Algorithms for Numbers and Public-key Cryptography

Part 3

Jean-Sébastien Coron

Université du Luxembourg

March 21, 2014

Summary

- C programming
  - Structures
- Algorithmic number theory.
  - Computing with large integers.

Structures in C

- Structures in C enable to group data of different type.
- Example 1: informations about someone
  - First name, last name, age.
- Example 2: a point in a plane
  - x and y coordinates.
- Exemple 3: a circle
  - Center and radius

Structures

- The `struct` keyword :
  - ```
    struct point {
      float x;
      float y;
    };
    ```
- This defines a new type: `struct point`.
- Each variable of this type has two fields :
  - `x` of type float
  - `y` of type float
    ```

Using structures

- To define a variable `p` with this new type :
  - `struct point p;`
- We access the `x` and `y` fields with `p.x` and `p.y`

```
struct point {
  float x;
  float y;
};
struct point p;
p.x=2;
p.y=3;
printf("%f\n",p.x);
```

typedef

- Replacing `struct point` by something shorter :
  - `typedef struct point Point2d;`
  - `Point2d p;` instead of `struct point p;`
- Or directly :

```
typedef struct {
  float x;
  float y;
} Point2d;
Point2d p;
p.x=2;
```

Examples

- The new type can be used as any other type :

```
Point2d milieu(Point2d p1,Point2d p2)
{
  Point2d m;
  m.x=(p1.x+p2.x)/2;
  m.y=(p1.y+p2.y)/2;
  return m;
}
```

- The function takes as input two parameters of type `Point2d` and returns a `Point2d`.

Assignation and comparison

- Assignation :
  - One can copy a `struct` variable into another, as with any other type :
  - ```
    Point2d p1,p2;
    p1.x=3;p1.y=4;
    p2=p1; // copy p1 into p2.
    ```
- Comparison:
  - One can not compare two `struct` variables with `if (p1==p2)`
  - One must compare each field separately.
    ```

Other example

- Function taking as input two points and outputting the distance between them.
  - For two points (x_1, y_1), (x_2, y_2), their distance is :

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

- ```
  float distance(Point2d p1,Point2d p2)
  {
    float dx,dy;
    dx=p2.x-p1.x;
    dy=p2.y-p1.y;
    return sqrt(dx*dx+dy*dy);
  }
  ```
  ```

Printing

- To print a `struct` variable, one must print each of its field.

```
void affiche(Point p)
{
  printf("Coordonnée x:%f\n",p.x);
  printf("Coordonnée y:%f\n",p.y);
}
```

Another example

- New type with first name, last name and age :

```
typedef struct {
    char *nom;
    char *prenom;
    int age;
} Personne;
```

- The new type `Personne` contains three fields :
  - Two strings `nom` and `prenom`
  - One `int` named `age`

Printing

- Printing a `Personne` variable :

```
void affiche(Personne p)
{
  printf("nom: %s, ",p.nom);
  printf("prenom: %s, ",p.prenom);
  printf("age: %d\n",p.age);
}
```

Main program

```
int main()
{
  Personne a;
  a.nom=(char *) strdup("Dupond");
  a.prenom=(char *) strdup("Jean");
  a.age=25;
  affiche(a);
}
```

- strdup
  - Allocates memory and copy the string given as input.

Structures

- Using structures

```
typedef struct {
  float x;
  float y;
} Point2d;

Point2d p;

p.x=2;
p.y=3;
Point2d q=p;
printf("%f\n",q.x);
```

Pointers and structures

- A pointer can refer to a structure.

```
typedef struct {
  float x;
  float y;
} Point2d;
Point2d p;
Point2d *q;
p.x=5;
q=&p;
(*q).y=3;
q->y=3; // equivalent
printf("%f\n",q->x);
printf("%f\n",p.y);
```

Pointers and structures

- Allocating memory :

```
typedef struct {
 float x;
 float y;
} Point2d;

Point2d *p;
p=(Point2d *) malloc(sizeof(Point2d));

p->x=3;
p->y=p->x+2;;
printf("%f\n",p->y);
```

Array of structures

- One can define an array of structures :

```
typedef struct {
  float x;
  float y;
} Point2d;

Point2d t[10];

t[5].x=3;
t[7].y=5;
```

Dynamic array

- One can define a dynamic array of structures :

```
typedef struct {
  float x;
  float y;
} Point2d;

Point2d *t;

t=(Point2d *) malloc(10*sizeof(Point2d));

