

Last Name:

First Name:

Computer Science C.Sc. 342

Take Home TEST *CSc or CPE*

Submit by to TA by 12:00 PM, May 1, 2023

Objective:

The objective of this take-home test is to demonstrate understanding how recursive function calls allocate memory, how recursive functions are executed, how control is transferred from one level to the next, how parameters are transferred from one level to the next.

1. First, study the textbook section 2.8 “Supporting procedures in computer hardware”.
2. Second, Implement the tutorial example on factorial recursive function calls as shown below:
 - 2.1.1 Run and debug a recursive function calls on three different platforms:
 - a. x86 Intel using Microsoft's Visual Studio,
 - b. MIPS on MARS Simulator, and
 - c. 64-bit I7 (I5 or I3) processor running Linux (or MAC with I7 or M1 processor).
 - d. Display and explain all frames on stack.
3.
 - a. Measure and plot the time it takes to compute Factorial (N), for N= 10, 100, 1000, 10,000.
 - b. Repeat tutorial example 1 and 2 to compute ***GCD(a,b) using recursive version of EUCLIDEAN algorithm for two integers $a>0, b>0$. To refresh GCD(a,b) computation please refer to last 3 pages of this assignment.***
4. **What to Submit:** report, working project files and how to use it.

! " # \$ % & ' () * & + , . ') (of a recursive procedure that calculates the factorial of a number and its code in both C and MIPS can be found in the textbook section 2.8 Implement and is shown below.

Create and explain Stack Frames for the recursive function call factorial(5)

```
int factorial (int N)
{
if (N==1)
return 1;
return (N*factorial(N-1));
}
void main()
{
int N_fact=factorial(5);
}
```

1.! Compile and run this program in Debug mode in .NET environment.

Last Name:

First Name:

For each `!''###$#%&%%#` display Frame on stack and write down the address on stack and value of

- Argument at current level
- local variable (if any) at current level
- return address at current level
- EIP
- EBP
- ESP

You may use arrow to point a specific location on stack frame.

At the end of calls you should display 5 frames on the stack as shown in FIGURE 1.

