Computer Programming: Skills & Concepts (CP)
Variables and ints

C. Alexandru

25 September 2017

Week 1 Lectures

- Structure of the CP course
- What is programming?
  - Imperative Programming?
  - C programming?
- 'recipe' analogy to imperative programming.
- 'Hello World' in detail.

Today's lecture

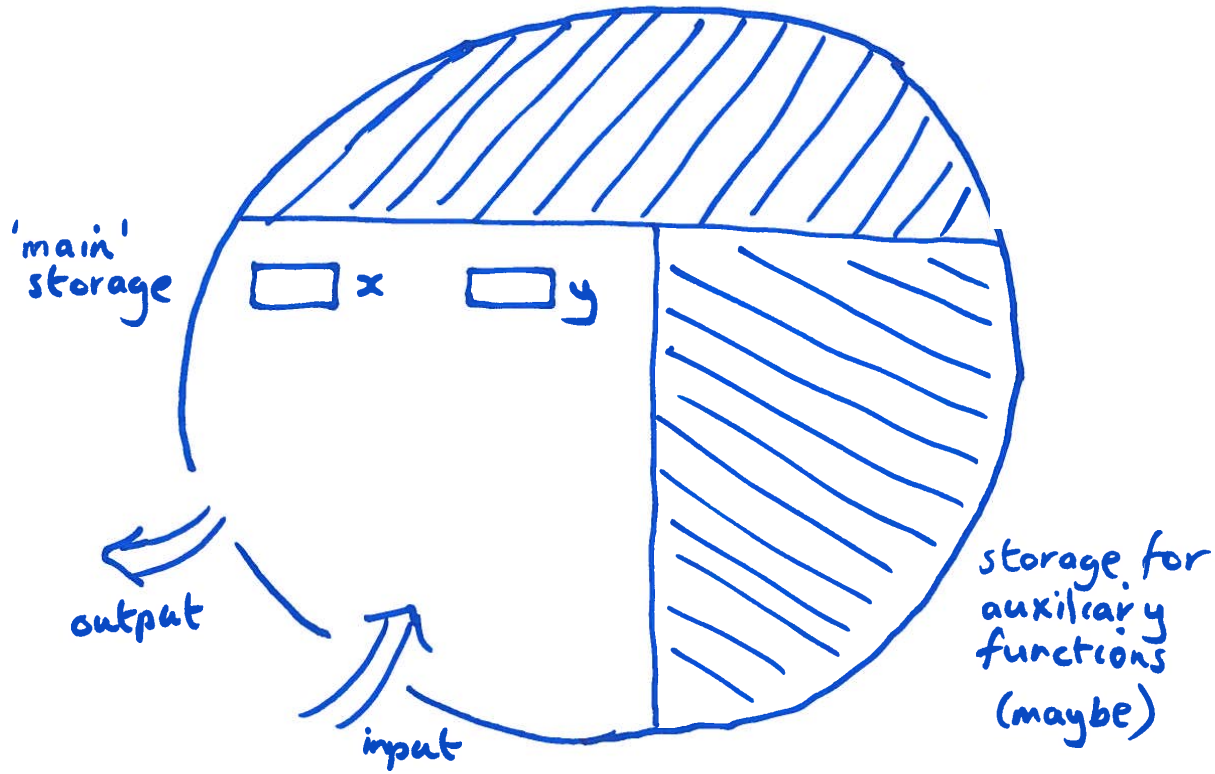
- Variables and change-of-state
- Our Squaring program
- Finding the maximum.
- Updating/Assigning variables

Variables

- For most computational tasks, the result (or output) will depend on some input
  - `hello.c` was unusual (no input, always does the same thing);
  - The `mypoem.c` program you write in your first lab is similarly unusual.
- If we read input, we need to store it somewhere to keep a record of its value.

  We store inputs in objects called Variables. The word 'Variable' means 'changing' – no fixed value; like a blackboard or slate where we can store one value, and change it later.

Program Environment in general



In this environment we have *two* variables `x` and `y`.
They are shown as being *un-initialized* (no particular value).

