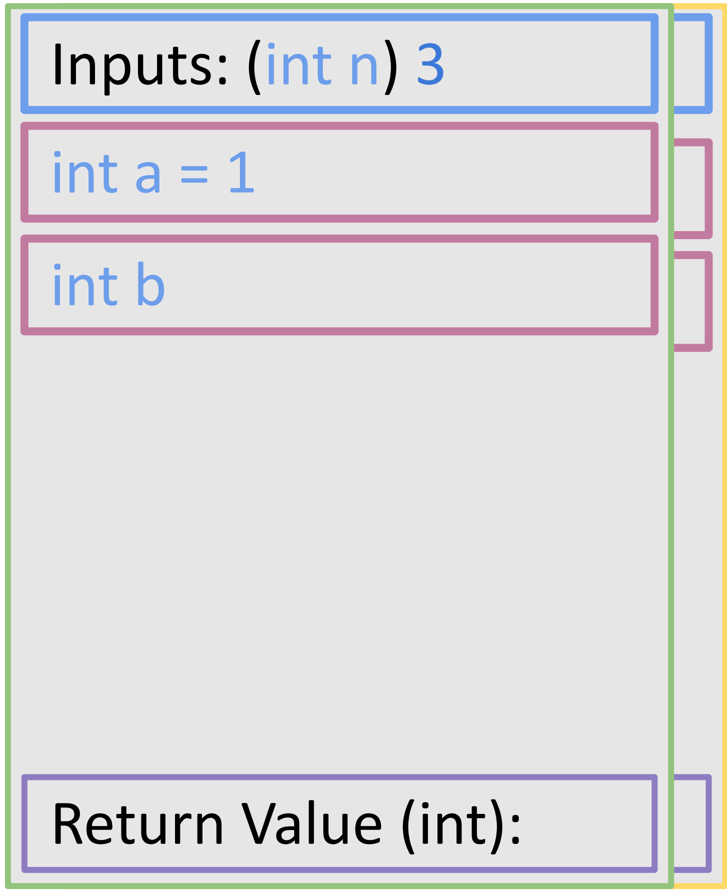
int a = 1

int b = 0

Return Value (int): 1

Example Function Call: Recursion

```
#include <stdio.h>
#define MAXLEN 10

int ret;

/* function declaration */
int fib(int n) {
  int a, b;
  if (n <= 1) { /* n = 3 > 1 */
    return n;
  }

  b = fib(n-2)
  return a + b;
}

int main () {
  ret = fib(4);
  printf(“%d\n”, ret);
  return 0;
}
```



Inputs: (int n) 3

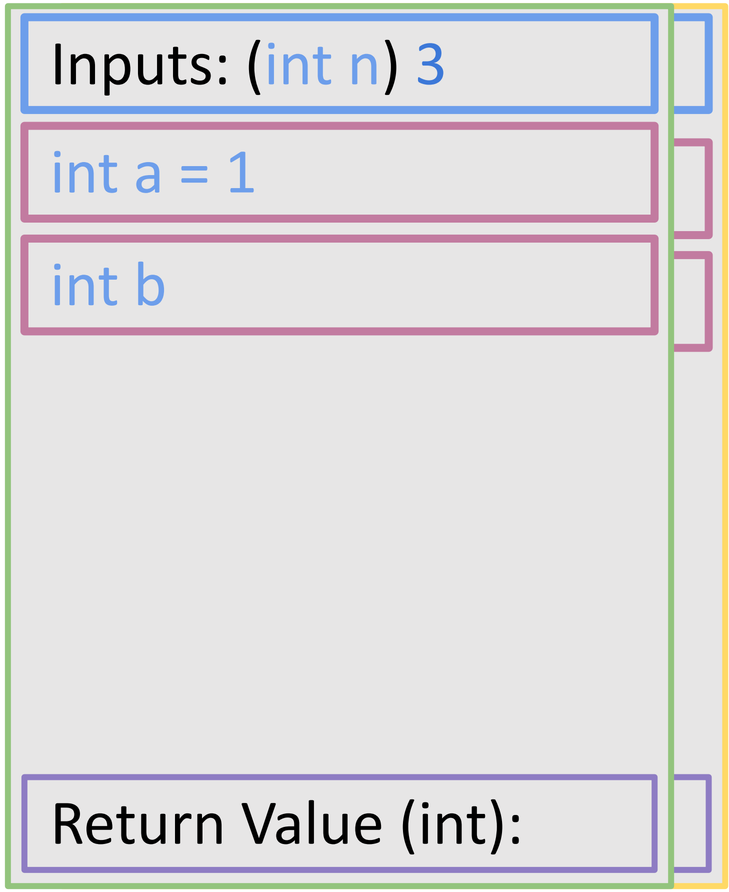
int a = 1

int b

Return Value (int):

Example Function Call: Recursion

```
#include <stdio.h>
#define MAXLEN 10

int ret;

/* function declaration */
int fib(int n) {
  int a, b;
  if (n <= 1) { /* n = 3 > 1 */
    return n;
  }
  a = fib(n-1) /* n - 1 = 2 */

  return a + b;
}

int main () {
  ret = fib(4);
  printf(“%d\n”, ret);
  return 0;
}
```



Inputs: (int n) 3

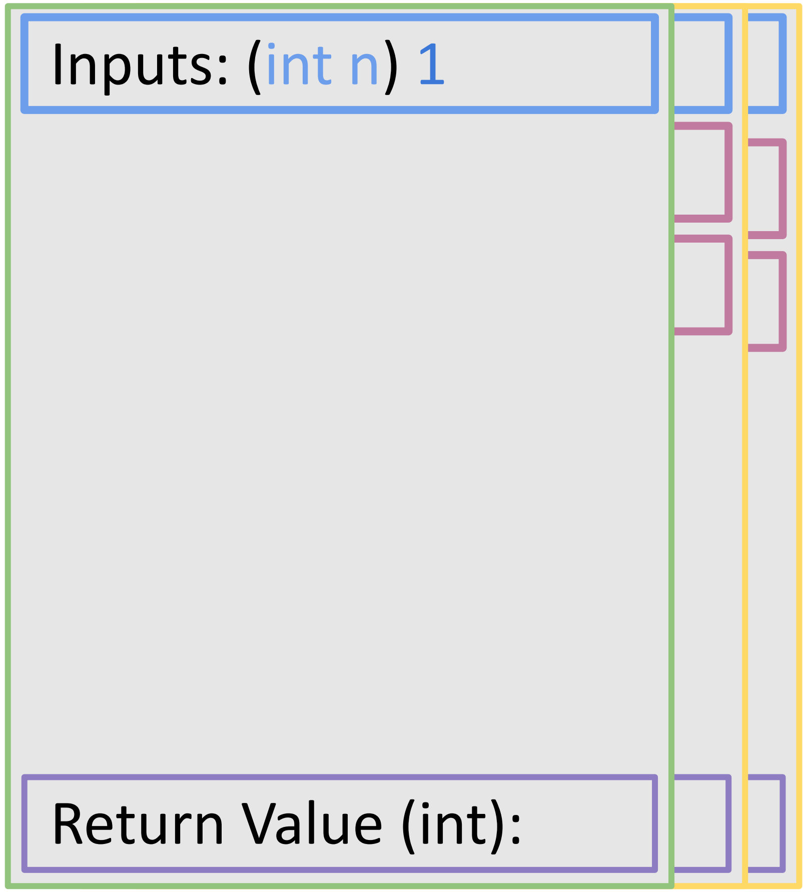
int a = 1

int b

Return Value (int):

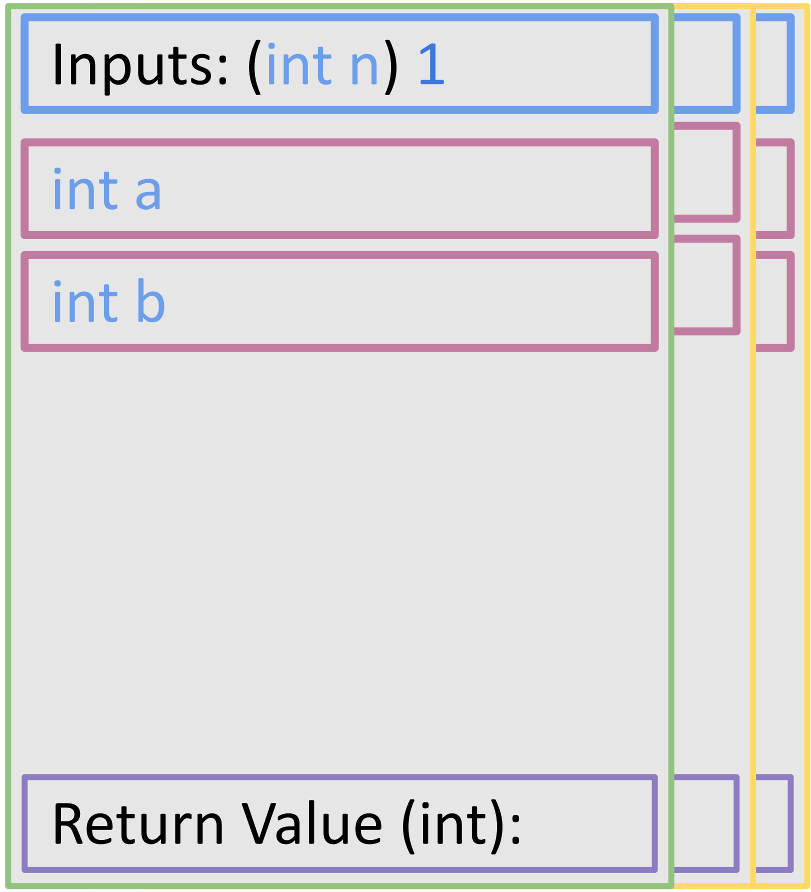
Example Function Call: Recursion

```
#include <stdio.h>
#define MAXLEN 10

int ret;

/* function declaration */
int fib(int n) {
  int a, b;
  if (n <= 1) { /* n = 3 > 1 */
    return n;
  }
  a = fib(n-1) /* n - 1 = 2 */
                          1
  return a + b;
}

int main () {
  ret = fib(4);
  printf(“%d\n”, ret);
  return 0;
}
```