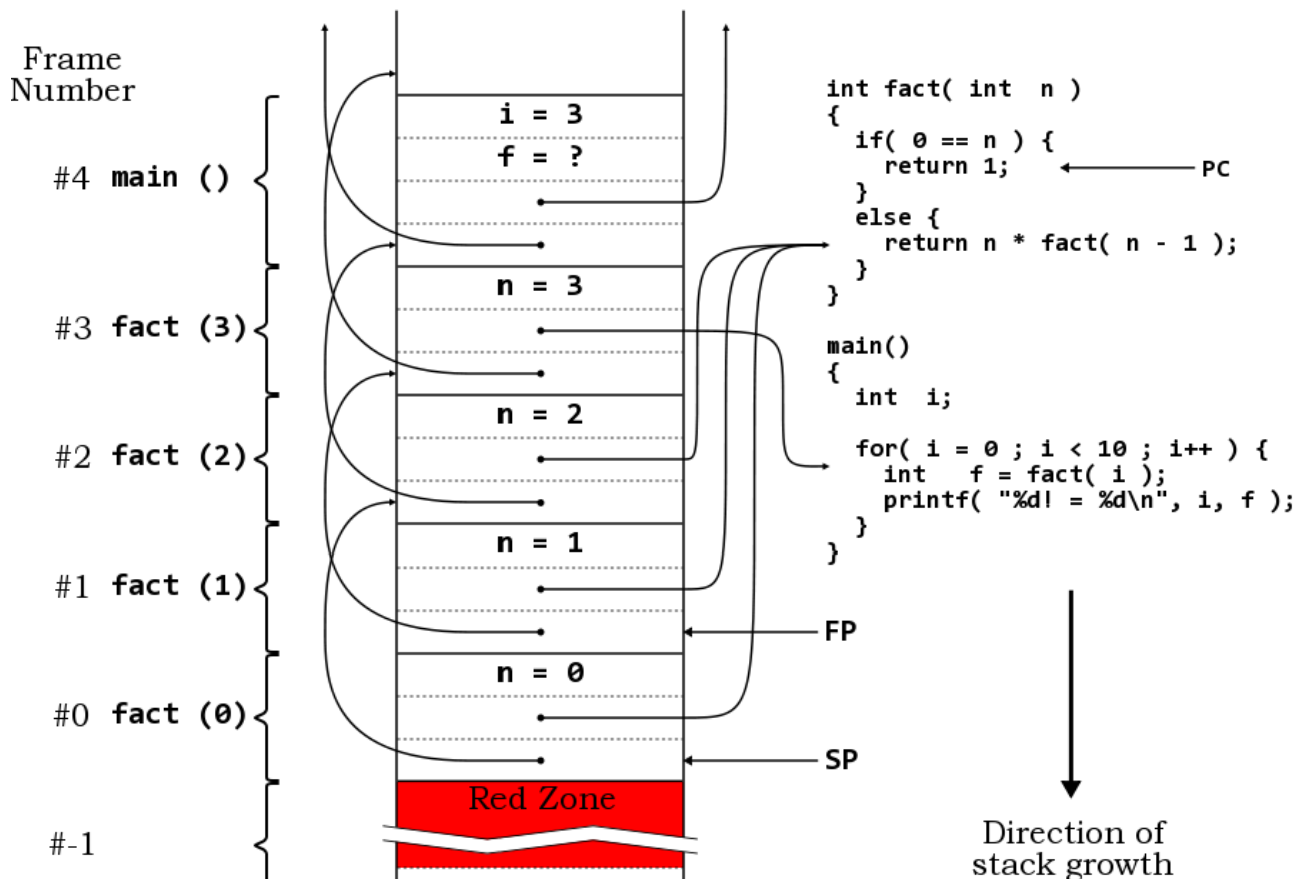


FIGURE 1. All arrows have to show labels to addresses on stack and corresponding values.

Please explain the return process – specify instructions and arguments used at each nested level when returning.

- 2.! (Optional) Create a lean version of the factorial() function. Instead of using CALL instruction (generated by compiler), create function call using similar to JAL instruction in MIPS - save the return address and then jump to function. Do not push and pop unnecessary information on stack (such as registers ebx, ecx, etc.) on stack.

Last Name:

First Name:

- 3.! Please repeat Section 1 using MIPS instructions and run the program on a simulator MARS. You can use example described in the section on nested procedure calls in the textbook.
- 4.! Please repeat Section 1 using GCC, GDB in LINUX environment, and run the program in command mode using GDB. You can use example described in the section on nested procedure calls in the textbook.

Sample screenshots for X86, MS Visual Studio in Debug mode

!!

```
1: int factorial(int N){
004013C0 55      push     ebp
004013C1 8B EC    mov     ebp,esp
004013C3 81 EC C0 00 00 00 sub     esp,0C0h
004013C9 53      push     ebx
004013CA 56      push     esi
004013CB 57      push     edi
004013CC 8D BD 40 FF FF FF lea     edi,[ebp-0C0h]
004013D2 B9 30 00 00 00 mov     ecx,30h
004013D7 B8 CC CC CC CC mov     eax,0CCCCCCCCh
004013DC F3 AB    rep stos dword ptr es:[edi]
2:      if (N == 1) return 1;
004013DE 83 7D 08 01 cmp     dword ptr [N],1
004013E2 75 07    jne     factorial+2Bh (004013EBh)
004013E4 B8 01 00 00 00 mov     eax,1
004013E9 EB 13    jmp     factorial+3Eh (004013FEh)
3:      return N*factorial(N - 1);
004013EB 8B 45 08 mov     eax,dword ptr [N]
004013EE 83 E8 01 sub     eax,1
004013F1 50      push     eax
004013F2 E8 E4 FD FF FF call    factorial (004011DBh)
004013F7 83 C4 04 add     esp,4
004013FA 0F AF 45 08 imul    eax,dword ptr [N]
4:      }
004013FE 5F      pop     edi
4:      }
004013FF 5E      pop     esi
00401400 5B      pop     ebx
EAX = CCCCCC
EBX = 7EFDE000
ECX = 00000000
EDX = 00000001
ESI = 00000000
EDI = 0015FAC4
EIP = 004013DE
ESP = 0015F9F8
EBP = 0015FAC4
EFL = 00000200
0x0015facc = 00000005
Saved EBP of main()
Return address during execution of factorial(5)
Argument during execution of factorial(5)
```

