Dynamic Memory Allocation

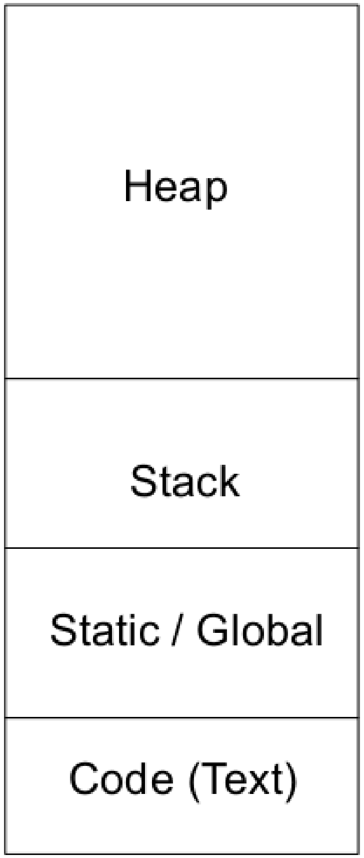
Memory statically, automatically, or dynamically

- Static-duration variables
  - Allocated in main memory
  - Along with the executable code of the program
  - Persist for the lifetime of the program
- Automatic-duration variables
  - Allocated on the stack
  - Come and go as functions are called and return
  - Stack size predefined by the compiler
  - Stack LIFO structure
  - Stack size OS and system architecture
- Both Static and Automatic require
  - Size of the allocation must be compile-time constant

Memory Allocation

```
int *ptr;
ptr = (int *) malloc(sizeof(int));
// Allocate an "unamed" (no variable name)
// memory location large enough to hold an int.
free(ptr);

// Arrays
char text[] = “Hello”;
char *cp;

cp = (char *) malloc(sizeof(char)10);
//         ________________________
// text : |_h_|_e_|_l_|_l_|_o_|_\0_|
//       ___     ____________________________________
// cp : |_*_| -> |_h_|_e_|_l_|_l_|_o_|_\0_|__|__|__|__|

text[2]
cp[2]
```

Memory Allocation



Free Storage < Heap

The label is also used in data structure but it does refer to same. Just label for large free pool of memory.

Function Calls :
Local variables
(exist in function)

Stack

> The amount of memory for these does not grow while application is running.