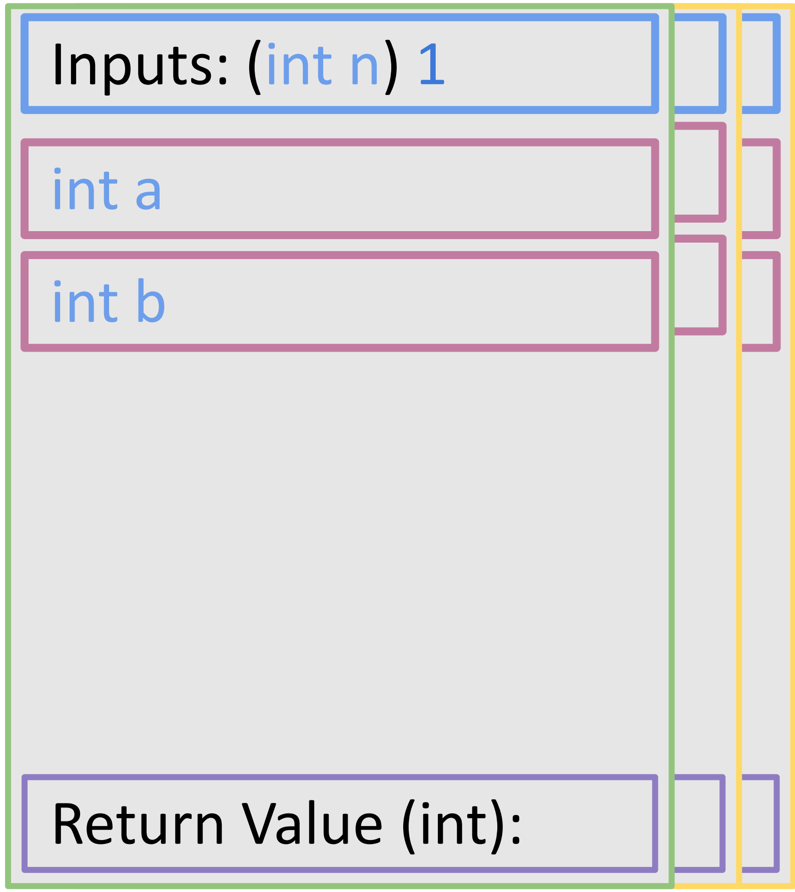
Example Function Call: Recursion

```
#include <stdio.h>
#define MAXLEN 10

int ret;

/* function declaration */
int fib(int n) {

  if (n <= 1) {
    return n;
  }
  a = fib(n-1)
  b = fib(n-2)
  return a + b;
}

int main () {
  ret = fib(4);
  printf(“%d\n”, ret);
  return 0;
}
```



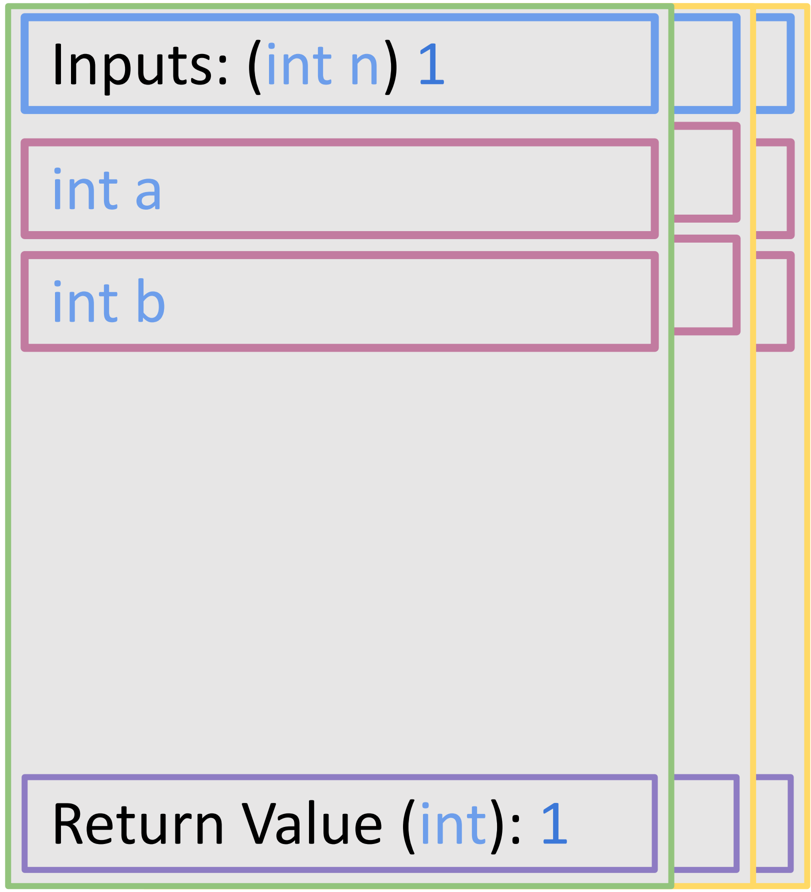
Example Function Call: Recursion

```
#include <stdio.h>
#define MAXLEN 10

int ret;

/* function declaration */
int fib(int n) {
  int a, b;
                /* n = 1 <= 1*/
    return n;
  }
  a = fib(n-1)
  b = fib(n-2)
  return a + b;
}

int main () {
  ret = fib(4);
  printf(“%d\n”, ret);
  return 0;
}
```



Example Function Call: Recursion

```
#include <stdio.h>
#define MAXLEN 10

int ret;

/* function declaration */
int fib(int n) {
  int a, b;
                /* n = 1 <= 1*/


  a = fib(n-1)
  b = fib(n-2)
  return a + b;
}

int main () {
  ret = fib(4);
  printf("%d\n", ret);
  return 0;
}
```