!

Last Name:

First Name:

```
Memory 1
Address: 0x0015F61C
0x0015F61C 00 00 47 00 00 00 00 00 7f 00 00 00 14 f7 15 00 5a 3c be 77 8b 00 00 00 9c b1 c3 77
0x0015F638 14 f7 15 00 ce 57 c0 77 d3 3c be 77 cf c0 b9 75 40 04 00 00 00 00 00 00 00 47 00
0x0015F654 50 01 47 00 50 01 47 00 64 f7 15 00 b8 50 47 00 00 10 00 00 64 f7 15 00 01 00 00 00
0x0015F670 77 01 00 00 04 fe 69 0f 50 01 47 00 fe ff ff ff 00 88 47 00 cf ee 5b 0f 00 00 56 0f
0x0015F68C 01 00 00 00 7f 00 00 00 b4 f6 15 00 3c f8 15 00 00 00 00 00 00 e0 fd 7e cc cc cc cc
0x0015F6A8 cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
0x0015F6C4 cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
0x0015F6E0 cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
0x0015F6FC cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
0x0015F718 cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
0x0015F734 cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
0x0015F750 cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
0x0015F76C 01 00 00 00 14 f9 15 00 00 00 00 00 e0 fd 7e cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
0x0015F788 cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
0x0015F7A4 cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
0x0015F7C0 cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
0x0015F7DC cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
0x0015F7F8 cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
0x0015F814 cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
0x0015F830 cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
0x0015F84C 00 00 00 00 00 e0 fd 7e cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
0x0015F868 cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
0x0015F884 cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
0x0015F8A0 cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
0x0015F8BC cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
0x0015F8D8 cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
0x0015F8F4 cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
0x0015F910 cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
0x0015F92C cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
0x0015F948 cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
0x0015F964 cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
0x0015F980 cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
0x0015F99C cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
0x0015F9B8 cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
0x0015F9D4 cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
0x0015F9F0 f7 13 40 00 04 00 00 00 a8 fb 15 00 00 00 00 00 e0 fd 7e cc cc cc cc cc cc cc cc cc
0x0015FA0C cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
0x0015FA28 cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
0x0015FA44 cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
0x0015FA60 cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
0x0015FA7C cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
0x0015FA98 cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
0x0015FAB4 cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
0x0015FAB8 cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc cc
```

Sample screenshots for MIPS, Simulator MARS environment

Last Name:

First Name:

Text Segment

Bkpt	Address	Code	Basic	Source
	0x00400000	0x24040005	addiu \$4,\$0,0x00000005	2: li \$a0, 5
	0x00400004	0x0c100004	jal 0x00400010	3: jal fact
	0x00400008	0x20500000	addi \$16,\$2,0x00000000	4: addi \$a0, \$v0, 0
	0x0040000c	0x0000000c	syscall	5: syscall
	0x00400010	0x23bdf000	addi \$29,\$29,0xffffffff	8: addi \$sp, \$sp, -8
	0x00400014	0xafbf0004	sw \$31,0x00000004(\$29)	9: sw \$ra, 4(\$sp)
	0x00400018	0xafaf0000	sw \$4,0x00000000(\$29)	10: sw \$a0, 0(\$sp)
	0x0040001c	0x28880001	slti \$8,\$4,0x00000001	12: slti \$t0, \$a0, 1
	0x00400020	0x11000003	beq \$8,\$0,0x00000003	13: beq \$t0, \$zero, L1
	0x00400024	0x20020001	addi \$2,\$0,0x00000001	15: addi \$v0, \$zero, 1
	0x00400028	0x23bd0008	addi \$29,\$29,0x00000008	16: addi \$sp, \$sp, 8
	0x0040002c	0x03e00008	jr \$31	17: jr \$ra
	0x00400030	0x2084ffff	addi \$4,\$4,0xffffffff	19: Li: addi \$a0, \$a0, -1
	0x00400034	0x0c100004	jal 0x00400010	20: jal fact
	0x00400038	0x8fa40000	lw \$4,0x00000000(\$29)	22: lw \$a0, 0(\$sp)
	0x0040003c	0x8fbf0004	lw \$31,0x00000004(\$29)	23: lw \$ra, 4(\$sp)
	0x00400040	0x23bd0008	addi \$29,\$29,0x00000008	24: addi \$sp, \$sp, 8
	0x00400044	0x70821002	mul \$2,\$4,\$2	25: mul \$v0, \$a0, \$v0
	0x00400048	0x03e00008	jr \$31	26: jr \$ra

Labels

Label	Address
fact.asm	
main	0x00400000
fact	0x00400010
L1	0x00400030

Registers

Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x00000000
\$v0	2	0x00000000
\$v1	3	0x00000000
\$a0	4	0x00000004
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x00000000
\$t1	9	0x00000000
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x00000000
\$s1	17	0x00000000
\$s2	18	0x00000000
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$s8	24	0x00000000
\$s9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10008000
\$fp	29	0xfffff4
\$sp	30	0x00000000
\$ra	31	0x00400038
\$pc		0x00400010
\$hi		0x00000000
\$lo		0x00000000

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x7ffffef0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000005	0x00400008	0x00000000
0x7fffff00	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7fffff20	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7fffff40	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7fffff60	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7fffff80	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7fffffa0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7fffffc0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7fffffe0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7fffff100	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7fffff120	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x7ffffef0	0x00000000	0x00000000	0x00000000	0x00000004	0x00400038	0x00000005	0x00400008	0x00000000
0x7fffff00	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7fffff20	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7fffff40	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7fffff60	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7fffff80	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7fffffa0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7fffffc0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7fffffe0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7fffff100	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7fffff120	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7fffff140	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

Argument at current level address: 0x7ffefec value: 4

Return address on stack address: 0x7ffef0 value: 0x00400038

Last Name:

First Name:

Sample screenshots for 64 bit Intel processor, GDB

```
=> 0x0000000004004f6 <+0>:    push    %rbp
0x0000000004004f7 <+1>:    mov     %rsp,%rbp
0x0000000004004fa <+4>:    sub     $0x10,%rsp
0x0000000004004fe <+8>:    mov     %edi,-0x4(%rbp)
0x000000000400501 <+11>:   cmpl    $0x1,-0x4(%rbp)
0x000000000400505 <+15>:   jne     0x40050e <factorial(int)+24>
0x000000000400507 <+17>:   mov     $0x1,%eax
0x00000000040050c <+22>:   jmp     0x40051f <factorial(int)+41>
0x00000000040050e <+24>:   mov     -0x4(%rbp),%eax
0x000000000400511 <+27>:   sub     $0x1,%eax
0x000000000400514 <+30>:   mov     %eax,%edi
0x000000000400516 <+32>:   callq   0x4004f6 <factorial(int)>
0x00000000040051b <+37>:   imul    -0x4(%rbp),%eax
0x00000000040051f <+41>:   leaveq  %eax
0x000000000400520 <+42>:   retq
End of assembler dump.
(gdb) nexti 3
0x0000000004004fe      1      int factorial(int N){
1: x/i $pc
=> 0x4004fe <factorial(int)+8>: mov     %edi,-0x4(%rbp)
(gdb) printf "rbp:%x\nrsp:%x\n",$rbp,$rsp
rbp:ffffdde0
rsp:ffffddd0
(gdb)
```

```
1: x/i $pc
=> 0x4004fe <factorial(int)+8>: mov     %edi,-0x4(%rbp)
(gdb) printf "rbp:%x\nrsp:%x\n",$rbp,$rsp
rbp:ffffdde0
rsp:ffffddd0
(gdb) nexti
2      if(N == 1) return 1;
1: x/i $pc
=> 0x400501 <factorial(int)+11>   cmpl    $0x1,-0x4(%rbp)
(gdb) x/12xw $rsp
0x7fffffffddd0: 0x00000000 0x00000000 0x00000000 0x00000001
0x7fffffffddde: 0xffffde00 0x00007fff 0x0040051b 0x00000000
0x7fffffffddff: 0xffffde20 0x00007fff 0xffffde10 0x00000002
(gdb) p $rip
$15 = (void (*)(void)) 0x400501 <factorial(int)+11>
(gdb)
```

