Programming Refresher Workshop

Session 2

Mr Aaron Tan

Contents

- Learning journal
- Greatest Common Divisor
- Subprogram
- Parameters
- Pre- and post-conditions
- Address Parameters
- Testing

Learning Journal

- For your own self evaluation

School of Computing

Refresher in Programming Workshop 2017
(http://www.comp.nus.edu.sg/~tantc/refresher/)

My name: _____

This journal is for you to fill in the grades you've obtained for the exercises. This is to allow you to gauge your programming proficiency, and to provide more information for you to decide if you would like to take the CS1010 module notwithstanding your exemption, in order to better prepare yourself for the follow-up module (CS1020 for IS/BZA, CS2040+CS2030 for CS, or CS2040C for InfoSec/CEG). This is only for your own record; you do not need to submit it.

As a rough guide, if you have obtained, on average, a grade 'B' or better for the exercises given out, you are quite ready to tackle the follow-up module(s) of CS1010. Otherwise, we would advise you to give due consideration on taking CS1010.

Note that CodeCrunch awards grades based 100% on the correctness of your programs. In CS1010, correctness constitutes only 60%, with style (such as proper variable names, indentation, appropriate comments, etc.) and design (such as neat logic, appropriate algorithm and data structures, etc.) occupying the remaining 40%, and these are graded by the tutors. Hence, the actual grade you obtain for your program might be worse that what you have got from CodeCrunch in this workshop.

If you are not using CodeCrunch, the input files and output files for all exercises are available at http://www.comp.nus.edu.sg/~tantc/refresher/ for your checking. You may use the following grading scheme:

Grade	Description
A	If you get right answers for at least 90% of test cases
B	If you get right answers for at least 75% but less than 90% of test cases
C	If you get right answers for at least 60% but less than 75% of test cases

Sessions 1 and 2	My grades
Ex1: Mind Reading	
Ex2: Body Mass Index	
Ex3: Perfect Square	
Ex4: Up-slopes	
Ex5: Candles	
Ex6: Bisection Method	

My strengths

My weaknesses

Greatest Common Divisor (GCD) (1/3)

- GCD(a, b) = largest integer that divides both a and b
  - Eg: GCD(20,15) = 5; GCD(16, 21) = 1; GCD(7, 7) = 7; GCD(0, 123) = 123
- Conditions:
  - Both a and b are non-negative
  - a and b cannot be both zero

Greatest Common Divisor (GCD) (2/3)

- Algorithm

gcd(a, b):

1. If (b = 0) then the answer is a. Stop.
2. $r \leftarrow$ remainder of a / b
   $a \leftarrow b$
   $b \leftarrow r$
3. Go to step 1.

Greatest Common Divisor (GCD) (3/3)

- Pseudo-code

```
int a, b, r;
// Read input values a and b

while (b > 0) {
    r = a % b;
    a = b;
    b = r;
}

// print a
```

- C program (gcd_v1.c)

```
#include <stdio.h>

int main(void) {
   int a, b, r;

   printf("Enter a and b: ");
   scanf("%d %d", &a, &b);

   while (b > 0) {
      r = a % b;
      a = b;
      b = r;
   }

   printf("Answer = %d\n", a);
   return 0;
}
```

Subprogram (1/3)

- A subprogram (or subroutine)
  - is a (usually small) code that does a specific task;
  - can be called many times – reusability;
  - Promotes top-down design;
  - eases maintenance and debugging
- Named differently in different languages
  - procedure (in Pascal), function (in C), method (in Java), etc.