Inputs: (int n) 1

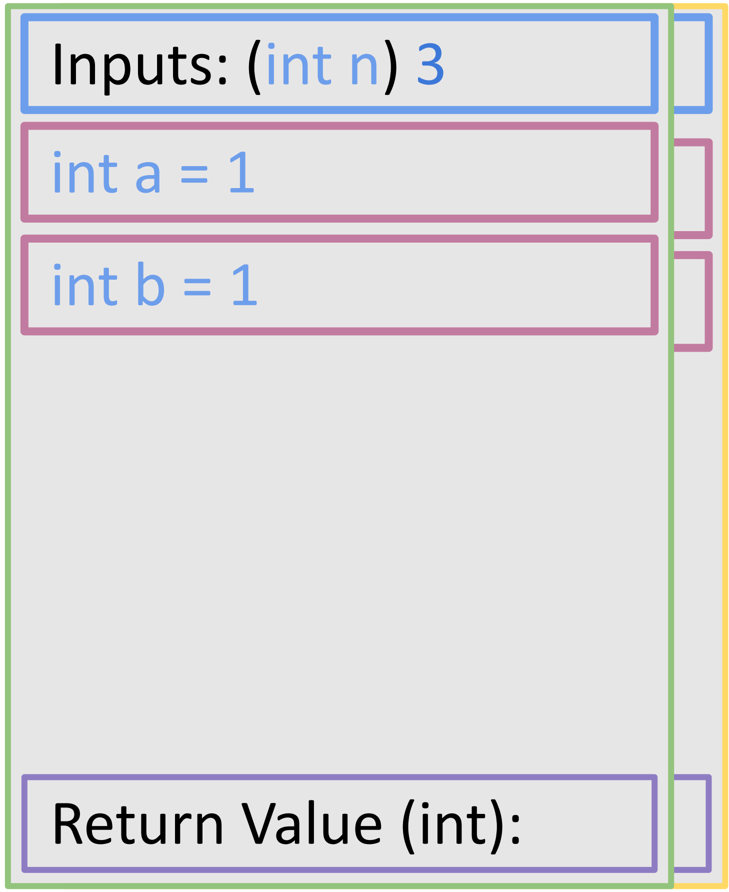
int a

int b

Return Value (int): 1

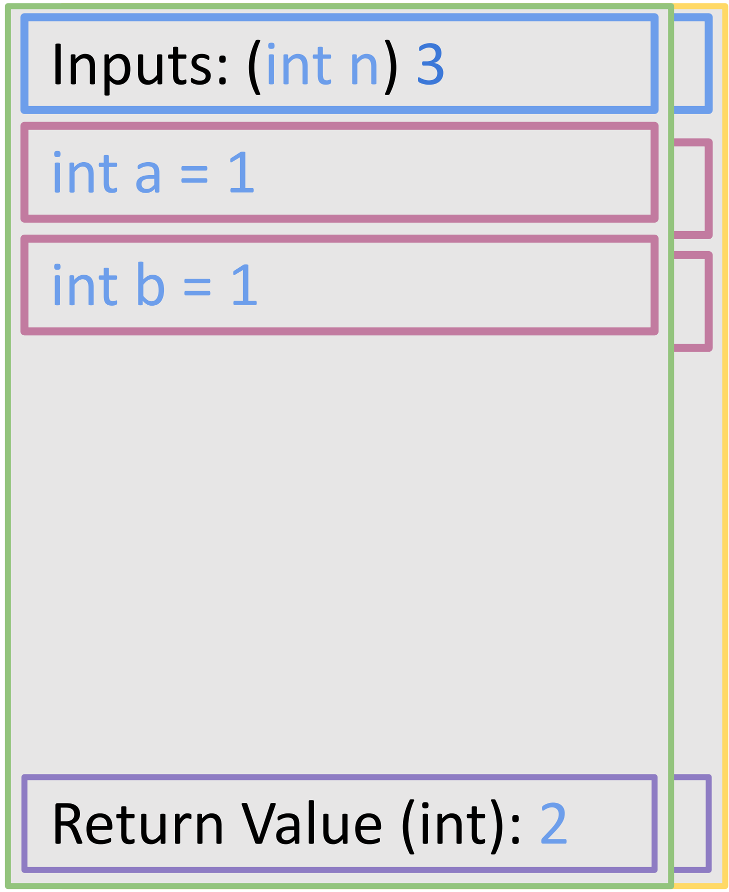
Example Function Call: Recursion

```
#include <stdio.h>
#define MAXLEN 10

int ret;

/* function declaration */
int fib(int n) {
  int a, b;
  if (n <= 1) { /* n = 3 > 1 */
    return n;
  }
  a = fib(n-1) /* n - 1 = 2 */

  return a + b;
}

int main () {
  ret = fib(4);
  printf(“%d\n”, ret);
  return 0;
}
```



Example Function Call: Recursion

```
#include <stdio.h>
#define MAXLEN 10

int ret;

/* function declaration */
int fib(int n) {
  int a, b;
  if (n <= 1) { /* n = 3 > 1 */
    return n;
  }
  a = fib(n-1) /* n - 1 = 2 */
  b = fib(n-2) /* n - 2 = 1 */

}

int main () {
  ret = fib(4);
  printf("%d\n", ret);
  return 0;
}
```



Inputs: (int n) 3

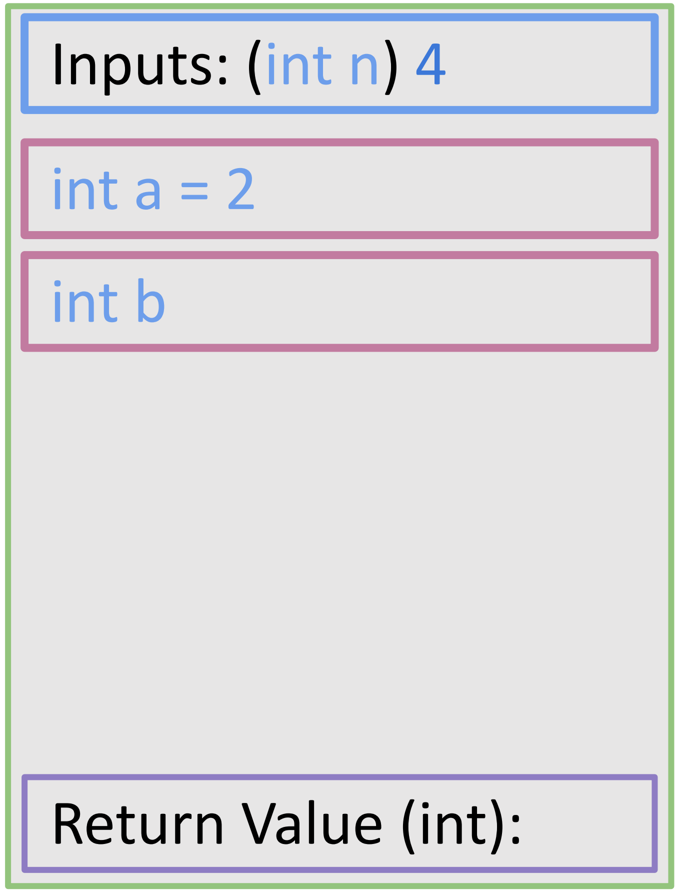
int a = 1

int b = 1

Return Value (int): 2

Example Function Call: Recursion

```
#include <stdio.h>
#define MAXLEN 10

int ret;

/* function declaration */
int fib(int n) {
  int a, b;
  if (n <= 1) {  /* n = 4 > 1 */
    return n;
  }

  b = fib(n-2)
  return a + b;
}

int main () {
  ret = fib(4);
  printf(“%d\n”, ret);
  return 0;
}
```



Inputs: (int n) 4

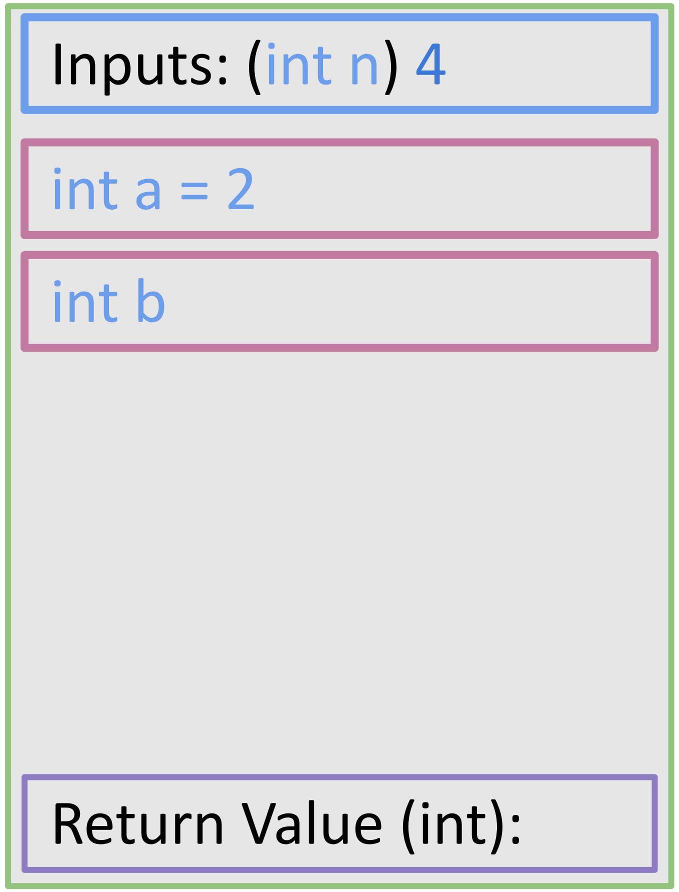
int a = 2

int b

Return Value (int):

Example Function Call: Recursion

```
#include <stdio.h>
#define MAXLEN 10

int ret;

/* function declaration */
int fib(int n) {
  int a, b;
  if (n <= 1) { /* n = 4 > 1 */
    return n;
  }
  a = fib(n-1) /* n - 1 = 3 */

  return a + b;
}

int main () {
  ret = fib(4);
  printf(“%d\n”, ret);
  return 0;
}
```



Inputs: (int n) 4

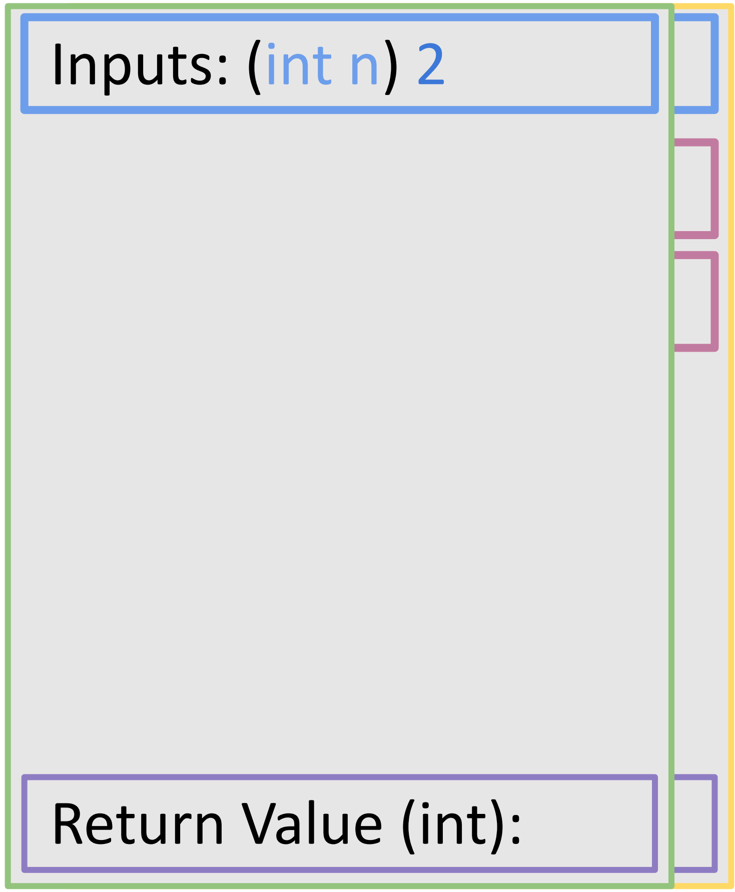
int a = 2

int b

Return Value (int):

