INFORMATICA – Preparazione alla prova di teoria

ESERCIZIO 1

La *mappa di Mersenne* è la funzione

$$f(x) = 2x + 1.$$

La *successione di Mersenne* si ottiene componendo iterativamente f a partire da 0. Più precisamente:

- lo zeresimo elemento della successione è 0,
- il primo elemento della successione è $f(0)$ (cioè 1),
- il secondo elemento della successione è $f(f(0))$ (cioè 3),
- il terzo elemento della successione è $f(f(f(0)))$ (cioè 7),
- $\cdots$
- in generale, l'n-esimo elemento della successione è $\underbrace{f(f(\cdots f}_{n \text{ volte}}(0)\cdots))$.

Nota che la successione di Mersenne *è strettamente crescente*. Un *primo di Mersenne* è un intero positivo che:

1. appaia nella sequenza di Mersenne,
2. sia primo.

Scrivete una funzione che abbia un parametro formale intero `n` e restituisca `true` se `n` è un primo di Mersenne, `false` altrimenti.

ATTENZIONE! Non potete usare la formula chiusa per l'n-esimo elemento della successione di Mersenne.

SOLUZIONE

```
// test di primalita'
bool primo( int n ) {
    for ( int i = 2; i < n; i++ )
        if ( n % i == 0 )
                return false;
    return n > 1;
}

// test primo di Mersenne
bool primoMersenne( int n ) {
    int m;
    for ( m = 0; m < n; m = 2 * m + 1 );
    return m == n && primo( n );
}
```

ESERCIZIO 2

L'array `int V[100]` è caricato nelle sue prime 50 componenti da interi positivi. Scrivete un frammento di codice che *operando solo su* `V`, con l'uso eventuale di poche variabili ausiliarie ma senza usare un array di appoggio, crei in `V` una copia consecutiva di ogni intero pari ed elimini ogni intero dispari. Al termine, il codice deve anche stampare quanti pari sono stati aggiunti e quanti dispari eliminati.

Esempio.

`V` iniziale : 4 7 7 12 9 15 18 18 32 ...
`V` finale : 4 4 12 12 18 18 18 18 32 32 ...

SOLUZIONE

```
int d = 50, pr = 0, dr = 0;

for ( int i = 0; i < d; i++ )
    if ( X[i] % 2 == 0 ) {
        for ( int j = 98; j >= i; j-- )
            X[j + 1] = X[j];
        d++;
        i++;
        pr++;
    }
    else {
        for ( int j = i + 1; j < 100; j++ )
            X[j - 1] = X[j];
        d--;
        i--;
        dr++;
    }

cout << "Pari aggiunti: " << pr << " Dispari eliminati: " << dr << endl;
```

ESERCIZIO 3

Scrivete la funzione

```
int* set( int* X, int d )
```

che restituisce un array ottenuto dall'array `X` di dimensione `d` *privandolo delle ripetizioni di elementi* (in altre parole, la funzione trasforma `X` in un insieme).

Esempio. Se l'array `X` contenesse gli interi `{1, 2, 1, 4, 1, 5, 2}`, la chiamata `set( X, 7 )` deve restituire un array di dimensione `4` contenente `{4, 1, 5, 2}` (l'ordine di comparizione degli elemnti non è importante).

SOLUZIONE

```
// ricopre (cancella) l'i-esimo elemento di X spostando a sinistra il segmento [i, t]
// l'estremo destro t viene diminuito di uno (s'e' perso un elemento)
void shiftleft( int* X, int i, int& t ) {
    for ( int j = i; j <= t; j++ )
        X[j] = X[j + 1];
    t--;
}

int* set( int* X, int d ) {
    int t = d - 1;
    for ( int i = 0; i <= t; i++ )
        for ( int j = i + 1; j <= t; j++ )
            if ( X[i] == X[j] ) {
                shiftleft( X, i, t );
                i--;
                break;
            }
    int* Y = new int[t + 1];
    for( int i = 0; i <= t; i++ )
        Y[i] =  X[i];
    return Y;
}
```

ESERCIZIO 4

Scrivete la funzione

`int* union( int* A, int da, int* B, int db )`

che restituisce un array di dimensione opportuna contenente l'*unione insiemistica* degli interi contenuti negli array `A` e `B` di dimensione, rispettivamente `da` e `db`.

