The Tromino Tiling Puzzle

— Pointers, (2D-)Arrays, and Recursion

魏恒峰

hfwei@nju.edu.cn

2017 年 10 月 27 日

A Quick Review of Our Last Class

Functions

```
int solve_josephus(int n);
```

```
int main(void) {
   ...
   // call the function
   int survivor = solve_josephus(int n);
   ...
}
```

```
int solve_josephus(int n) {
}
```

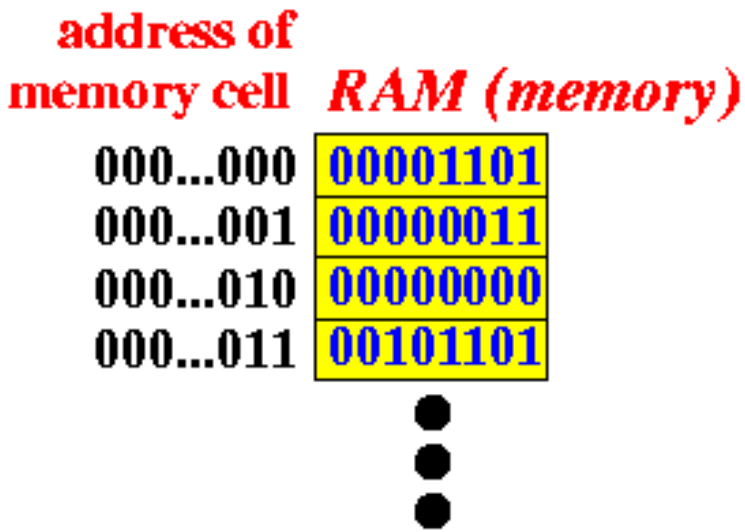
1D Arrays

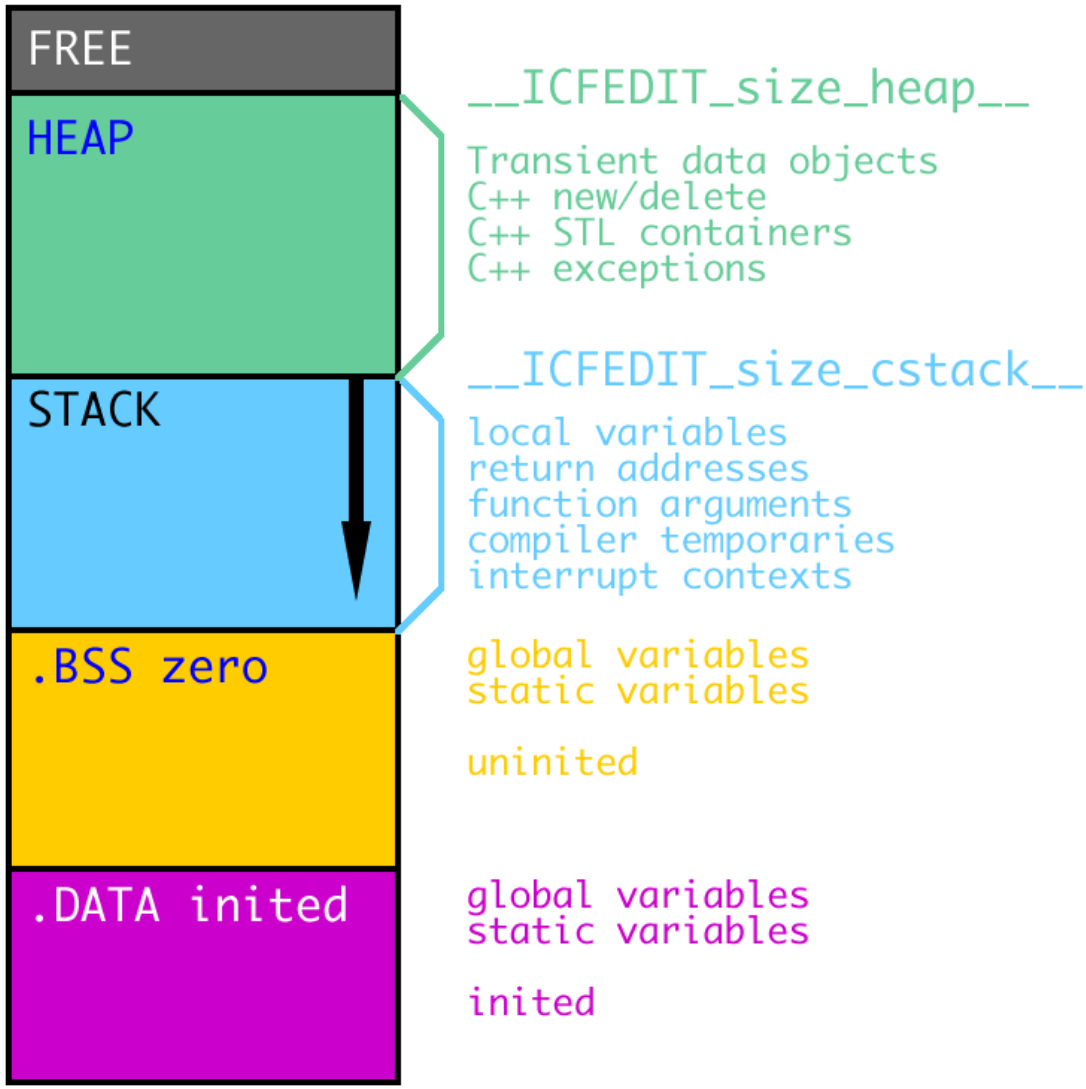
```
int soldiers[5];
int soldiers[5] = {1, 2, 3, 4, 5};
int soldiers[5] = {1, 2, 3};

// Correction: 1, 0, 0, 0, 0 (not 1, 1, 1, 1, 1)
int soldiers[5] = {1};

int *soldiers = malloc(sizeof(int) * n);

// VLA (Variable Length Array)
int soldiers[n];

// access
soldiers[2]      soldiers[i]
```

