Programmation C : TP 4

Ibrahim ALAME

19/01/2024

Diviseurs

Soit n un entier assez grand (≥ 2). Écrire une fonction `long* diviseurs(long n)` permettant de renvoyer un tableau dynamique (pointeur) d'entiers contenant les diviseurs propres de n inférieurs ou égal à $\sqrt{n}$ et se terminant par -1. Exemple d'exécution:

```
int main() {
        long n=24;
    long* t = diviseurs(n);
    printf("Diviseurs de %d:\n",n);
    int k=0;
    while(t[k]!=-1){
        printf("%d, %d\n",t[k],n/t[k]);
        k++;
    }
    free(t);

    return 0;
}
```

```
Diviseurs de 24:
2, 12
3, 8
4, 6
```

Nombres premiers

Commenter et expliquer les instructions de la fonction suivante:

```
int* nombresPremiers(int n){
    int* P=(int*) malloc(sizeof(int));
    P[0]=2;
    int dimP = 1;
    for(int i=3;i<n;i=i+2){
        int estPremier = 1;
        for(int k=0;k<dimP;k++) {
```

```
            int p = P[k];
            if (p * p > i) break;
            if (i % p == 0){
                estPremier = 0;
                break;
            }
        }
        if (estPremier){
            P=(int*) realloc(P,(dimP+1) * sizeof(int));
            P[dimP]=i;
            dimP++;
        }
    }
    P=(int*)realloc(P,(dimP+1)*sizeof(int));
    P[dimP]=-1;
    return P;
}
```

En s'inspirant de cette fonction, écrire une fonction `int ithPrime(int n)` permettant de déterminer le nième nombre premier. Exemple:

```
int main() {
        int n=5;
        printf("%d ième nombre premier:\n",n);
        int k = ithPrime(n);
    printf("%d\n",k);

    return 0;
}
```

```
5 ième nombre premier:
11
```

Numération

Expliquer les instructions de la fonction suivantes et donner des exemples d'exécution:

```
int* numeration(n,base){
    int* B= (int*)malloc(sizeof(int));
    int k=0;
    while (n!=0){
        B[k] = n % base;
        n = n / base;
        k++;
```

```
        B = (int*) realloc(B,(k+1)*sizeof(int));
    }
    B[k]=-1;
    return B;
}
```

Puissance

Soit $(k_i)_i$ les chiffres de l'écriture d'un entier n en base 2. $k_i = 0$ ou 1. Nous avons pour tout entier $x > 1$:

$$x^n = x^{\sum_{i=0}^{N} k_i 2^i} = \prod_{i=0}^{N} x^{k_i 2^i} = \prod_{i=0}^{N} z_i^{k_i}$$

où $z_i = x^{2^i}$.

1. Montrer que (z_i) vérifie $\left\{ \begin{array}{l} z_{i+1} = z_i^2 \\ z_0 = x \end{array} \right.$

2. Compléter le code suivant:

```
long puissance(long x, int n){
    int* d=numeration(n,2);
    long p = ...;
    long z = ...;
    int i=0;
    while(d[i]!=-1){
        int k=d[i];
        if(k==1) p=...;
        z=...;
        i++;
    }
    return p;
}
```

3. Écrire une fonction analogue à `puissance` permettant de calculer la puissance modulo: $x^n \bmod m$.

```
long puissanceModulo(long x, int n, int m){
    int* d=numeration(n,2);
    long p = ...;
    long z = ... % m;
    int i=0;
    while(d[i]!=-1){
        int k=d[i];
        if(k==1) p=... % m;
        z=... % m;
```

```
        i++;
    }
    return p;
}
```

Numérisation d'un texte

1. Écrire une fonction `char int2char(int n)` qui retourne le nième caractère de l'alphabet. On fait la convention que l'espace est le 0 ème caractère de l'alphabet puis écrire la bijection inverse `int char2int(char c)` qui à un caractère de l'alphabet fait correspondre son rang dans l'alphabet. On pourra utiliser la variable:

$$alphabet = \text{" abcdefghijklmnopqrstuvwxyz"}$$

2. Pour convertir une chaine de caractères $S = "a_0 a_1 a_2 ... a_n"$ en un entier N on suppose que la chaine S est la représentation de N en base 27:

$$N = \sum_{i=0}^{n} |a_i| 27^i \quad \text{où} \quad |a_i| = \text{char2int}(a_i)$$