Example Function Call: Recursion

```
#include <stdio.h>
#define MAXLEN 10

int ret;

/* function declaration */
int fib(int n) {
  int a, b;
  if (n <= 1) { /* n = 4 > 1 */
    return n;
  }
  a = fib(n-1) /* n - 1 = 3 */
                         2

  return a + b;
}

int main () {
  ret = fib(4);
  printf(“%d\n”, ret);
  return 0;
}
```

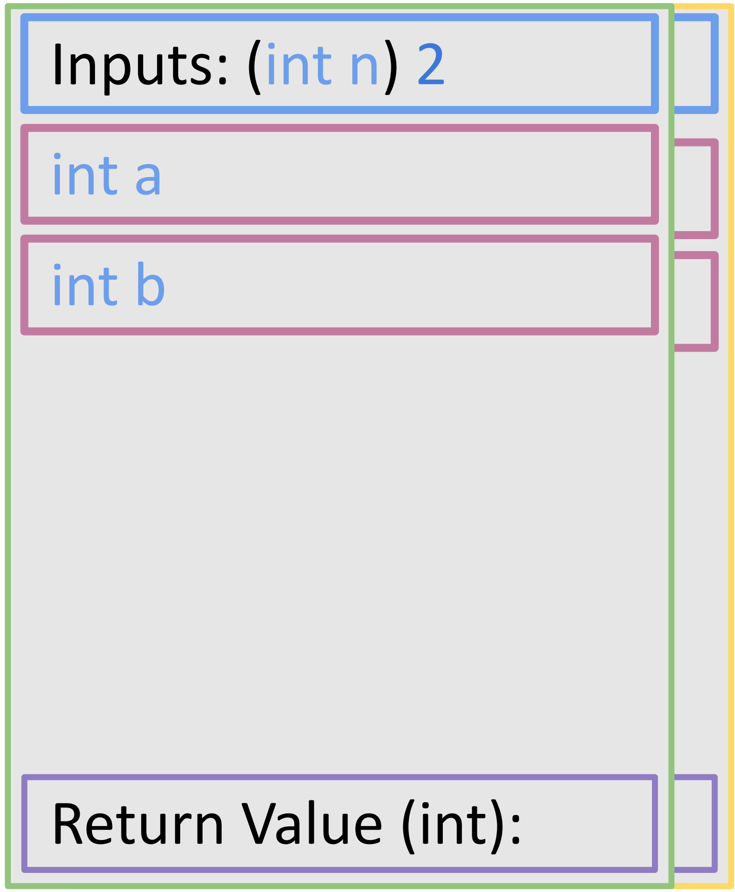


Inputs: (int n) 2

Return Value (int):

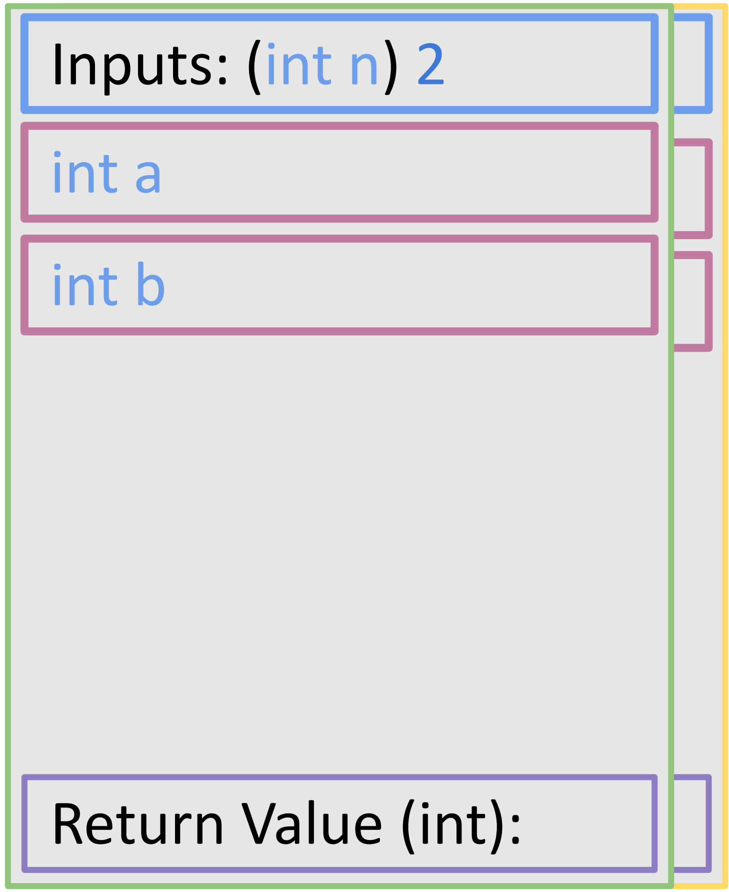
Example Function Call: Recursion

```
#include <stdio.h>
#define MAXLEN 10

int ret;

/* function declaration */
int fib(int n) {

  if (n <= 1) {
    return n;
  }
  a = fib(n-1)
  b = fib(n-2)
  return a + b;
}

int main () {
  ret = fib(4);
  printf("%d\n", ret);
  return 0;
}
```



Example Function Call: Recursion

```
#include <stdio.h>
#define MAXLEN 10

int ret;

/* function declaration */
int fib(int n) {
  int a, b;
                        2
    return n;
  }
  a = fib(n-1)
  b = fib(n-2)
  return a + b;
}

int main () {
  ret = fib(4);
  printf(“%d\n”, ret);
  return 0;
}
```



Inputs: (int n) 2

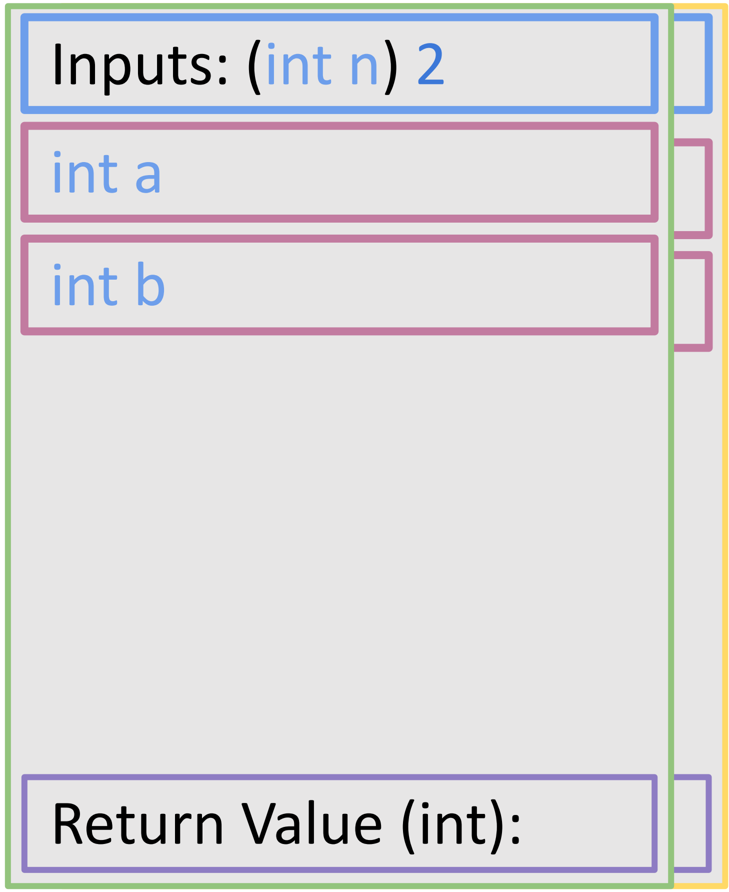
int a

int b

Return Value (int):

Example Function Call: Recursion

```
#include <stdio.h>
#define MAXLEN 10

int ret;

/* function declaration */
int fib(int n) {
  int a, b;
  if (n <= 1) {  /* n = 2 > 1 */
    return n;
  }

  b = fib(n-2)
  return a + b;
}

int main () {
  ret = fib(4);
  printf("%d\n", ret);
  return 0;
}
```