Subprogram (2/3)

- Non-modular (gcd_v1.c)

```
#include <stdio.h>

int main(void) {
   int a, b, r;

   printf("Enter a and b: ");
   scanf("%d %d", &a, &b);

   while (b > 0) {
      r = a % b;
      a = b;
      b = r;
   }

   printf("Answer = %d\n", a);
   return 0;
}
```

- Modular (gcd_v2.c)

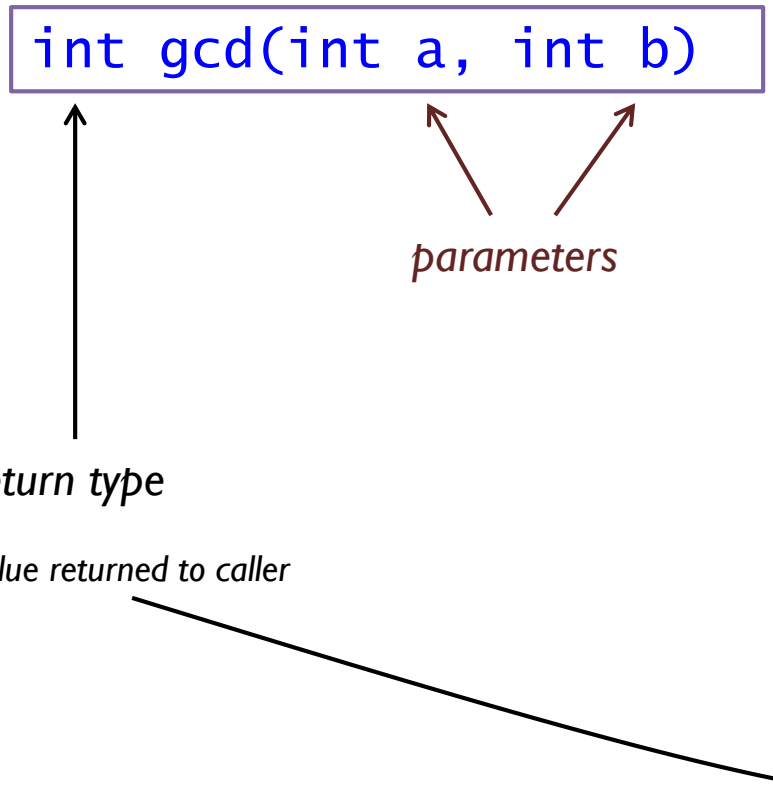
```
#include <stdio.h>
int gcd(int, int);

int main(void) {
   int a, b;
   printf("Enter a and b: ");
   scanf("%d %d", &a, &b);
   printf("GCD(%d,%d) = %d\n", a, b, gcd(a,b));
   return 0;
}

// Returns the greatest common divisor of a and b
// Pre-cond: a and b non-negative, and not both 0
int gcd(int a, int b) {
   int r;
   while (b > 0) {
      r = a % b;
      a = b;
      b = r;
   }
   return a;
}
```

Subprogram (3/3)

Function header defines contract with caller



- Modular (gcd_v2.c)

```
#include <stdio.h>
int gcd(int, int);

int main(void) {
   int a, b;
   printf("Enter a and b: ");
   scanf("%d %d", &a, &b);
   printf("GCD(%d,%d) = %d\n", a, b, gcd(a,b));
   return 0;
}

// Returns the greatest common divisor of a and b
// Pre-cond: a and b non-negative, and not both 0
int gcd(int a, int b) {
   int r;
   while (b > 0) {
      r = a % b;
      a = b;
      b = r;
   }
   return a;
}
```

Call

Local variable

Function definition

Parameters (1/2)

- Parameters
  - local to the function
  - pass-by-value
  - may (usually) have names different from variables in caller
- Function header specifies what the function does, and interface with caller
- Function body shows how it is done
  - could be replaced by a better/alternative code/algorithm

```
#include <stdio.h>
int gcd(int, int);

int main(void) {
   int a, b;
   printf("Enter a and b: ");
   scanf("%d %d", &a, &b);
   printf("GCD(%d,%d) = %d\n", a, b, gcd(a,b));
   return 0;
}

// Returns the greatest common divisor of x and y
// Pre-cond: x and y non-negative, and not both 0
int gcd(int x, int y) {
   int r;
   while (y > 0) {
      r = x % y;
      x = y;
      y = r;
   }
   return x;
}
```