ATTENZIONE! Essendo l'unione insiemistica, l'array restituito non deve contenere elementi ripetuti.

SOLUZIONE

```
// testa se l'intero x compare nell'array X
bool exists( int* X, int d, int x ) {
    for ( int i = 0; i <  d; i++ )
        if ( X[i] == x )
             return true;
    return false;
}

int* union( int* A, int da, int* B, int db ) {
    int* C = new int[da + db];
    int k = 0;
    for ( int i = 0; i < da; i++ )
        if ( !exists( C, k, A[i] ) ) {
            C[k] = A[i];
            k++;
        }
    for ( int i = 0; i < db; i++ )
        if ( !exists( C, k, B[i] ) ) {
            C[k] = B[i];
            k++;
        }
    int* X = new int[k];
    for ( int i = 0; i < k; i++ )
        X[i] == C[i];
    return X;
}
```

ESERCIZIO 5

Scrivete la funzione

```
void ordina( int* X, int dim, int p )
```

ove `X` è un array di dimensione `dim` mentre `p` è un *intero positivo*. La funzione deve ordinare in senso crescente i soli elementi di `X` che stanno nelle *posizioni multiple di* `p` e lasciare invariato il resto dell'array.

SOLUZIONE

```
// scambia i contenuti di x e y
void sweep( int& x, int& y ) {
    int t = x;
    x = y;
    y = t;
}

void ordina( int* X, int dim, int p ) {
    for ( int i = 0; i < dim - p; i = i + p )
        for ( int j = i + p; j < dim; j = j + p )
            if ( X[j] < X[i] )
                swap( X[i], X[j] );
}
```

ESERCIZIO 6

Considerate la seguente funzione *ricorsiva*:

```
int funz( int* X, int x, int s, int t ) {
    if ( s > t )
        return 0;
    else
        return ( X[s] == x ? 1 : 0 ) + funz( X, x, s + 1, t );
}
```

Che significato ha il valore intero restituito dalla chiamata `funz( A, x, 0, d - 1 )`, ove `A` è un array di interi di dimensione `d`?

SOLUZIONE

La chiamata `funz( A, x, 0, d - 1 )` restituisce il *numero di occorrenze* di `x` nell'array `A` o, in altre parole, quante volte `x` compare in `A`.

———— ESERCIZIO 7 ————

Sia la seguente funzione *ricorsiva* che può essere richiamata ESCLUSIVAMENTE per valori positivi di `n`:

```
int funz( int n ) {
    if ( n == 1 )
        return 0;
    else
        return 1 + funz( n / 2 );
}
```

Che significato ha il valore intero restituito dalla chiamata `funz( n )`, con $\mathtt{n} \geq 1$?

SOLUZIONE

Per $\mathtt{n} \geq 1$, la chiamata `funz( n )` restituisce $\lfloor \log_2 \mathtt{n} \rfloor = \max\{\mathtt{k} \in \mathbf{N} : 2^{\mathtt{k}} \leq \mathtt{n}\}$.

———— ESERCIZIO 8 ————

Scrivete un frammento di codice che chieda all'utente una sequenza di interi terminata da zero. Al termine, il programma deve stampare il messaggio `"OK"` se la sequenza inserita è *strettamente monotona crescente*, `"KO"` altrimenti. (Ricordiamo che una sequenza di interi $x_1, x_2, \ldots, x_i, x_{i+1}, \ldots$ è strettamente monotona crescente se e solo se $x_i < x_{i+1}$ per ogni $i \geq 1$.)

SOLUZIONE

```
int a, b;
bool monotona = true;

cin >> a;
if ( a != 0 ) {
    cin >> b;
    while ( b != 0 )
        if ( b <= a )
            monotona = false;
        else {
            a = b;
            cin >> b;
        }
}

if ( monotona )
    cout << "OK" << endl;
else
    cout << "KO" << endl;
```