Memory Model

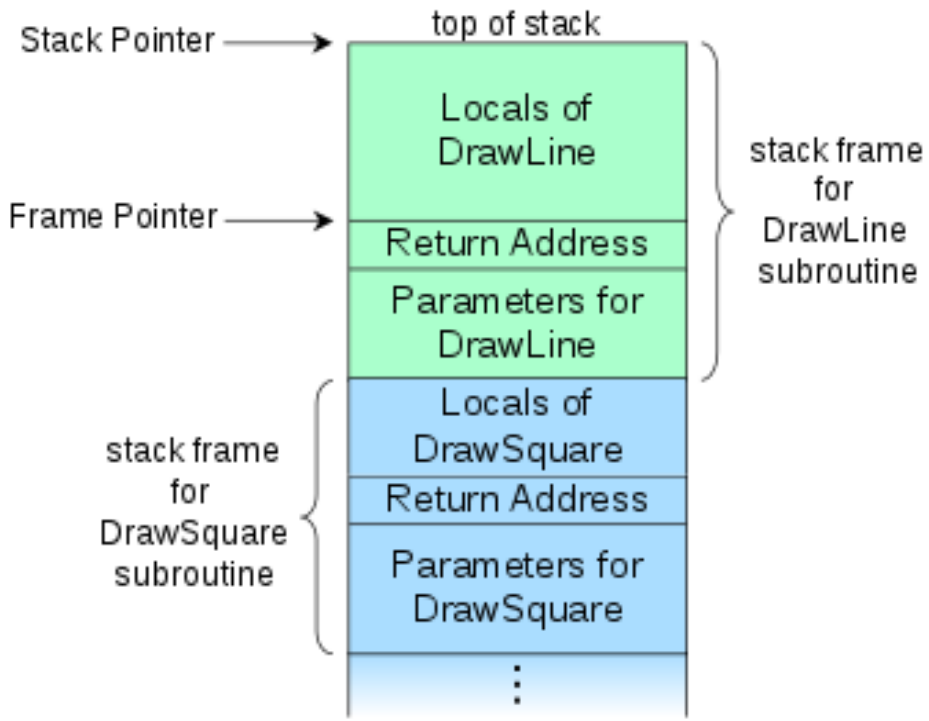
Definition (Memory (K&R))

The memory is organized as a collection of consecutively addressed cells that may be manipulated individually or in contiguous groups.




FREE
HEAP
STACK
.BSS zero
.DATA inited
__ICFEDIT_size_heap__
Transient data objects
C++ new/delete
C++ STL containers
C++ exceptions
__ICFEDIT_size_cstack__
local variables
return addresses
function arguments
compiler temporaries
interrupt contexts
global variables
static variables
uninited
global variables
static variables
inited

Types	Scopes	Lifttimes
Static/Global	The entire file	The lifetime of the program
Automatic		
Dynamic		




```
void DrawSquare(int len) {
  ...
  DrawLine(len, dir);
  ...
}
```

Pointers and Arrays

In C, there is a strong relationship between pointers and arrays, strong enough that pointers and arrays should be discussed simultaneously.