Parameters (2/2)

- Parameters are data the subprogram needs from caller to perform its work
  - Only include the necessary

```
int main(void) {
  int a, b, rem;
  . . .
  printf("GCD(%d,%d) = %d\n", a, b,
         gcd(a,b,rem));
}

int gcd(int x, int y, int r) {
  while (y > 0) {
    r = x % y;
    x = y;
    y = r;
  }
  return x;
}
```

Pre-conditions

- To indicate the conditions that must be satisfied for the subprogram to be effective
  - In our GCD function, *a* and *b* must be non-negative, and not both 0 for the function to work correctly
- It is the responsibility of the caller to ensure that the pre-conditions of the subprogram are met before calling the subprogram

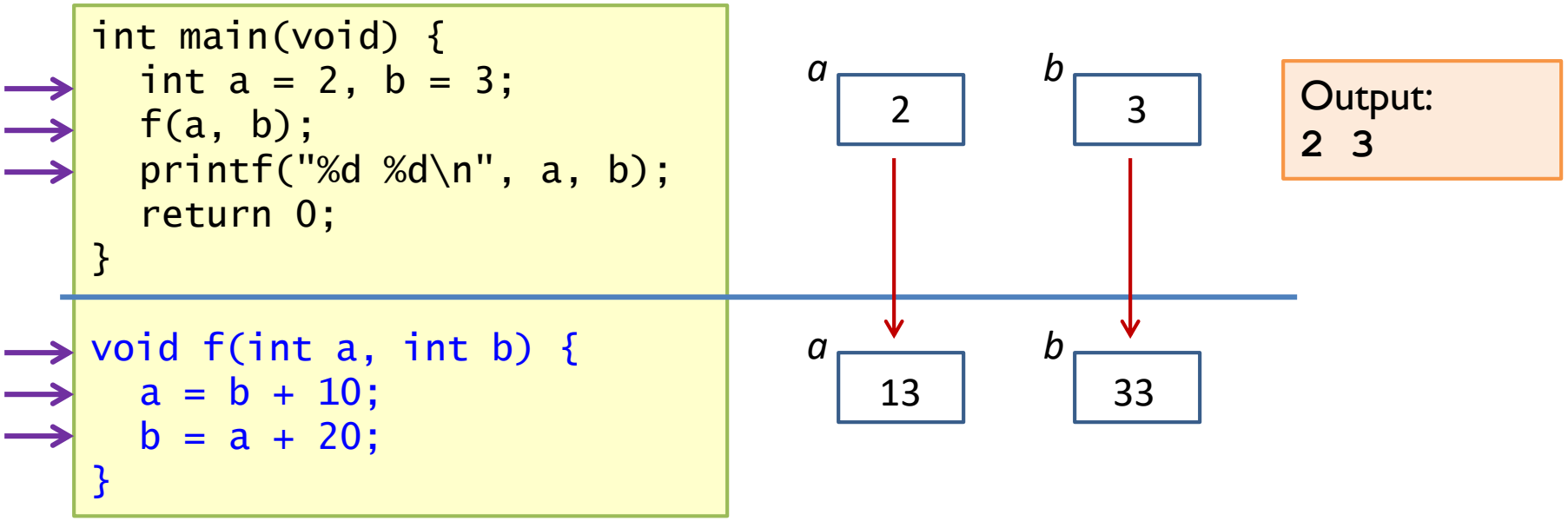
```
// Returns the greatest common divisor of a and b
// Pre-cond: a and b non-negative, and not both 0
int gcd(int a, int b) {
  int r;
  while (b > 0) {
    r = a % b;
    a = b;
    b = r;
  }
  return a;
}
```

Post-conditions

- A post-condition is a condition that must always be true just after the execution of that section of code
  - Sometimes tested using assertions within the code
  - Often, post-conditions are simply included in the documentation of that section of code
  - In our GCD function, *a* and *b* must be non-negative, and not both 0 for the function to work correctly

