Lecture 7

9 October 2018

Admin Matters
Unit 17: **Call by Reference**
Unit 18: **Heap**
Unit 19: ***n*D-Array**

Midterm
PE 1

(grading on-going)

PE1 stats

based on sample inputs/outputs

square

square

22 passes

digits

digits

100 passes

goldbach

goldbach

172 passes

newton

newton

145 passes

vote

vote

225 passes

Tutorial 7

Problem Sets from Units Today

Assignment 4

Released last Friday

(to be graded on correctness, style, documentation)

So Far

Problem Solving

decomposition
recursion
flowchart
conditionals
assertion

loops
array/list

C language / syntax

types in C
functions in C
+ - * / %
if else
&& || !

for/while
do-while
arrays
& *

Behavioural / Mental model

machine code
data in memory
types
call stack
memory addr

Tools / Good Practice

clang
vim
bash
style

Today

Problem Solving

decomposition
recursion
flowchart
conditionals
assertion

loops
array/list

C language / syntax

types in C
functions in C
+ - * / %
if else
&& || !

for/while
do-while
arrays
& *
malloc/free

Behavioural / Mental model

machine code
data in memory
types
call stack
memory addr
call by value/reference
heap

Tools / Good Practice

clang
vim
bash
style
documentation

Documentation

AY18/19 Sem 1

```
bool is_weekday(long day);
```

what is day?
what does it do?

```
/**
 * Check if a given day is a weekday.
 *
 * @param[in] day The day of the week
 *                (1 for Monday, 7 for Sunday).
 *
* @return true if the day is a weekday, false
*         otherwise.
*/
```

@param[in]

@param[out]

@param[in,out]

@return

@pre
@post

```
/**
 * Check if a given day is a weekday.
 *
 * @param[in] day The day of the week
 *                (1 for Monday, 7 for Sunday).
* @pre day >= 1 && day <= 7
* @return true if the day is a weekday, false
*         otherwise.
*/
```

Previously, in CS1010..

& address of a variable

if x is a variable, then &x gives us the address of x. (where does x live?)

* variable at an address

if x is an address, then *x is the variable stored in that address.
(who lives in x?)

```c
int main()
{
  long x;
  long *ptr;
  ptr = &x;
  *ptr = 1;
}
```

Arrays

array decay

```
long a[10];
```

`a`

is equivalent to

`&a[0]`

```
long size = cs1010_read_long();
    :
long a[size];
```

variable-size array

```
long size = cs1010_read_long();
   :
long *a = cs1010_read_long_array(size);
```

variable-size array

Strings

A string is just an array of char terminated by ‘\0’

```
char str[7] = “hello!”;
```

```
char str[7] = { ‘h’, ‘e’, ‘l’,
      ‘l’, ‘o’, ‘!’, ’\0’}
```

Rule:

You MUST only read and write into memory allocated for you.

Sometimes you write into memory you do not own, and your code runs. It does not mean it is ok.

```
long array[4];
array[4] = 10;
```

```
long array[4];

    :
// { i >= 0 && i < 4 }
array[i] = 0;
    :
```

```
long array[10000];
```

My program crashed. So I make my array big enough. It does not crash any more. Yay!

```
long array[10000];
```

If you code is buggy, there will still be an input that is big enough that will crash your code. Your code is still wrong.

```
char *str = “hello!”;
str[5] = ‘.’;
```

```
long add(long a, long b)
{
  long sum;
  sum = a + b;
  return sum;
}

int main()
{
  long x = 1;
  long y;
  y = add(x, 10);
}
```

A function is a black box. Whatever happens in the function stays in the function.

```
long x = 1;
foo(x);
// { x == 1 }
```

Effect-free programming

Pure functions

```
void set_to_0s(long len, long a[len]) {
  for (long i = 0; i < len; i += 1) {
    a[i] = 0;
  }
}
```

```
long a[10];
a[0] = 1;
foo(a);
// { a[0] == ?? }
```

```
long a[10];
a[0] = 1;
foo(a);
// { a[0] == ?? }
```

Call by reference

Function with side effects is no longer a black box.

@param[in]
@param[out]
@param[in,out]

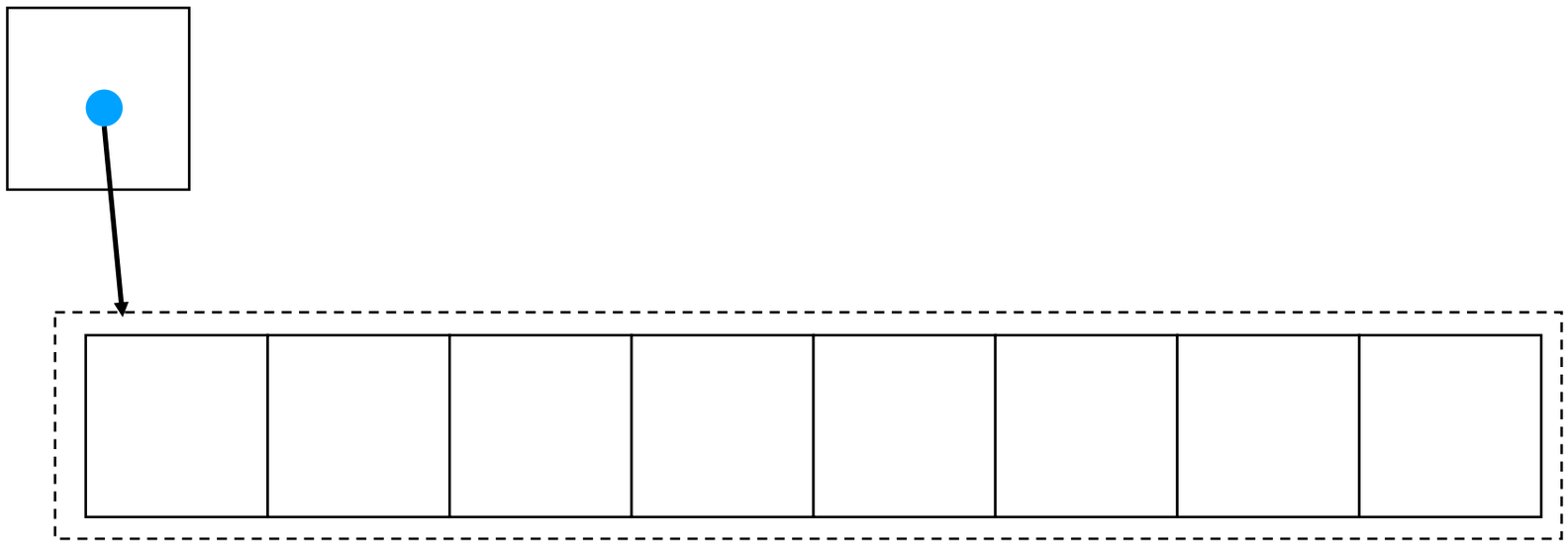
Heap

Global Variables

```
long x;

int main() {
  x = 1;
  foo();
  // { x == 1 ?? }
}
```

```
long (*matrix_row)[20];
```



```
long *(matrix_row[20]);
```