Algoritmi elementari

Problema:

Enunț

1. Scrieți o funcție care determină reversul unui număr dat (adică cifrele numărului se citesc de la dreapta spre stânga).

Observatii:

1. reversul unui număr care se termină cu 0 va ignora cifrele 0 de la final.

Programul

```
/*
Returneaza reversul unui numar.
In:
	- n: numarul pentru care se va calcula reversul.
Out:
	- reversul lui n.
*/
int revers(int n) {
	int r = 0;
	while (n > 0) {
		r = r * 10 + n % 10;
		n /= 10;
	}
	return r;

}
```

Exemple

Date de intrare	Rezultate
1344	4431
1001	1001
3300	33

Problema: palindrom

Enunt

2. Să se verifice dacă un numar este palindrom.
a) folosind reversul
b) fara a calcula reversul (fara a parcurge tot numarul), fara sir, string sau alte structuri de date.

Programul clasic

```
/*
Determină dacă un număr dat este palindrom.
In:
	- n: numarul care se va verifica.
Out:
	- true daca n e palindrom și false dacă nu.
*/
bool estePalindrom1(int n) {
	return n == revers(n);
}
```

Programul optimizat

```
/*
Determină dacă un număr dat este palindrom, intr-un mod mai eficient.
In:
	- n: numarul care se va verifica.
Out:
	- true daca n e palindrom și false dacă nu.
*/
bool estePalindrom2(int n) {
	int m = 0;
	if (n % 10 == 0) {
		return false;
	}
	while (n > 0) {
		m = m * 10 + n % 10;
		if (m == n) {
			return true;
		}
		n /= 10;
		if (m == n) {
			return true;
		}

		if (m > n) {
			return false;
		}
	}
	return false;

}
```

Exemple

Date de intrare	Rezultate
1344	false
1001	true
3300	false

Problema: factorial

Enunt

3. Determinare factorial.
 a) iterativ
 b) recursiv

Programul iterativ

```
/*
Determină factorialul unui numar. n! = 1*2*3*...*n.
In:
	- n: numarul pentru care se calculeaza factorialul.
Out:
	- factorialul lui n, n!.
*/
int factorial1(int n) {
	int f = 1;
	for (int i = 1; i <= n; i++) {
		f *= i;
	}
	return f;
}
```

Programul recursiv

```
/*
Determină factorialul unui numar. n! = 1*2*3*...*n.
In:
	- n: numarul pentru care se calculeaza factorialul.
Out:
	- factorialul lui n, n!.
*/
int factorial2(int n) {
	if (n == 0) {
		return 1;
	}
	return n * factorial2(n - 1);
}
```

Exemple

Date de intrare	Rezultate
5	120
0	1
3	6

Problema: zerouri factorial

Enunt

4. Fiind dat un numar natural n, sa se determine numarul de cifre 0 cu care se termina factorialul numarului n.

a) prin calcularea factorialului

b) fara calcularea factorialului.

Programul care calculează factorialul

```
/*
Determină numarul de zerouri de la sfarsitul lui n!, calculand factorialul.
In:
	- n: numarul pentru care se determina zerourile terminale in n!.
Out:
	- numarul de zerouri terminale ale lui n!.
*/
int cifre0_1(int n) {
	int f = factorial1(n);
	int nr = 0;
	while (f % 10 == 0) {
		nr++;
		f /= 10;
	}
	return nr;
}
```

Programul care nu calculează factorialul

```
/*
Determină numarul de zerouri de la sfarsitul lui n!, fara a calcula factorialul.
In:
	- n: numarul pentru care se determina zerourile terminale in n!.
Out:
	- numarul de zerouri terminale ale lui n!.
*/
int cifre0_2(int n) {
	int nr = 0;
	for (int i = 5; n / i >= 1; i *= 5) {
		nr += n / i;
	}
	return nr;
}
```

Exemple

Date de intrare	Rezultate
100	24
5	1
7	1

Problema: fibonacci

Enunt

5. Sa se determine al n-lea termen din sirul lui Fibonacci.
 a) iterativ
 b) recursiv

Programele iterative

```
/*
Determină al n-lea numar fibonacci.
In:
	- n: numarul fibonacci dorit.
Out:
	- al n-lea numar fibonacci.
*/
int fibo1(int n) {
	int prev = 0, crt = 1;
	for (int i = 1; i < n; i++) {
		int f = crt + prev;
		prev = crt;
		crt = f;

	}
	return n == 0 ? prev : crt;
}

/*
Determină al n-lea numar fibonacci.
In:
	- n: numarul fibonacci dorit.
Out:
	- al n-lea numar fibonacci.
*/
int fibo2(int n) {
	int prev = 0, crt = 1;
	for (int i = 0; i < n / 2; i++) {
		prev += crt;
		crt += prev;
	}
	return n % 2 == 0 ? prev : crt;
}
```

Programele recursive

```
/*
Determină al n-lea numar fibonacci.
In:
	- n: numarul fibonacci dorit.
Out:
	- al n-lea numar fibonacci.
*/
int fibo3(int n) {
	if (n == 0 || n == 1) {
		return n;
	}
	return fibo3(n - 2) + fibo3(n - 1);
}
```

```c
/*
Determină al n-lea numar fibonacci recursiv, folosind parametri ajutatori.
In:
	- n: cate numere mai avem de calculat.
	- prev: precedentul număr fibonacci.
	- crt: numărul fibonacci curent.
Out:
	- al n-lea numar fibonacci.
*/
int fibo4_aux(int n, int prev, int crt) {
	if (n == 0 || n == 1) {
		return n == 0 ? prev : crt;
	}
	return fibo4_aux(n - 1, crt, prev + crt);
}

/*
Determină al n-lea numar fibonacci.
In:
	- n: numarul fibonacci dorit.
Out:
	- al n-lea numar fibonacci.
*/
int fibo4(int n) {
	return fibo4_aux(n, 0, 1);
}
```

Exemple

Date de intrare	Rezultate
7	13
5	5
6	8

Problema: proprietate

Enunt

6. Se citeste nu numar natural n. Sa se afiseze toate numerele din intervalul [a, b) dat (a,b>0) care au propiretatea P cu n. Doua numere au proprietatea P daca scrierile lor in baza 10 contin acelasi cifre (ex: 23 si 3223 au proprietatea P).

Programul

```
/*
Determina o reprezentare pe biti a cifrelor unui numar.
In:
	- n: numarul dat.
Out:
	- un numar in care al k-lea bit este 1 daca exista cifra k in n.
*/
int configuratie_cifre(int n) {
	int b = 0;
	while (n > 0) {
		b |= 1 << (n % 10);
		n /= 10;
	}
	return b;
}


/*
Determina daca doua numere sunt in proprietatea P.
In:
	- nr1: primul numar.
	- nr2: al doilea numar.
Out:
	- true daca nr1 P nr2, false altfel.
*/
bool proprietateP(int nr1, int nr2) {
	int b1 = configuratie_cifre(nr1);
	int b2 = configuratie_cifre(nr2);
	return b1 == b2;
}

/*
Afiseaza toate numerele din [a, b) care sunt in proprietatea P cu n.
*/
void afiseaza_numere(int n, int a, int b) {
	for (int i = a; i < b; i++) {
		if (proprietateP(n, i)) {
			cout << i << endl;
		}
	}

}
```

Exemple

Date de intrare	Rezultate
32, 100, 1000	223 232 233 322 323 332