Inputs: (int n) 2

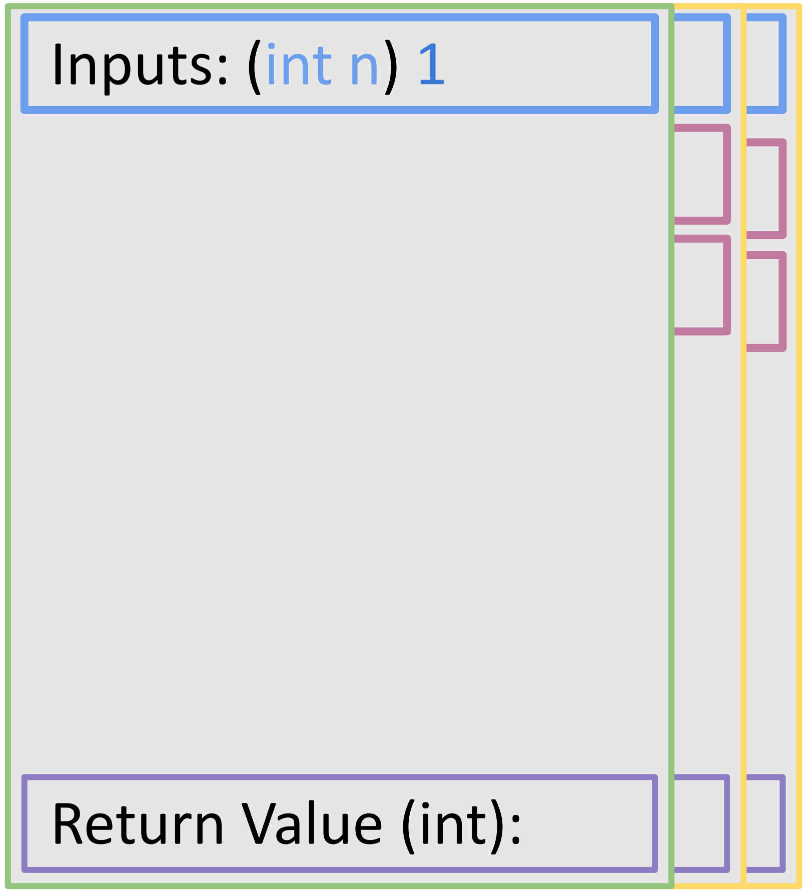
int a

int b

Return Value (int):

Example Function Call: Recursion

```
#include <stdio.h>
#define MAXLEN 10

int ret;

/* function declaration */
int fib(int n) {
  int a, b;
  if (n <= 1) { /* n = 2 > 1 */
    return n;
  }
                          1

  b = fib(n-2)
  return a + b;
}

int main () {
  ret = fib(4);
  printf(“%d\n”, ret);
  return 0;
}
```

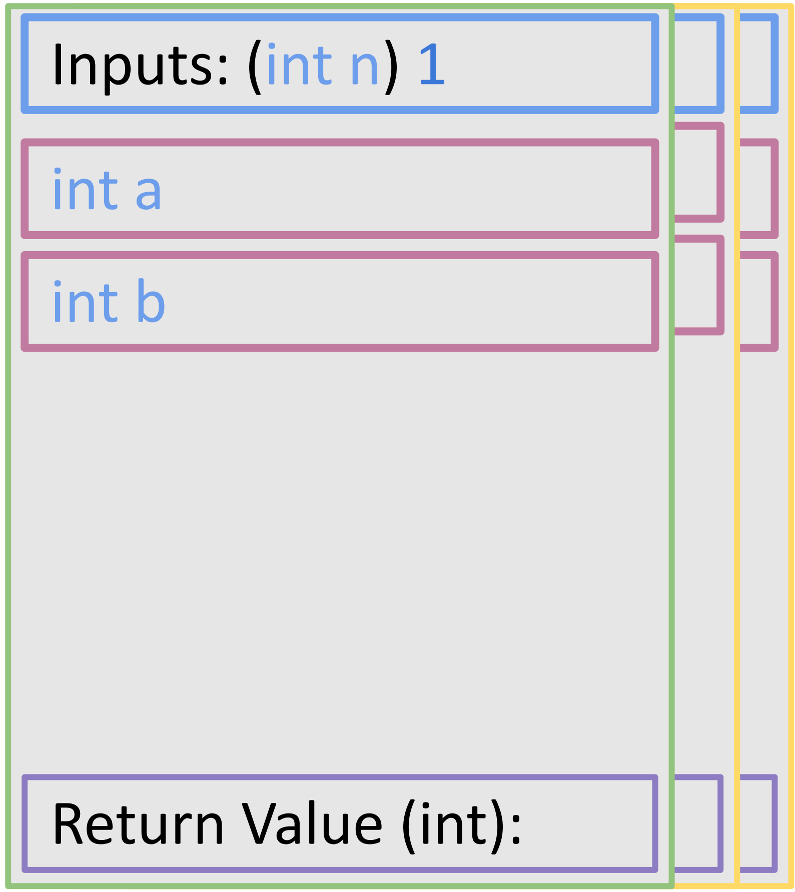


Inputs: (int n) 1

Return Value (int):

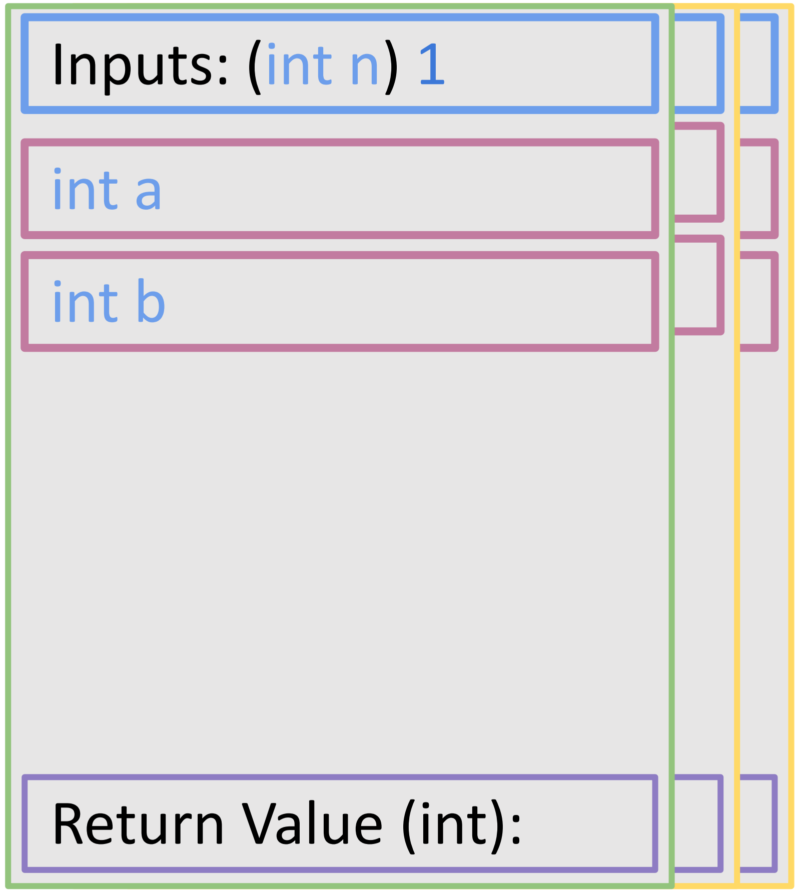
Example Function Call: Recursion

```
#include <stdio.h>
#define MAXLEN 10

int ret;

/* function declaration */
int fib(int n) {

  if (n <= 1) {
    return n;
  }
  a = fib(n-1)
  b = fib(n-2)
  return a + b;
}

int main () {
  ret = fib(4);
  printf(“%d\n”, ret);
  return 0;
}
```



Example Function Call: Recursion

```
#include <stdio.h>
#define MAXLEN 10

int ret;

/* function declaration */
int fib(int n) {
  int a, b;
                /* n = 1 <= 1*/
    return n;
  }
  a = fib(n-1)
  b = fib(n-2)
  return a + b;
}

int main () {
  ret = fib(4);
  printf(“%d\n”, ret);
  return 0;
}
```



Inputs: (int n) 1

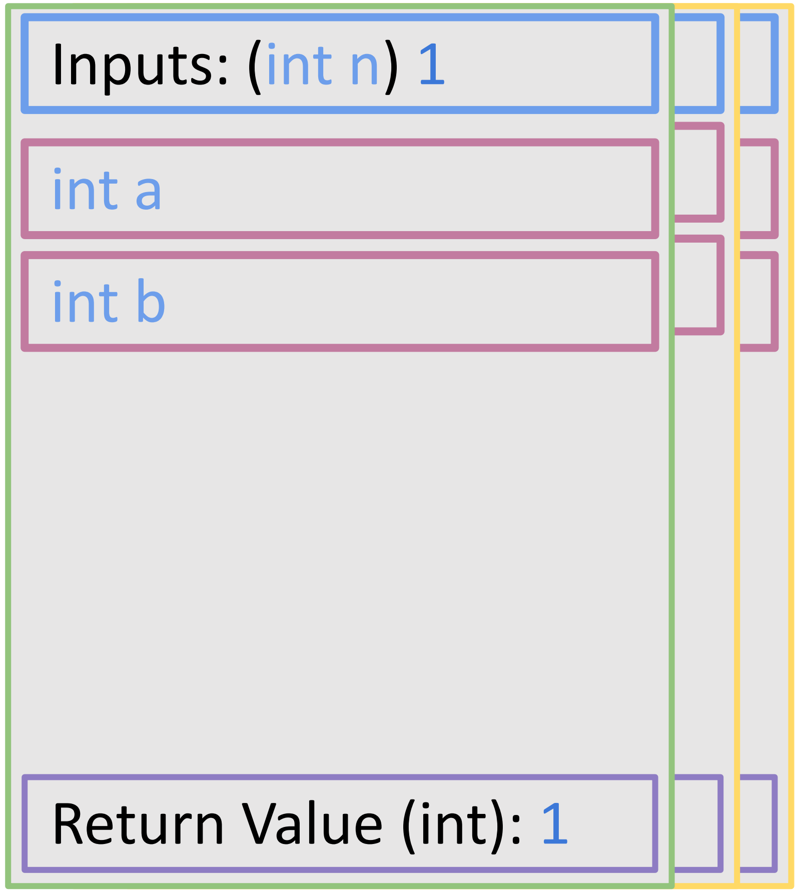
int a

int b

Return Value (int):