Argument during factorial(1)

Return address during factorial(1)

Saved RBP of factorial(2)

!"#\$%&'()*+,\$-./:\$

\$

\$

\$

\$

(01203\$45\$67#\$89: 4; 2<=>?\$

\$

3.2. The Euclidean Algorithm

3.2.1. The Division Algorithm. The following result is known as *The Division Algorithm*:¹ If $a, b \in \mathbb{Z}$, $b > 0$, then there exist unique $q, r \in \mathbb{Z}$ such that $a = qb + r$, $0 \leq r < b$. Here q is called *quotient* of the *integer division* of a by b , and r is called *remainder*.

3.2.2. Divisibility. Given two integers a, b , $b \neq 0$, we say that b *divides* a , written $b|a$, if there is some integer q such that $a = bq$:

$$b|a \Leftrightarrow \exists q, a = bq.$$

We also say that b *divides* or is a *divisor of* a , or that a is a *multiple* of b .

3.2.3. Prime Numbers. A *prime* number is an integer $p \geq 2$ whose only positive divisors are 1 and p . Any integer $n \geq 2$ that is not prime is called *composite*. A non-trivial divisor of $n \geq 2$ is a divisor d of n such that $1 < d < n$, so $n \geq 2$ is composite iff it has non-trivial divisors. *Warning*: 1 is not considered either prime or composite.

Some results about prime numbers:

1. For all $n \geq 2$ there is some prime p such that $p|n$.
2. (Euclid) There are infinitely many prime numbers.
3. If $p|ab$ then $p|a$ or $p|b$. More generally, if $p|a_1a_2 \dots a_n$ then $p|a_k$ for some $k = 1, 2, \dots, n$.

3.2.4. The Fundamental Theorem of Arithmetic. Every integer $n \geq 2$ can be written as a product of primes uniquely, up to the order of the primes.

It is customary to write the factorization in the following way:

$$n = p_1^{s_1} p_2^{s_2} \dots p_k^{s_k},$$

where all the exponents are positive and the primes are written so that $p_1 < p_2 < \dots < p_k$. For instance:

$$13104 = 2^4 \cdot 3^2 \cdot 7 \cdot 13.$$

¹The result is not really an “algorithm”, it is just a mathematical theorem. There are, however, algorithms that allow us to compute the quotient and the remainder in an integer division.

3.2.5. Greatest Common Divisor. A positive integer d is called a *common divisor* of the integers a and b , if d divides a and b . The greatest possible such d is called the *greatest common divisor* of a and b , denoted $\gcd(a, b)$. If $\gcd(a, b) = 1$ then a, b are called *relatively prime*.

Example: The set of positive divisors of 12 and 30 is $\{1, 2, 3, 6\}$. The greatest common divisor of 12 and 30 is $\gcd(12, 30) = 6$.