Variables and Imperative programming

- Imperative programming ie carried out as a sequence of step-by-step commands;
- Some commands will fetch input, some will send output, most commands will change the state of the program environment by changing a variable;
- Variables can appear in a few ways:
  - (i) At the start of a command, with an $=$, for example:
    `x = ...`
    This is an assignment command, used to *put a value* (whatever appears on the right) *into* the variable x.
  - (ii) Combined inside some other expression like:
    `...(x+2)*4...` OR `...5*x-3...`
    This is an evaluation of the expression, where the value of `x` is looked up and used in evaluation.
    - Evaluation in C is done as call-by-value (more later)

Defining variables

- Usually have one variable for each 'piece' of input.
- May have other variables besides the variables storing inputs.
  - May want/need to give names to (and store) important quantities;
  - May be helpful to store certain quantities for re-use;
  - As the computational problems get more complex/interesting, more necessity for interesting choice of variables.
- In C, and most other programming languages, variables must have a fixed type (more later ...).
- In C, a variable name starts with a letter, and can be made up of letters (a–z, A–Z), digits (0–9) and underscores (_).
  Actually, it can also start with _ , but by convention this is only done for 'private' variables that are internal to a particular code package.

Squaring a Number

input: Ask the user to input a (integer) number.

problem-solving: Compute the square of the integer.

output: Tell the user what the squared value is.

We will need at least one variable, to store the number input, because this value 'varies'. We will choose to have an extra variable to store the squared value.

Variables for `square.c`

Must declare variables in advance of using them.

```
int x, y;
```

Variable declaration is terminated by a semi-colon.

- Says "Make two integer variables `x` and `y` available for computation (in the program environment)."
- Called a *declaration* (*not* a command).
- Declaration must come before the variable is ever used in a command (and preferably just inside the start of `main`)

Input with `scanf`

`scanf` is the twin of `printf`. Reads numbers from input and stores them in variables.

But `scanf` requires a "`&`" before its arguments.
(Explanation later in the course. . . for now it's just magic.)

For example:

```
int x, y;
scanf("%d", &x);
```

Doing the Squaring

- The `scanf("%d", &x);` statement will *read the user-provided value into the variable* `x`.
- We need to *compute the value of this number squared*.
- The *expression* `x*x` represents the *value* of `x` squared.
- We can *store this squared value* in the variable `y` using an *assignment command*

  ```
  y = x*x;
  ```

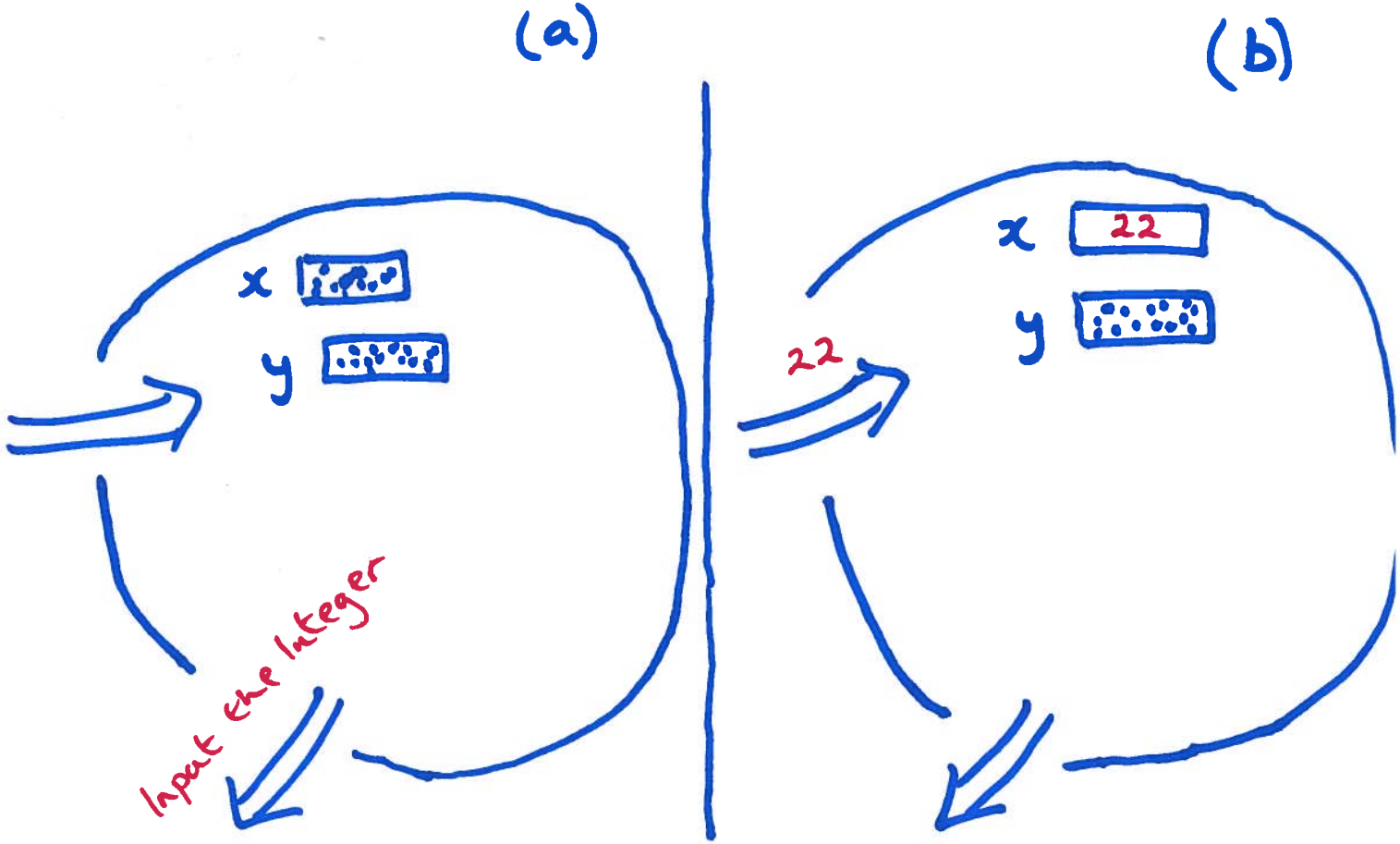
- Then squared value is ready to print out.