Example Function Call: Recursion

```
#include <stdio.h>
#define MAXLEN 10

int ret;

/* function declaration */
int fib(int n) {
  int a, b;
                /* n = 1 <= 1*/


  a = fib(n-1)
  b = fib(n-2)
  return a + b;
}

int main () {
  ret = fib(4);
  printf(“%d\n”, ret);
  return 0;
}
```



Inputs: (int n) 1

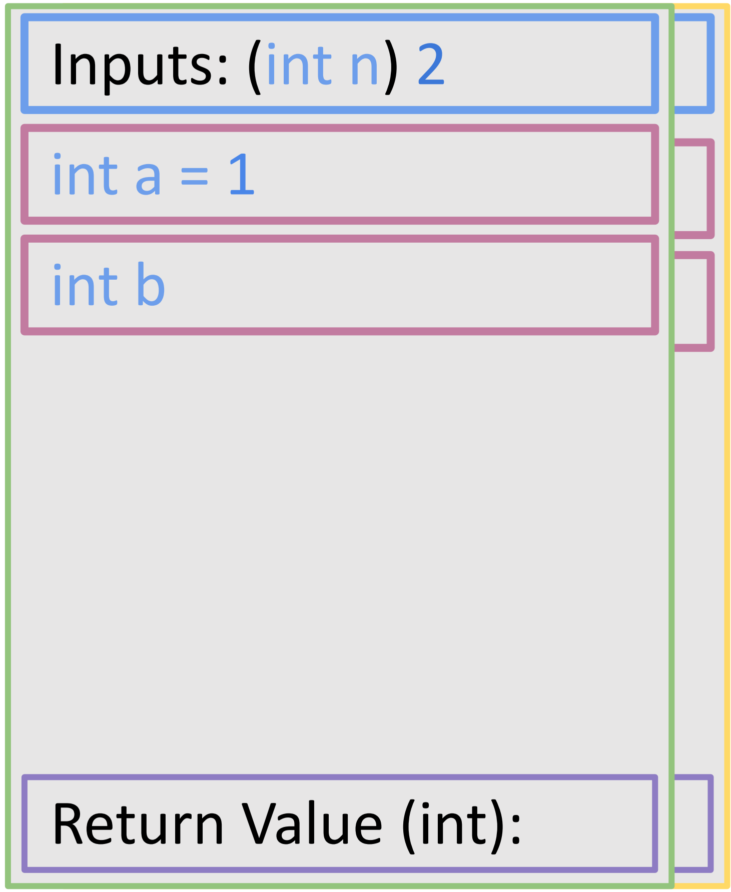
int a

int b

Return Value (int): 1

Example Function Call: Recursion

```
#include <stdio.h>
#define MAXLEN 10

int ret;

/* function declaration */
int fib(int n) {
  int a, b;
  if (n <= 1) { /* n = 2 > 1 */
    return n;
  }

  b = fib(n-2)
  return a + b;
}

int main () {
  ret = fib(4);
  printf("%d\n", ret);
  return 0;
}
```



Inputs: (int n) 2

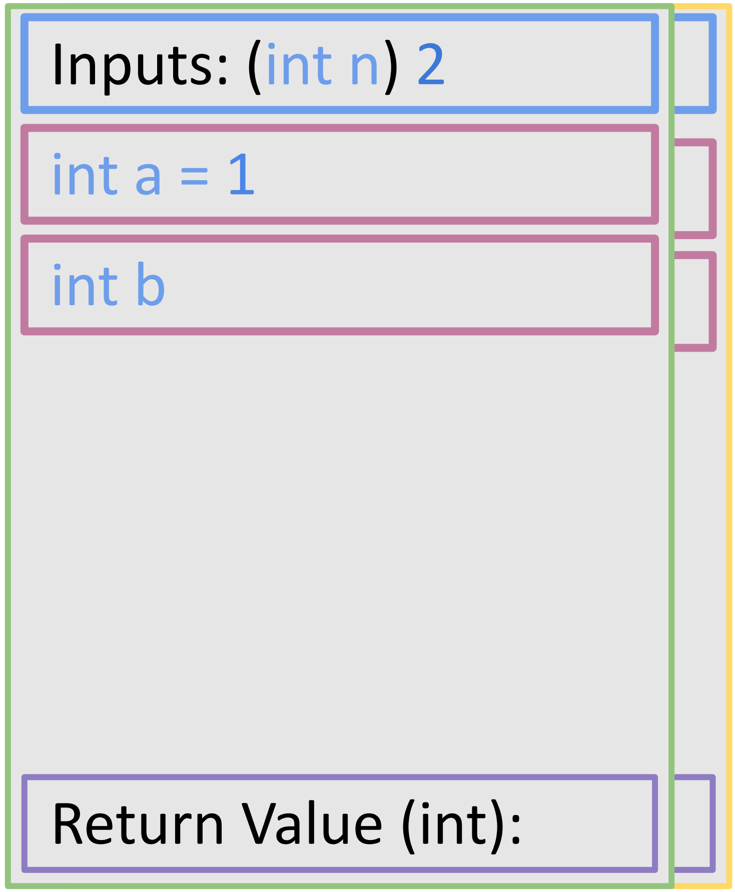
int a = 1

int b

Return Value (int):

Example Function Call: Recursion

```
#include <stdio.h>
#define MAXLEN 10

int ret;

/* function declaration */
int fib(int n) {
  int a, b;
  if (n <= 1) { /* n = 2 > 1 */
    return n;
  }
  a = fib(n-1) /* n - 1 = 1 */

  return a + b;
}

int main () {
  ret = fib(4);
  printf(“%d\n”, ret);
  return 0;
}
```



Inputs: (int n) 2

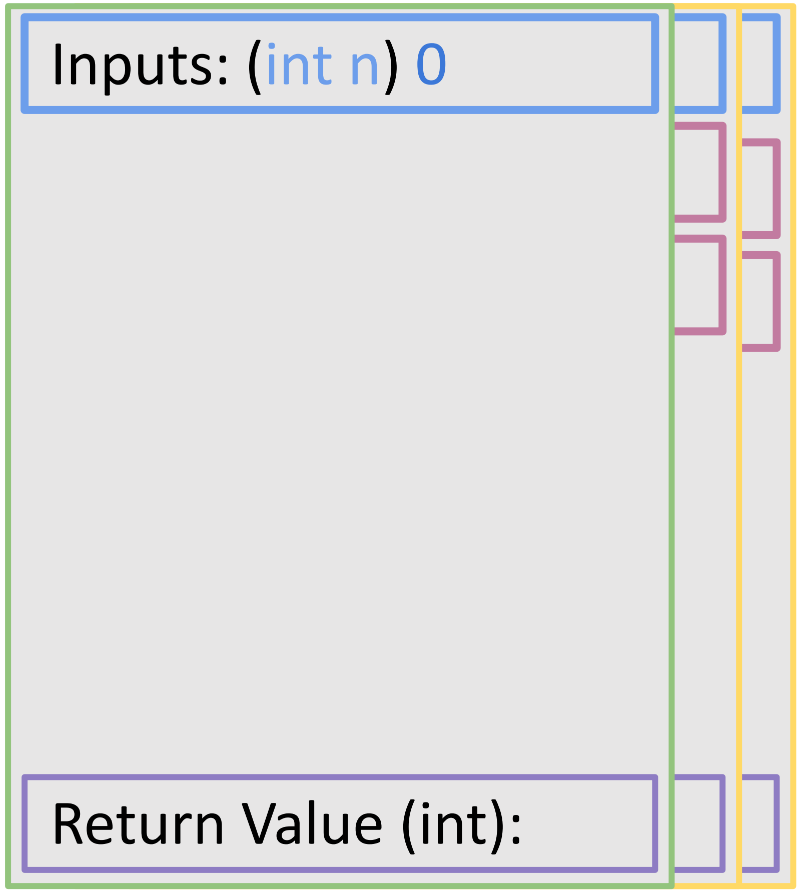
int a = 1

int b

Return Value (int):

Example Function Call: Recursion

```
#include <stdio.h>
#define MAXLEN 10

int ret;

/* function declaration */
int fib(int n) {
  int a, b;
  if (n <= 1) { /* n = 2 > 1 */
    return n;
  }
  a = fib(n-1) /* n - 1 = 1 */
                          0

  return a + b;
}

int main () {
  ret = fib(4);
  printf(“%d\n”, ret);
  return 0;
}
```