t[5].x=3;
t[7].y=5;
```

Computing with large integers

- Limited precision in C :
  - `int`: 32 bits. Computing with values $< 2^{32}$.
- Computing with large integers :
  - One represents the big integers in base B in an array.
  - One implements addition, multiplication, division on big integers.
  - Existing libraries :
    - GMP: `www.swox.com/gmp`
    - NTL: `www.shoup.net`
    - Some parts written in assembly for better efficiency.

Technique

- Representing large integers :
  - An integer is represented as an array of digits in base B, with a sign bit.

$$a = \pm \sum_{i=0}^{k-1} a_i B^i = \pm (a_{k-1} \ldots a_0)_B$$

with $0 \leq a_i < B$. If $a \neq 0$, we must have $a_{k-1} \neq 0$.

- Basis :
  - One generally takes $B = 2^v$ for some v.
  - One can also take $B = 10$.

Addition

- Computing $c = a + b$ with $a, b > 0$
  - Let $a = (a_{k-1} \ldots a_0)$ and $b = (b_{\ell-1} \ldots b_0)$ with $k \geq \ell \geq 1$. Ket $c = (c_k c_{k-1} \ldots c_0)$

$$
\begin{array}{l}
carry \leftarrow 0 \\
\text{for } i = 0 \text{ to } \ell - 1 \text{ do} \\
\quad tmp \leftarrow a_i + b_i + carry \\
\quad carry \leftarrow tmp/B;\ c_i \leftarrow tmp \mod B \\
\text{for } i = \ell \text{ to } k - 1 \text{ do} \\
\quad tmp \leftarrow a_i + carry \\
\quad carry \leftarrow tmp/B;\ c_i \leftarrow tmp \mod B \\
c_k \leftarrow carry
\end{array}
$$

Substraction

- Computing $c = a - b$ with $a, b > 0$
  - Let $a = (a_{k-1} \dots a_0)$ and $b = (b_{\ell-1} \dots b_0)$ with $k \geq \ell \geq 1$.
    Let $c = (c_k c_{k-1} \dots c_0)$
    $carry \leftarrow 0$
    for $i = 0$ to $\ell - 1$ do
    $\quad tmp \leftarrow a_i - b_i + carry$
    $\quad carry \leftarrow tmp/B;\ c_i \leftarrow tmp \mod B$
    for $i = \ell$ to $k - 1$ do
    $\quad tmp \leftarrow a_i + carry$
    $\quad carry \leftarrow tmp/B;\ c_i \leftarrow tmp \mod B$
    $c_k \leftarrow carry$
  - If $a \geq b$ then $c_k = 0$, otherwise $c_k = -1$.
  - If $c_k = -1$, compute $c' = b - a$ and let $c := -c'$.

Multiplication

- Computing $c = a \cdot b$ with $a, b > 0$
  - Let $a = (a_{k-1} \ldots a_0)$ and $b = (b_{\ell-1} \ldots b_0)$ avec $k, \ell \geq 1$. Let $c = (c_{k+\ell-1} \ldots c_0)$

    $carry \leftarrow 0$
    for $i = 0$ to $k + \ell - 1$ do
        $c_i \leftarrow 0$
    for $i = 0$ to $k - 1$ do
        $carry \leftarrow 0$
        for $j = 0$ to $\ell - 1$ do
            $tmp \leftarrow a_i \cdot b_j + c_{i+j} + carry$
            $carry \leftarrow tmp/B$; $c_{i+j} \leftarrow tmp \mod B$
        $c_{i+\ell} \leftarrow carry$

Modular exponentiation

- We want to compute $c = a^b \mod n$.
  - Example: RSA
    - $c = m^e \mod N$ where m is the message, e the public exponent, and N the modulus.
- Naïve method:
  - Multiplying a in total b times by itself modulo n
  - Very slow: if b is 100 bits, roughly 2^{100} multiplications !

Square and multiply algorithm

- Let $b = (b_{\ell-1} \ldots b_0)_2$ the binary representation of b
  - $b = \sum_{i=0}^{\ell-1} b_i \cdot 2^i$
- Square and multiply algorithm :
  - Input : a, b and n
  - Output : $a^b \mod n$
  - $c \leftarrow 1$

    for $i = \ell - 1$ down to 0 do

    $\quad c \leftarrow c^2 \mod n$

    $\quad$ if $b_i = 1$ then $c \leftarrow c \cdot a \mod n$

    Output c

Analysis

- Let B_i be the integer with binary representation $(b_{\ell-1} \ldots b_i)_2$
  - $B_i = \sum_{j=i}^{\ell-1} b_j \cdot 2^{j-i}$
  - $B_{i-1} = 2 \cdot B_i + b_{i-1}$
- Claim : let c_i be the value of c at the end of step i :

$$c_i = a^{B_i} \mod n$$

  - Claim is true for $i = \ell - 1$
    - $B_{\ell-1} = b_{\ell-1}$
    - $c_{\ell-1} = 1$ if $b_{\ell-1} = 0$ and $c_{\ell-1} = a$ if $b_{\ell-1} = 1$
    - $c_{\ell-1} = a^{b_{\ell-1}} = a^{B_{\ell-1}} \mod n$

Analysis (2)

- Assume that claim is true for i.
  - Then $c_i = a^{B_i} \mod n$
  - $c_{i-1} = (c_i)^2 \mod n$ if $b_{i-1} = 0$
  - $c_{i-1} = (c_i)^2 \cdot a \mod n$ if $b_{i-1} = 1$

$$\begin{aligned} c_{i-1} &= (c_i)^2 \cdot a^{b_{i-1}} \mod n \\ c_{i-1} &= (a^{B_i})^2 \cdot a^{b_{i-1}} \mod n \\ c_{i-1} &= a^{2 \cdot B_i + b_{i-1}} = a^{B_{i-1}} \mod n \end{aligned}$$

- The output value c is $c = c_0$
  - $c_0 = a^{B_0} \mod n$ and $B_0 = b$ gives

$$c = a^b \mod n$$