square.c

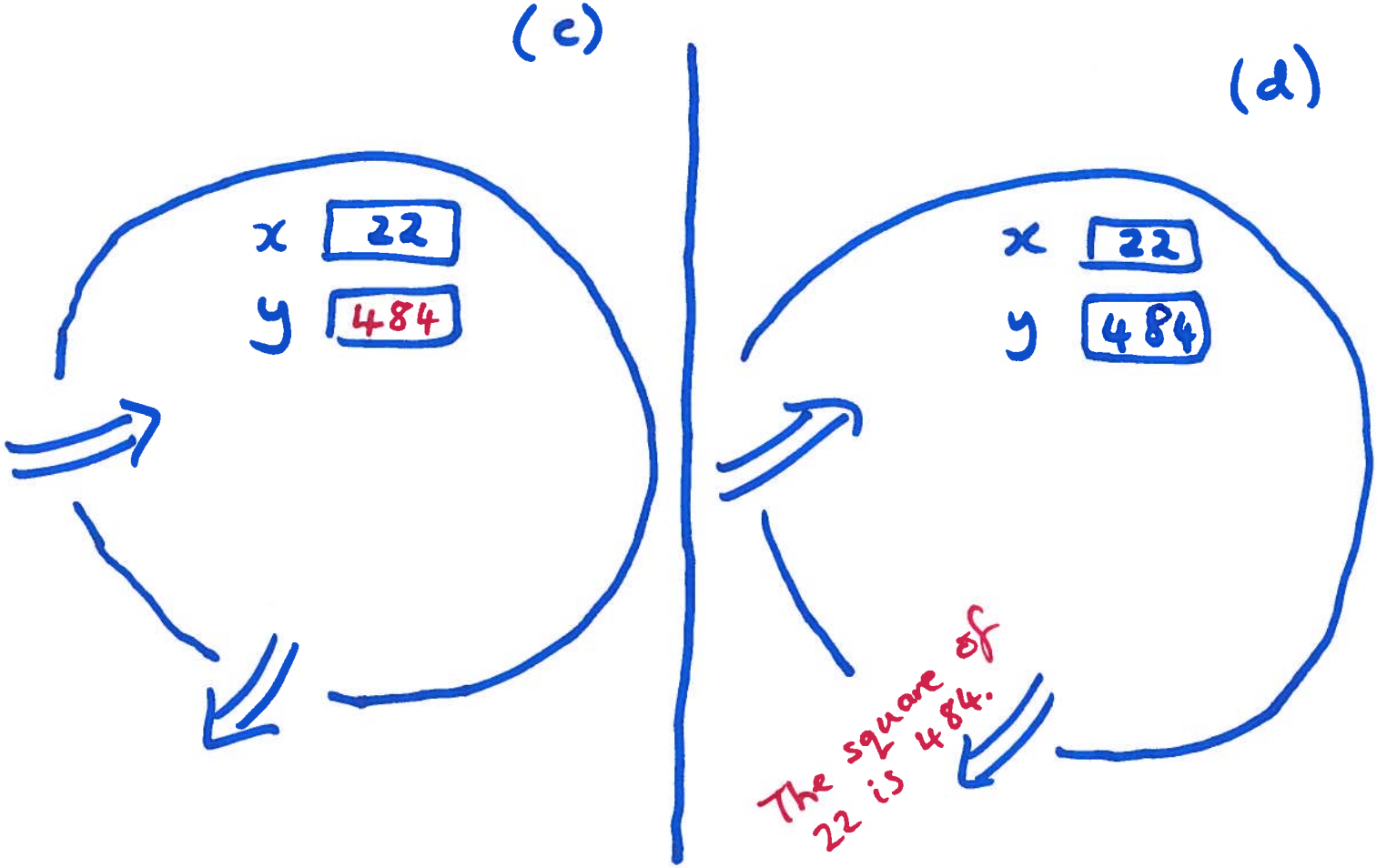
```c
#include <stdlib.h>
#include <stdio.h>

int main(void) {
  int x;
  int y;
  printf("Input the integer: ");
  scanf("%d", &x);
  y = x*x;
  printf("The square of %d is %d.\n", x, y);
  return EXIT_SUCCESS;
}
```