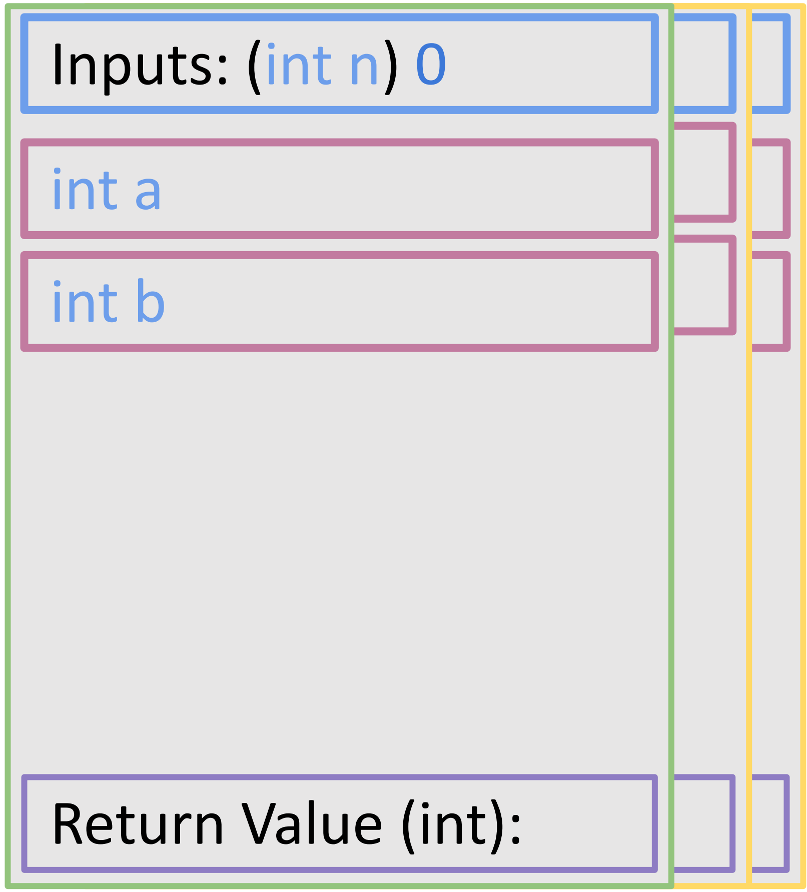


Inputs: (int n) 0

Return Value (int):

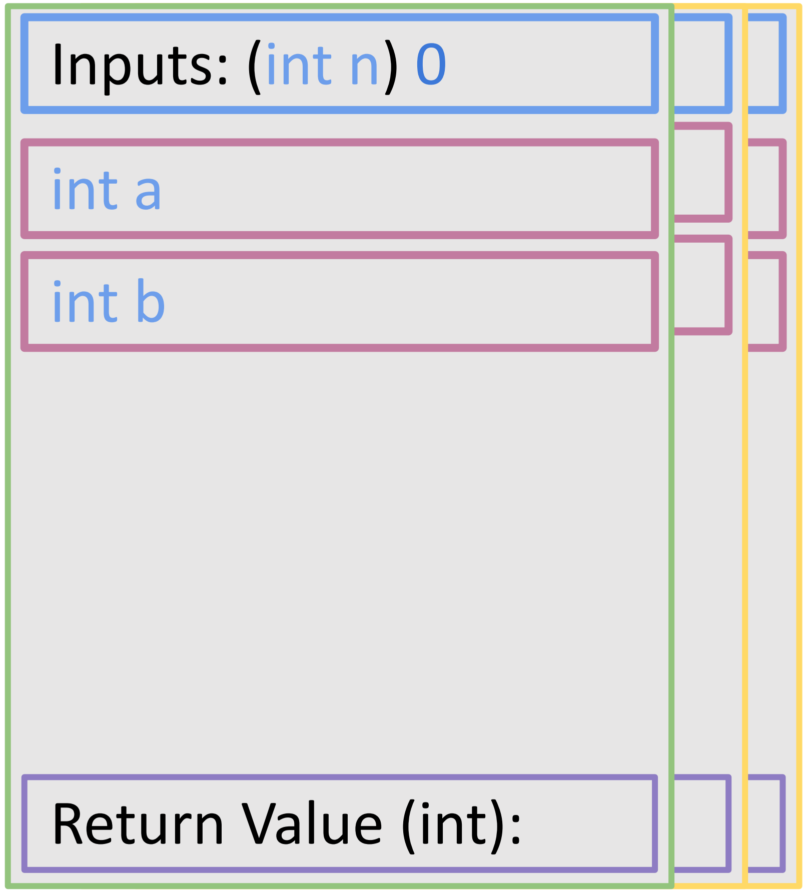
Example Function Call: Recursion

```
#include <stdio.h>
#define MAXLEN 10

int ret;

/* function declaration */
int fib(int n) {

  if (n <= 1) {
    return n;
  }
  a = fib(n-1)
  b = fib(n-2)
  return a + b;
}

int main () {
  ret = fib(4);
  printf(“%d\n”, ret);
  return 0;
}
```



Example Function Call: Recursion

```
#include <stdio.h>
#define MAXLEN 10

int ret;

/* function declaration */
int fib(int n) {
  int a, b;
              { /* n = 0 <= 1 */
    return n;

  a = fib(n-1)
  b = fib(n-2)
  return a + b;
}

int main () {
  ret = fib(4);
  printf(“%d\n”, ret);
  return 0;
}
```



Inputs: (int n) 0

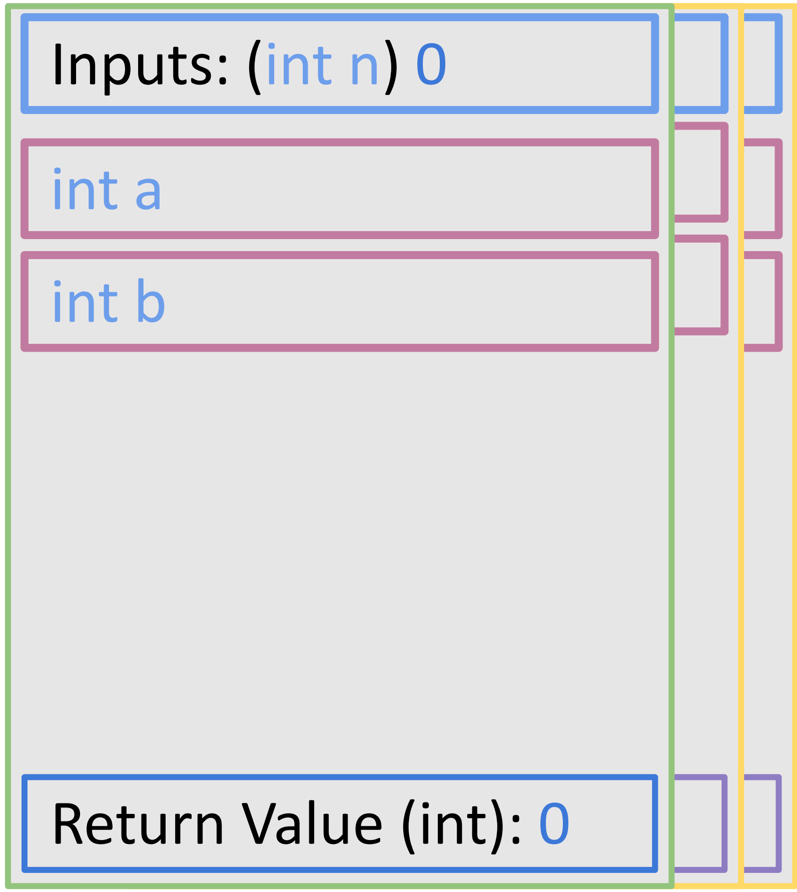
int a

int b

Return Value (int):

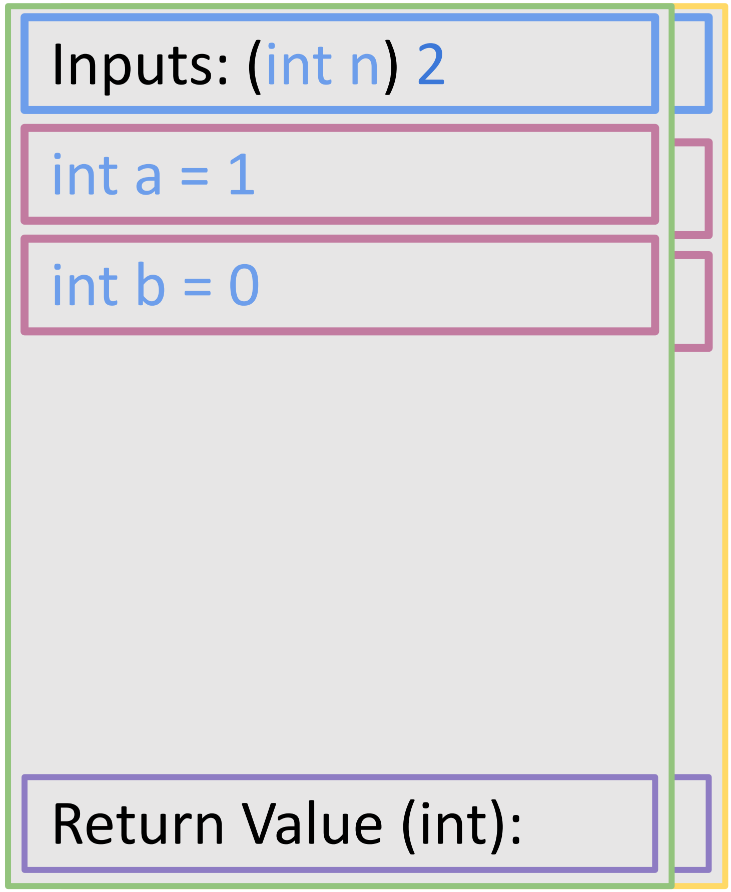
Example Function Call: Recursion

```
#include <stdio.h>
#define MAXLEN 10

int ret;

/* function declaration */
int fib(int n) {
  int a, b;
              { /* n = 0 <= 1 */


  a = fib(n-1)
  b = fib(n-2)
  return a + b;
}

int main () {
  ret = fib(4);
  printf(“%d\n”, ret);
  return 0;
}
```