Global :
life of application

Static / Global

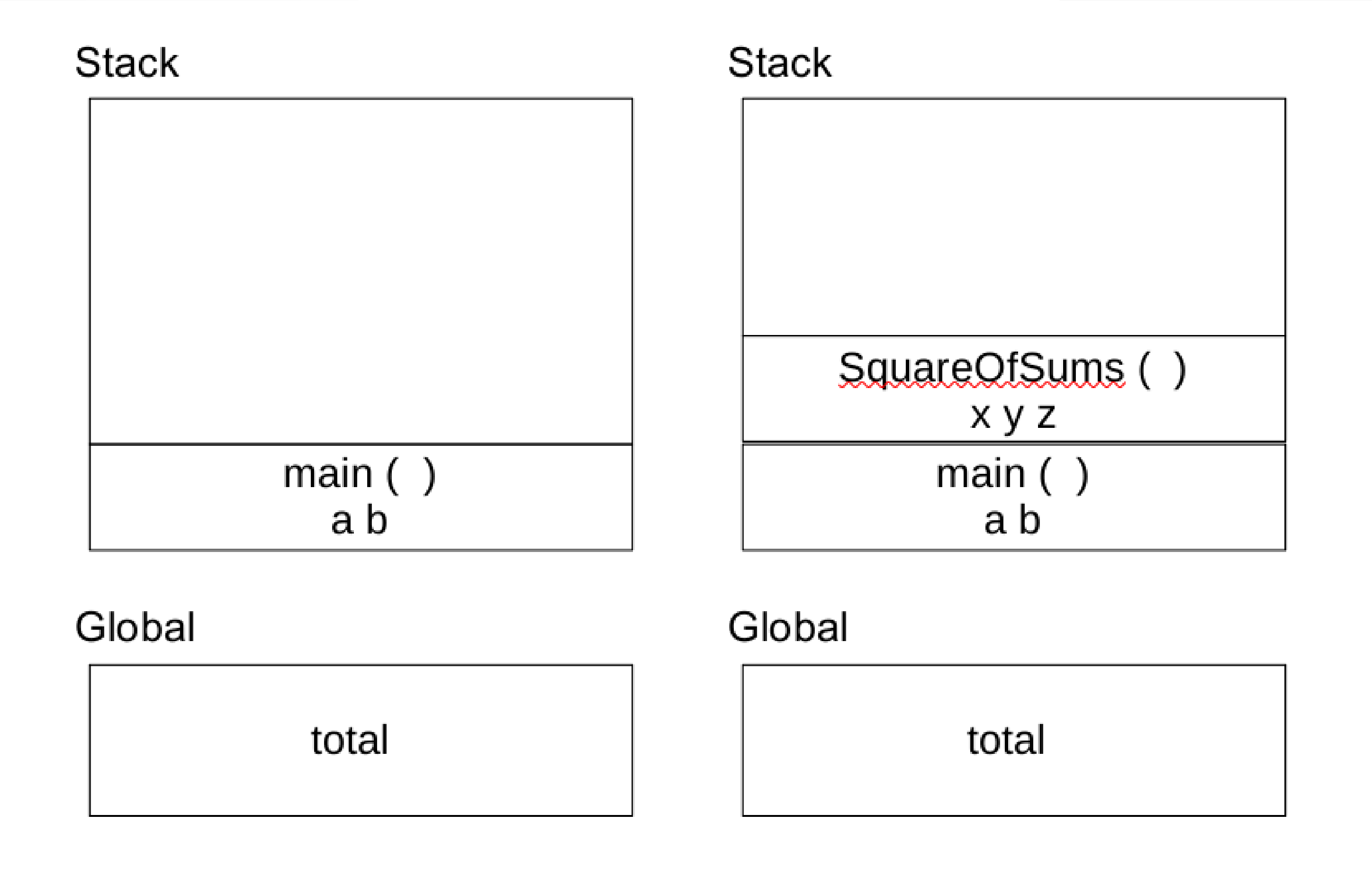
Instructions

Code (Text)

Program Memory Allocation

```c
#include <stdio.h>

int total; // only as an example
int Square(int);
int SquareOfSum(int,int);

int main() {
    int a = 4, b = 8;
    total = SquareOfSum(a,b);
    printf("Output = %d\n",total);
}

int Square(int x) { return x*x; }

int SquareOfSum(int x,int y) {
    int z = Square(x+y);
    return z; }
```