Program Environment for `square.c`



Program Environment for `square.c` cont.



Finding the Maximum

input: Ask the user for two integers, one at a time.

problem-solving: Find the larger of the two integers.

output: Output the larger of the two.

We will need at least two variables, to store the inputs.

Variables and `scanf` for Maximum

Declare variables in advance of using them.

```
int x, y, m;
```

- Says "Make three integer variables `x`, `y` and `m` available for computation (in the program environment)."

Now we can store values in them.

```
int x, y, m;
scanf("%d", &x);
scanf("%d", &y);
```

MAX of two integer variables

```
if (x > y) {
  m = x;
} else {
  m = y;
}
printf("MAX is %d: ", m);
```

- `(x > y)` is the condition to be evaluated. It evaluates to True only if `x` is larger than `y`.
- *where did we get the values* `x` *and* `y`?

max.c

```c
#include <stdlib.h>
#include <stdio.h>

int main(void) {
  int x, y, m;
   printf("Input the two integers: ");
   scanf("%d", &x);
   scanf("%d", &y);
   if (x > y) {
     m = x;
   } else {
     m = y;
   }
   printf("MAX is %d: ", m);
   return EXIT_SUCCESS;
}
```

Variables in C

Variables are "boxes" to store a value

- A C variable holds a single value;
- *Have to define what type of item a variable will hold*, eg: `int x;` or maybe `int x = 2;`
- In C, the value can change over time as a result of *program statements* which act on the variable, eg: `x = x + 1;`

Updating Variables

```
int n;                  ← n is declared – but not given a value!

n = 2 * n;              ← n is doubled (from what? ERROR)
n = 9;                  ← n gets the value 9
n = n + 1;              ← n gets the value 9+1, i.e. 10
n = 22 * n + 1;         ← n gets the value ?
```

The Assignment Statement

A variable is updated by an *assignment statement*

```
n = 22 * n + 1;
```

The left-hand side `n` is the variable being updated.
The right-hand side `22 * n + 1` is an *expression* for the new value.
First compute the expression, *then* change the variable to the new value.

Reading Assignment

Note: Reading assignments point out the sections in our recommended book that discuss what we've done in the lecture. If you're using a different book, then find the appropriate section in it. If you're not using a book, and want more info, google some keywords!

Read sections 1.1, 1.2, 1.3, 1.5 and 1.6 (all in Chapter 1) of "A Book on C" (skip 1.4).