Réciproquement

$$\text{numeration}(N, 27) = [|a_0|, |a_1|, |a_2|, ..., |a_n|]$$

Écrire la fonction `text2int` permettant de convertir une chaine de caractères en un entier puis écrire sa réciproque. Quel est le domaine de définition de la fonction `text2int`. Exemple d'exécution:

```
int main() {
    char text[] = "salut";
    int n = text2int(text);
    char* s = int2text(n);
    printf("%s => %d\n",text,n);
    printf("%d => %s\n",n,s);
    free(s);

    return 0;
}
```

```
salut => 11050957
11050957 => salut
```

Cryptage RSA

Le petit théorème de Fermat est généralisé par le théorème d'Euler : pour tout entier naturel non nul N et tout entier a premier avec N, on a

$$a^{\varphi(N)} \equiv 1 \bmod N$$

où $\varphi(N)$ désigne l'indicatrice d'Euler de N, égale au nombre des entiers premiers avec N et inférieurs à N. Si N est un nombre premier, alors $\varphi(N) = N - 1$ et l'on retrouve le petit théorème de Fermat.

Le théorème chinois s'exprime par

$$p \wedge q = 1 \Longrightarrow \varphi(p \times q) = \varphi(p) \times \varphi(q)$$

Si on choisit p et q premiers assez grands et on pose $N = pq$:

$$a^{(p-1)(q-1)} \equiv 1 \bmod N$$

donc

$$a^{k(p-1)(q-1)+1} \equiv a \bmod N$$

Il existe une valeur de k pour laquelle l'exposant $K = k(p-1)(q-1)+1$ est non premier et donc il existe une factorisation propre $K = ed$. Le couple (e, N) est appelé clé public et (d, N) clé privée. On définit deux fonctions (cryptage et décryptage):

$$f : a \mapsto a^e \bmod N \quad \text{et} \quad g : a \mapsto a^d \bmod N$$

Il est évident que $f \circ g = Id$ modulo N. Exemple d'exécution:

```c
int main() {
    long p = ithPrime(4000);
    long q = ithPrime(5000);
    long N=p*q;
    printf("%d\n",N);
    long K=3*(p-1)*(q-1)+1;
    long* D=diviseurs(K);
    long e = D[0];
    long d = K/D[0];
    char* message = "salut";
    int M=text2int(message);
    printf("Message numérisé: ( %s => %d )\n",message,M);
    int Mc = puissanceModulo(M,e,N);
    printf("Message numérique crypté: ( %d => %d )\n",M, Mc);
    printf("Message alphabétique crypté: ( %d = %s )\n",Mc,int2text(Mc) );
    int Md = puissanceModulo(Mc,d,N);
    printf("Message numérique décrypté: ( %d => %d = %s )\n",Mc, Md,int2text(Md));
    free(D);

    return 0;
}
```

```
Message numérisé: ( salut => 11050957 )
Message numérique crypté: ( 11050957 => 1589246440 )
Message alphabétique crypté: ( 1589246440 = ggbltbd )
Message numérique décrypté: ( 1589246440 => 11050957 = salut )
```

Long message

Expliquer le code suivant:

```
char** split(char* s){
    if(s==NULL)
        return NULL;
    int n = strlen(s);
    int N=n/5+2;
    char** t = (char**) malloc(N*sizeof(char*));
    for(int i=0;i<N-1;i++){
        t[i] = (char*) malloc(6*sizeof(char));
        for(int j=0;j<5;j++)
            t[i][j]=s[5*i+j];
        t[i][6]='\n';
    }
    t[N-1] = (char*) malloc(4*sizeof(char));
    strcpy(t[N-1],"FIN");
    return t;
}
```

Écrire une fonction `long* cryptage(char* m)` et une fonction `char* decryptage(long* L)` permettant de crypter et décrypter un message par la méthode RSA.

Casser le code RSA

Étant donné la clé public $(e = 13, N = 1838127743)$, calculer p et q puis k et enfin, la clé privée d.