Program Memory Allocation



Stack

main ( )
a b

Global

total

Stack

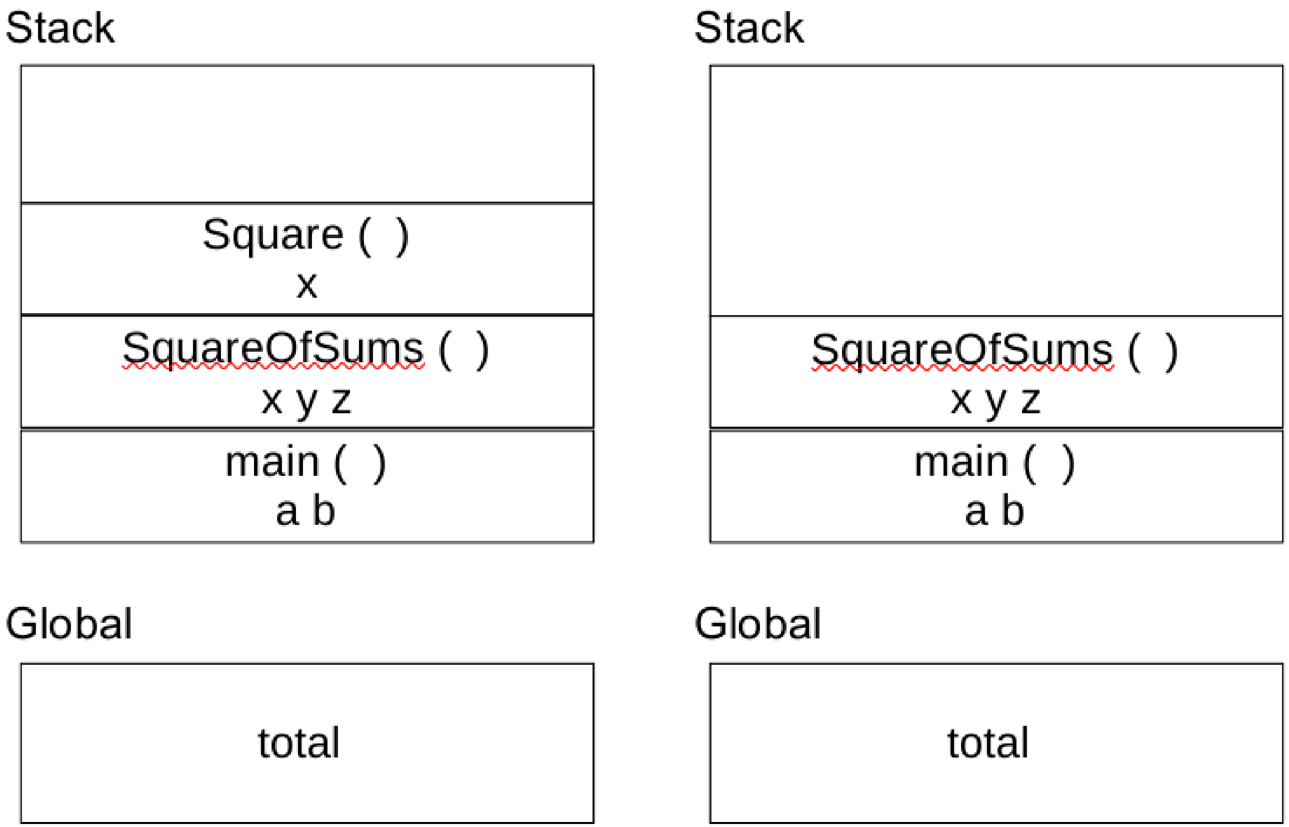
SquareOfSums ( )
x y z

main ( )
a b

Global

total

Program Memory Allocation



Stack

Square ( )
x

SquareOfSums ( )
x y z

main ( )
a b

Global

total

Stack

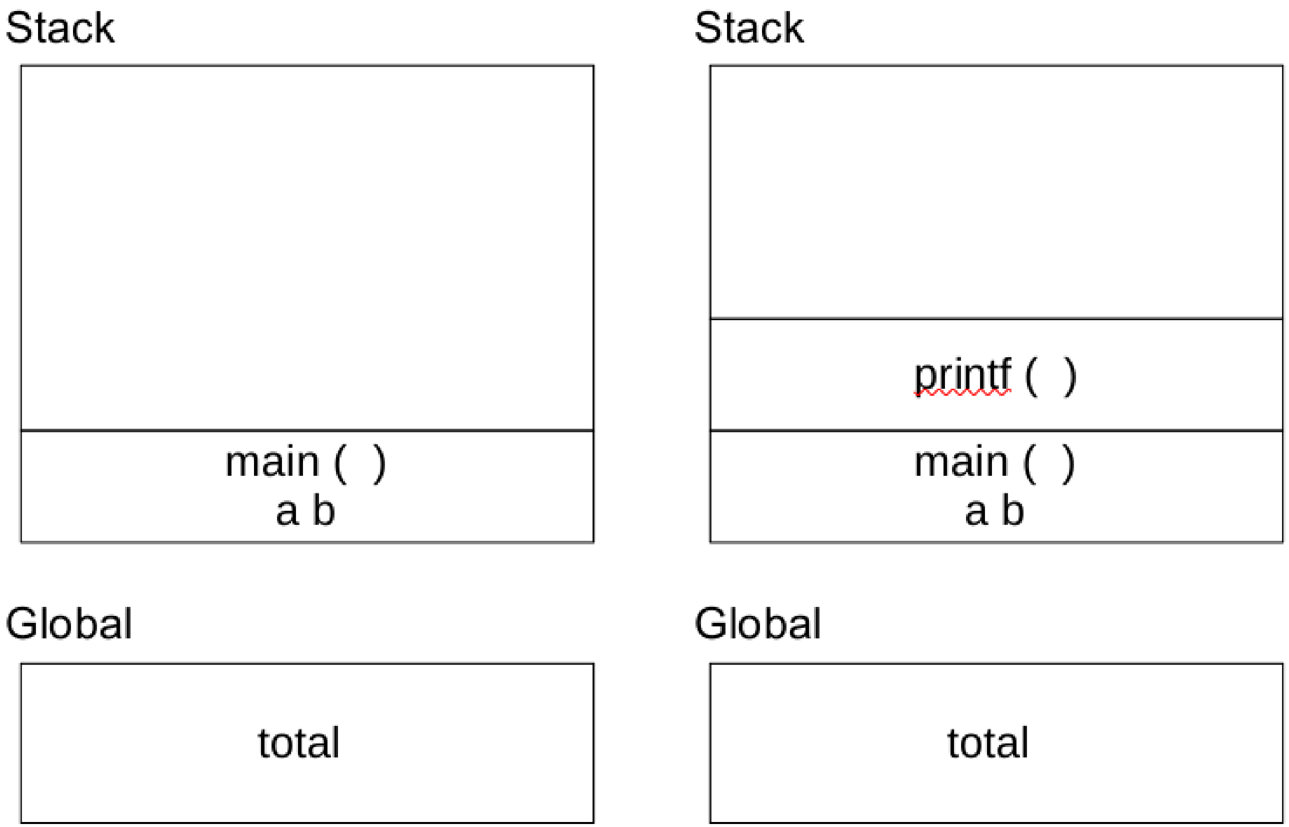
SquareOfSums ( )
x y z

main ( )
a b

Global

total

Program Memory Allocation



Stack

main ( )
a b

Global

total

Stack

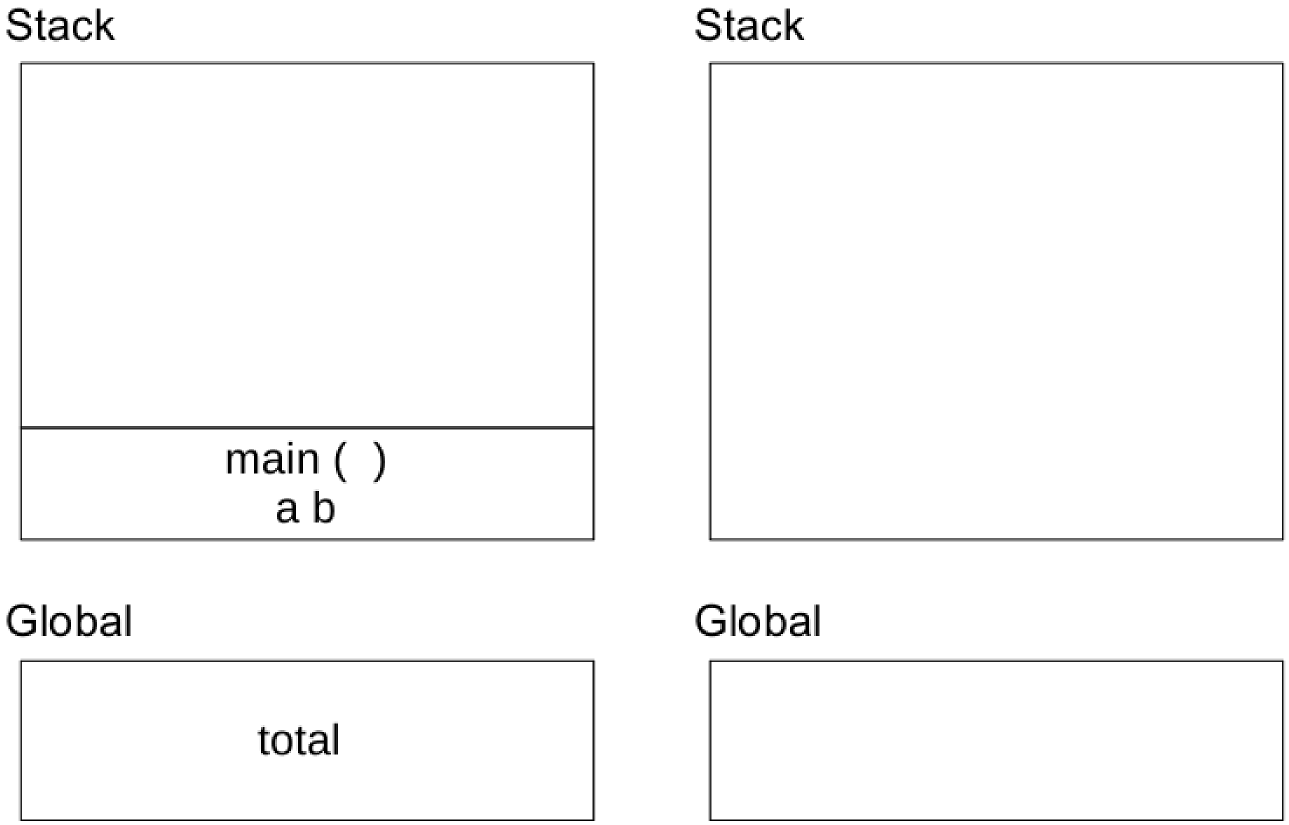
printf ( )

main ( )
a b

Global

total

Program Memory Allocation



Stack

main ( )
a b

Global

total

Stack

Global

Dynamic Memory Allocation

- Fixed-size data objects is inadequate
- memory is more explicitly (but more flexibly) managed
- Refers to performing manual memory management
- Allows program to allocate/de-allocate memory at runtime
- Used when size of memory is not known at compile-time
- Four library functions provided defined in <stdlib.h>:
    - malloc() or "memory allocation"
    - calloc() or "contiguous allocation"
    - free() or "memory de-allocation"
    - realloc() or "re-allocation"

malloc(): Memory Allocation

- Used to allocate a single large block of memory
- Returns a pointer of type void*
- Pointer can be cast into a pointer of any form.
- Does not initialize memory at execution time
- Initial value of the allocated block is unknown

```
ptr = (cast-type*) malloc(byte-size);
// Example:
ptr = (int*) malloc(100 * sizeof(int));
```

calloc(): Contiguous Allocation

- Used to allocate a specified amount of memory
- Initialize it to zero
- Returns a void pointer to this memory location
- Pointer can be cast to the desired type
- Provide function specify number of elements and type
- Provided the number of elements and type
- Initializes all bits to zero

```
ptr = (cast-type*)calloc(n, element-size);
// Example:
ptr = (float*) calloc(25, sizeof(float));
```

Questions?