A few properties of divisors are the following. Let m, n, d be integers. Then:

1. If $d|m$ and $d|n$ then $d|(m + n)$.
2. If $d|m$ and $d|n$ then $d|(m - n)$.
3. If $d|m$ then $d|mn$.

Another important result is the following: Given integers a, b, c , the equation

$$ax + by = c$$

has integer solutions if and only if $\gcd(a, b)$ divides c . That is an example of a *Diophantine equation*. In general a Diophantine equation is an equation whose solutions must be integers.

Example: We have $\gcd(12, 30) = 6$, and in fact we can write $6 = 1 \cdot 30 - 2 \cdot 12$. The solution is not unique, for instance $6 = 3 \cdot 30 - 7 \cdot 12$.

3.2.6. Finding the gcd by Prime Factorization. We have that $\gcd(a, b)$ = product of the primes that occur in the prime factorizations of both a and b , raised to their lowest exponent. For instance $1440 = 2^5 \cdot 3^2 \cdot 5$, $1512 = 2^3 \cdot 3^3 \cdot 7$, hence $\gcd(1440, 1512) = 2^3 \cdot 3^2 = 72$.

Factoring numbers is not always a simple task, so finding the gcd by prime factorization might not be a most convenient way to do it, but there are other ways.

3.2.7. The Euclidean Algorithm. Now we examine an alternative method to compute the gcd of two given positive integers a, b . The method provides at the same time a solution to the Diophantine equation:

$$ax + by = \gcd(a, b).$$

It is based on the following fact: given two integers $a \geq 0$ and $b > 0$, and $r = a \bmod b$, then $\gcd(a, b) = \gcd(b, r)$. Proof: Divide a by

b obtaining a quotient q and a remainder r , then

$$a = bq + r, \quad 0 \leq r < b.$$

If d is a common divisor of a and b then it must be a divisor of $r = a - bq$. Conversely, if d is a common divisor of b and r then it must divide $a = bq + r$. So the set of common divisors of a and b and the set of common divisors of b and r are equal, and the greatest common divisor will be the same.

The Euclidean algorithm is as follows. First we divide a by b , obtaining a quotient q and a remainder r . Then we divide b by r , obtaining a new quotient q' and a remainder r' . Next we divide r by r' , which gives a quotient q'' and another remainder r'' . We continue dividing each remainder by the next one until obtaining a zero remainder, and which point we stop. The last non-zero remainder is the gcd.

Example: Assume that we wish to compute $\gcd(500, 222)$. Then we arrange the computations in the following way:

$$\begin{aligned} 500 &= 2 \cdot 222 + 56 &\rightarrow r &= 56 \\ 222 &= 3 \cdot 56 + 54 &\rightarrow r' &= 54 \\ 56 &= 1 \cdot 54 + 2 &\rightarrow r'' &= 2 \\ 54 &= 27 \cdot 2 + 0 &\rightarrow r''' &= 0 \end{aligned}$$

The last nonzero remainder is $r'' = 2$, hence $\gcd(500, 222) = 2$. Furthermore, if we want to express 2 as a linear combination of 500 and 222, we can do it by working backward:

$$\begin{aligned} 2 &= 56 - 1 \cdot 54 = 56 - 1 \cdot (222 - 3 \cdot 56) = 4 \cdot 56 - 1 \cdot 222 \\ &= 4 \cdot (500 - 2 \cdot 222) - 1 \cdot 222 = 4 \cdot 500 - 9 \cdot 222. \end{aligned}$$

The algorithm to compute the gcd can be written as follows:

```

1: procedure gcd(a,b)
2:   if a<b then   {make a the largest}
3:     swap(a,b)
4:   while b  $\neq$  0
5:     begin
6:       r := a mod b
7:       a := b
8:       b := r
9:     end
10:  return a
11: end gcd

```

The next one is a recursive version of the Euclidean algorithm:

```
1: procedure gcd_rekurs(a,b)
2:   if b=0 then
3:     return a
4:   else
5:     return gcd_rekurs(b,a mod b)
6: end gcd_rekurs
```