— *K&R*

Pointers

Definition (Pointers (K&R))

A pointer is a variable that contains the address of a variable.

```
int a = 0;
int *p = &a;
```

Definition (Pointers (K&R))

A pointer is a variable that contains the address of a variable.

```
int a = 0;
int *p = &a;
```

Definition (Pointers in Memory (K&R))

A pointer is a group of cells (often two or four) that can hold an address.

```
swap(a, b);

void swap(int a, int b) {
  int temp = a;
  a = b;
  b = tmp;
}
```

```
swap(a, b);

void swap(int a, int b) {
  int temp = a;
  a = b;
  b = tmp;
}
```

Pointer arguments enable a function to access and change objects in the function that called it. — *K&R*

```
swap(a, b);

void swap(int a, int b) {
  int temp = a;
  a = b;
  b = tmp;
}
```

Pointer arguments enable a function to access and change objects in the function that called it. — *K&R*

```
swap(&a, &b);

void swap(int *a, int *b) {
  int temp = *a;
  *a = *b;
  *b = tmp;
}
```

1D Arrays

Definition (Name of an Array)

The value of a variable of type array is the address of element zero of the array.

$$a \triangleq \&a[0]$$

```
int a[5];
a,    &a[0] // what are they?

int *pa = a;
int *pa = &a[0];

&a // what is this?
```

Definition (Name of an Array)

The value of a variable of type array is the address of element zero of the array.

$$a \triangleq \&a[0]$$

```
int a[5];
a,    &a[0] // what are they?

int *pa = a;
int *pa = &a[0];

&a // what is this?
```

array-1d.c

```
int a[5];

int *pa;
```

Definition (Equivalence between Accesses)

$$pa[i] \triangleq a[i] \triangleq *(a+i)$$

When an array name is passed to a function, what is passed is a pointer, the location of the initial element.

— *K&R*

```
void f(int a[5])
void f(int a[], int n);
void f(int *a, int n);

f(a, 5);
f(pa, 5);
```

2D Arrays

```
int a[3][5] = {
  {1,2,3,4,5},
  {6,7,8,9,10},
  {11,12,13}
};
```

```c
int a[3][5] = {
  {1,2,3,4,5},
  {6,7,8,9,10},
  {11,12,13}
};
```

Elements (of an 2D array) are stored by rows.

— *K&R*

array-2d.c (Part I)

In C, a 2D array is really a 1D array, each of whose elements is an array.

— *K&R*

In C, a 2D array is really a 1D array, each of whose elements is an array.

— *K&R*

```
a,    &a[0],    a[0],    &a[0][0],    &a
int (*pa)[5] = a; // a pointer to an array of 5
    integers
```

array-2d.c (Part II)

In C, a 2D array is really a 1D array, each of whose elements is an array.

— *K&R*

```
a,    &a[0],    a[0],    &a[0][0],    &a
int (*pa)[5] = a; // a pointer to an array of 5
    integers
```

array-2d.c (Part II)

```
a[i][j]
```

```c
void f(int a[3][5]);

void f(int a[][5], int m); // m rows

void f(int (*a)[5], int m);

f(a, 3);
f(pa, 3);
```

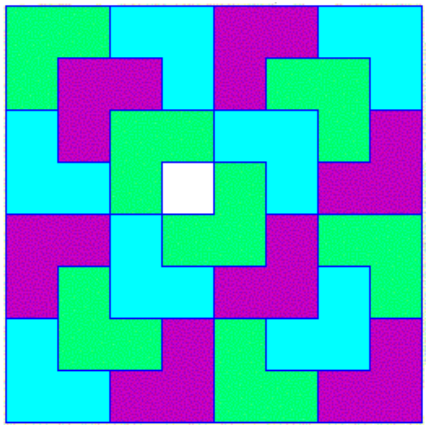
```
void f(int a[3][5]);

void f(int a[][5], int m); // m rows

void f(int (*a)[5], int m);

f(a, 3);
f(pa, 3);
```

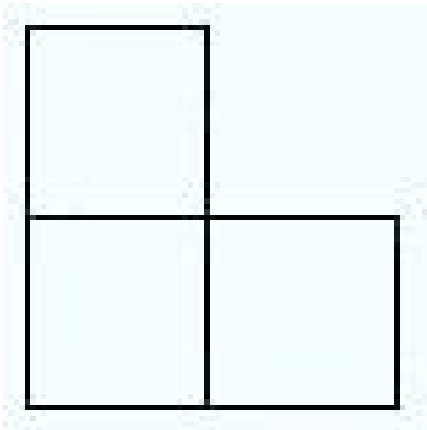
```
void f(int m, int n, int a[m][n]);

void f(int m, int n, int a[][n]);

void f(int m, int n, int (*a)[n]);

f(3, 5, a);
f(3, 5, pa);
```

The Tromino Tiling Puzzle

Theorem (Tromino Tiling Theorem)

For any positive integer k, a $2^k \times 2^k$ checkerboard with any one square removed can be tiled using right trominoes.





☞ Play with the Interactive Tromino Puzzle

tromino-tiling-vla.c


Thank
You!