

Introduction to Programming (in C++)

Loops

Professors de Pro1

Dept. of Computer Science, UPC

- The `while` statement
- Applications: product, GCD, prime numbers
- The `for` statement
- Nested loops and advanced examples

The `while` Statement

The while Statement

Syntax

```
while (condition) statement;
```

- The condition must return a Boolean value.
- **Semantics:**
 - The condition is evaluated.
 - If true: the statement is executed and control returns to the while.
 - If false: the while terminates.

Write the numbers 1...N

```
int i = 1;
while (i <= N) {
    cout << i << endl;
    i = i + 1;
}
```

Product of Two Numbers

- Constraint: **do not** use the `*` operator.
- The algorithm calculates $x + x + x + \dots + x$ (y times).

```
// Input:  read two non-negative numbers x and y
// Output: write the product x*y
int main() {
    int x, y;
    cin >> x >> y;    // Let x=A, y=B
    int p = 0;
    // Invariant: A*B = p + x*y
    while (y > 0) {
        p = p + x;
        y = y - 1;
    }
    cout << p << endl;
}
```

A Quick Algorithm for the Product

Observation

- If y is **even**: $p = (x \cdot 2) \cdot (y/2)$
- If y is **odd**: $p = x \cdot (y - 1) + x$, and $(y - 1)$ becomes even

x	y	p
17	38	0
34	19	0
34	18	34
68	9	34
68	8	102
544	1	102
544	0	646

This mimics the binary paper-and-pencil multiplication method.

Euclid's Algorithm for GCD

Properties

- $\text{gcd}(a, a) = a$
- If $a > b$: $\text{gcd}(a, b) = \text{gcd}(a - b, b)$

```
while (a != b) {  
    if (a > b) a = a - b;  
    else b = b - a;  
}  
cout << a << endl;
```

Faster version

- $\text{gcd}(a, 0) = a$
- $\text{gcd}(a, b) = \text{gcd}(b, a \bmod b)$

```
// gcd(A,B) = gcd(a,b)  
while (b != 0) {  
    int r = a%b;  
    a = b;  
    b = r;  
}  
cout << a << endl;
```

Efficiency

Subtraction version: up to a iterations. Modulo version: $5 \log_{10}(b)$ iterations (Lamé, 1844).

Prime Numbers

```
// Input:  a natural number N > 0
// Output: write "is prime" or "is not prime"
int main() {
    int N;
    cin >> N;
    int divisor = 2;
    bool is_prime = (N != 1);
    while (is_prime and divisor*divisor <= N) {
        is_prime = N%divisor != 0;
        divisor = divisor + 1;
    }
    if (is_prime) cout << "is prime" << endl;
    else         cout << "is not prime" << endl;
}
```

Key optimisation

If N is not prime, one of its factors a satisfies $a \leq \sqrt{N}$.

So we only need to check divisors up to $\sqrt{N}$ (i.e. $\text{divisor} * \text{divisor} \leq N$).

Efficiency Matters

Slow version ($N = 2110454939$)

```
> time prime_slow < number  
is prime  
10.984u 0.004s 0:11.10
```

Fast version ($N = 2110454939$)

```
> time prime_fast < number  
is prime  
0.004u 0.000s 0:00.00
```

The optimised algorithm is about **2700**× faster!

The for Statement

The for Statement

while form

```
i = N;  
while (i <= M) {  
    do_something;  
    i = i + 1;  
}
```

for form

```
for (i = N; i <= M; i++) {  
    do_something;  
}
```

General equivalence

for (S_init; condition; S_iter) S_body; is equivalent to S_init; while
(condition) { S_body; S_iter; }

Examples with for

Write numbers in $[N, M]$

```
int N, M;  
cin >> N >> M;  
for (int i = N; i <= M; ++i) cout << i << endl;
```

Calculate x^y

```
int x, y;  
cin >> x >> y;  
int p = 1;  
for (int i = 0; i < y; ++i) p = p*x;  
cout << p << endl;
```

Drawing a Triangle

Output for $n = 6$

```
*  
**  
***  
****  
*****  
*****
```

```
// Input:  read a number n > 0  
// Output: write a triangle of size n  
int main() {  
    int n;  
    cin >> n;  
    // Inv: rows 1..i-1 have been written  
    for (int i = 1; i <= n; ++i) {  
        // Inv: '*' written j-1 times in row i  
        for (int j = 1; j <= i; ++j)  
            cout << '*';  
        cout << endl;  
    }  
}
```

Perfect Numbers

- A number $n > 0$ is **perfect** if it equals the sum of all its proper divisors.
 - Example: 6 is perfect ($1 + 2 + 3 = 6$)
 - Example: 12 is *not* perfect ($1 + 2 + 3 + 4 + 6 = 16 \neq 12$)

```
int n;
cin >> n;
int sum = 0, d = 1;
// Inv: sum is the sum of all divisors up to d-1
while (d <= n/2 and sum <= n) {
    if (n%d == 0) sum += d;
    d = d + 1;
}
if (sum == n) cout << "is perfect" << endl;
else        cout << "is not perfect" << endl;
```