Example Function Call: Recursion

```
#include <stdio.h>
#define MAXLEN 10

int ret;

/* function declaration */
int fib(int n) {
  int a, b;
  if (n <= 1) { /* n = 2 > 1 */
    return n;
  }
  a = fib(n-1) /* n - 1 = 1 */

  return a + b;
}

int main () {
  ret = fib(4);
  printf(“%d\n”, ret);
  return 0;
}
```



Inputs: (int n) 2

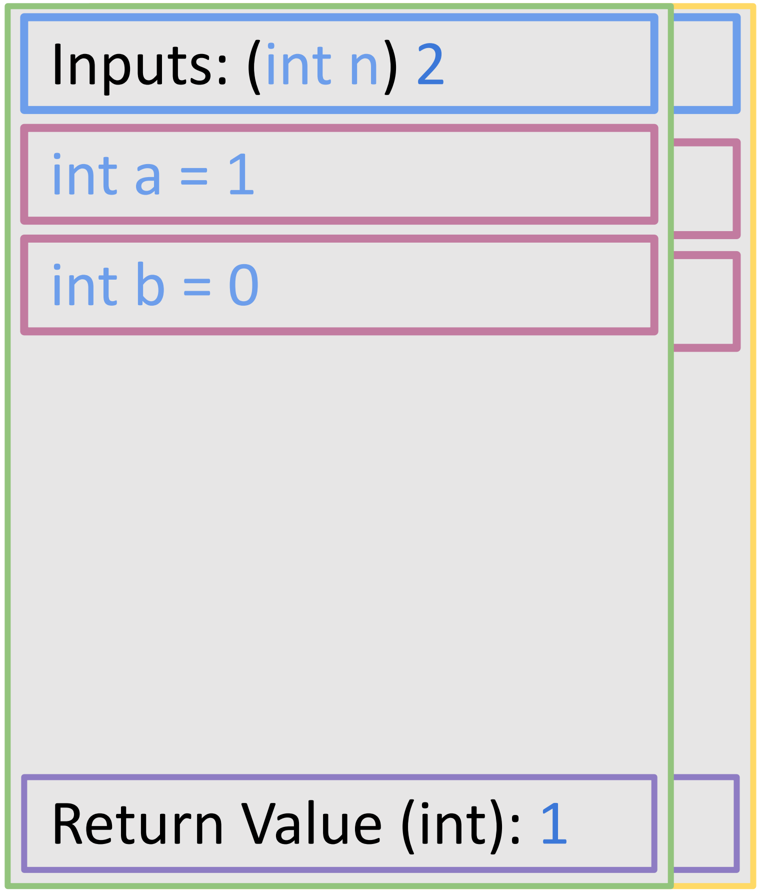
int a = 1

int b = 0

Return Value (int):

Example Function Call: Recursion

```
#include <stdio.h>
#define MAXLEN 10

int ret;

/* function declaration */
int fib(int n) {
  int a, b;
  if (n <= 1) { /* n = 2 > 1 */
    return n;
  }
  a = fib(n-1) /* n - 1 = 1 */
  b = fib(n-2) /* n - 2 = 0 */

}

int main () {
  ret = fib(4);
  printf(“%d\n”, ret);
  return 0;
}
```



Inputs: (int n) 2

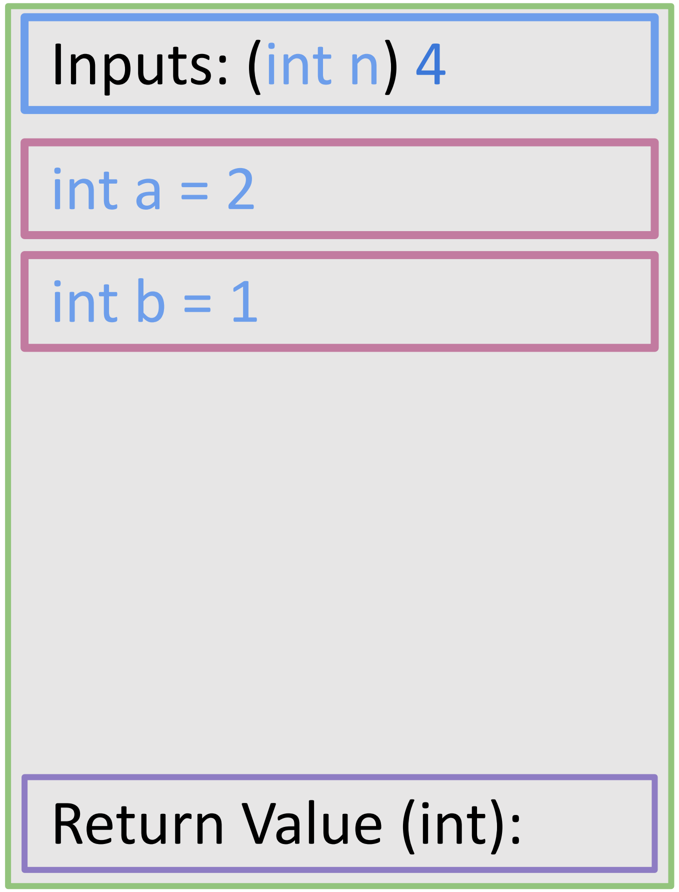
int a = 1

int b = 0

Return Value (int): 1

Example Function Call: Recursion

```
#include <stdio.h>
#define MAXLEN 10

int ret;

/* function declaration */
int fib(int n) {
  int a, b;
  if (n <= 1) { /* n = 4 > 1 */
    return n;
  }
  a = fib(n-1) /* n - 1 = 3 */

  return a + b;
}

int main () {
  ret = fib(4);
  printf(“%d\n”, ret);
  return 0;
}
```



Inputs: (int n) 4

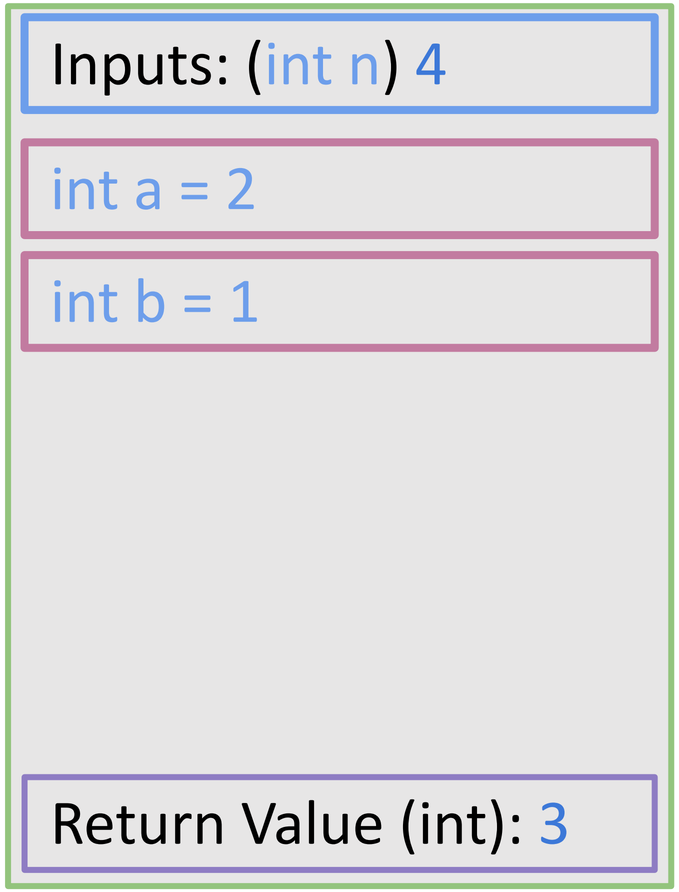
int a = 2

int b = 1

Return Value (int):

Example Function Call: Recursion

```
#include <stdio.h>
#define MAXLEN 10

int ret;

/* function declaration */
int fib(int n) {
  int a, b;
  if (n <= 1) { /* n = 4 > 1 */
    return n;
  }
  a = fib(n-1) /* n - 1 = 3 */

  return a + b;
}

int main () {
  ret = fib(4);
  printf(“%d\n”, ret);
  return 0;
}
```



Inputs: (int n) 4

int a = 2

int b = 1

Return Value (int): 3