Address Parameters (1/2)

- With pass-by-value, no change in values of variables in caller

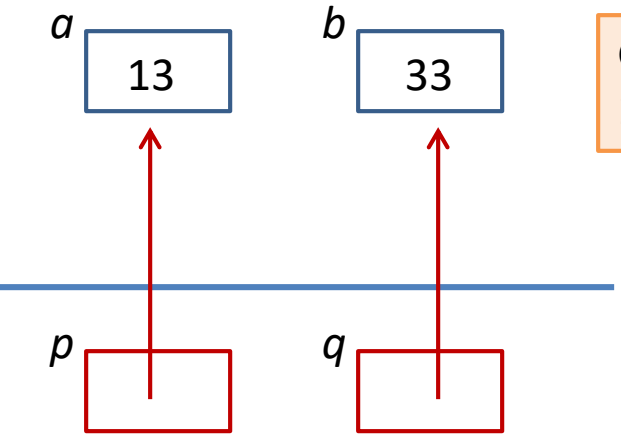


Address Parameters (2/2)

- To change values of variables in caller, pass in addresses of variables

```
int main(void) {
   int a = 2, b = 3;
   g(&a, &b);
   printf("%d %d\n", a, b);
   return 0;
}

void g(int *p, int *q) {
   *p = *q + 10;
   *q = *p + 20;
}
```



Output:
13 33

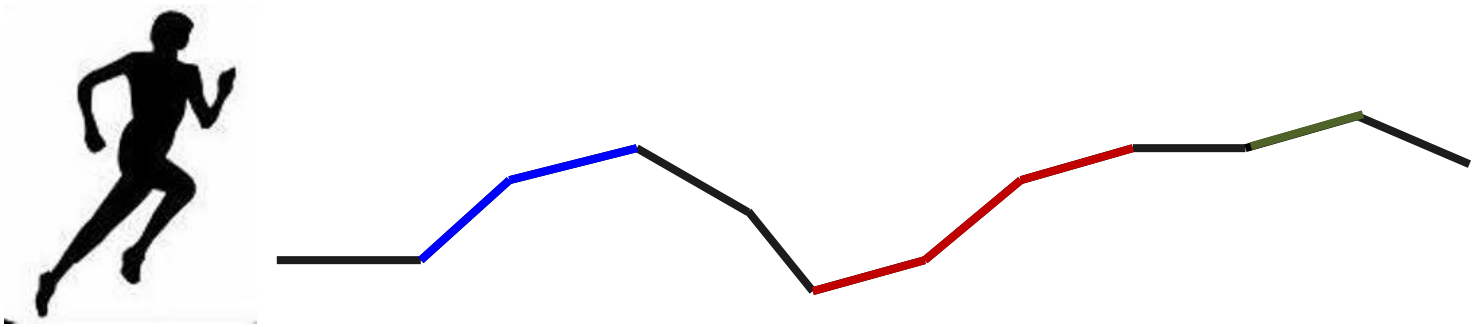
- For Java, C, C++
  - &a: address of variable a
  - *p: variable pointed to by p

Testing

- It is your responsibility to test your program thoroughly before submission.
- Do not use CodeCrunch to test your program.
- Test different cases
  - For GCD, these could be some cases to test:
    - Cases where $a < b$. Eg: gcd(12, 21), gcd(20, 30)
    - Cases where $a = b$. Eg: gcd(71, 71), gcd (6, 6)
    - Cases where a and b have common factor larger than 1. Eg: gcd(14, 8), gcd(35, 77)
    - Cases where a and b have no common factor larger than 1. Eg: gcd(10,27), gcd(123, 35)
    - Cases where a or b is 0. Eg: gcd(0, 7), gcd(8, 0)

Day 1 Exercise 4: Up-slopes

- Count the number of up-slopes



Up-slopes: Test Cases

Examples will be given during lecture.

The End