Principii de programare

1. Principiul de numărare

Când avem de numărat ceva, declarăm şi iniţializăm o variabilă *contor* cu 0, apoi parcurgem mulţimea vizată şi incrementăm variabila *contor* ori de câte ori găsim un element favorabil.

```
#include<iostream>
using namespace std;
int deCateOri(char text[], char ch){
	//returneaza de cate ori se afla caracterul ch
	//in stringul text
	int contor=0;
	for(int i=0; text[i]!='\0';i++){
		if(text[i]==ch) contor++;
	}
	return contor;
}
int main(){
	char text[]="ab abc abcd";
	char ch='b';
	cout<<"in stringul"<<endl;
	cout<<text<<endl;
	cout<<"caracterul"<<endl;
	cout<<ch<<endl;
	cout<<"se afla de "<<deCateOri(text,ch)<<" ori"<<endl;
	return 0;
}
```

2. Principiul de sumare

Când avem de calculat o sumă, declarăm şi iniţializăm o variabilă *s* cu 0, apoi formăm pe rând termenii sumei şi îi acumulăm în variabila *s*.

```
#include<iostream>
#include<math.h>
using namespace std;
double suma(int n){
	//calculeaza 1/(n+1)+1/(n+2)+...+1/(n+n)
	double s=0;
	for(int i=1;i<=n;i++){
		s+=1.0/(n+i);
	}
	return s;
}
int main(){
	cout<<suma(1000000)<<endl;
	cout<<log(2.0)<<endl;
	return 0;
}
```

3. Principiul de înmulţire

Când avem de calculat un produs, declarăm şi iniţializăm o variabilă *p* cu 1, apoi formăm pe rând factorii produsului şi amplificăm cu ei variabila *p*.

```
#include<iostream>
using namespace std;
double produs(int n){
	//calculeaza (1-1/(2*2))*(1-1/(3*3))*...*((1-1/(n*n))
	double p=1.0;
	for(int i=2; i<=n; i++){
		p*=1.0-1.0/(i*(double)i);
	}
	return p;
}
int main(){
	cout<<produs(1000000)<<endl;
	cout<<1.0/2<<endl;
	return 0;
}
```

4. Principiul de optim

Când avem de aflat o valoare optimă dintr-un şir de valori, declarăm şi iniţializăm o variabilă *valOptim* cu prima valoare din şir, apoi formăm pe rând termenii şirului şi actualizăm variabila *valOptim* ori de câte ori găsim o valoare mai bună.

```
#include<iostream>
using namespace std;
int deCateOri(char text[], char ch){
	//returneaza de cate ori se afla caracterul ch in stringul text
	int contor=0;
	for(int i=0; text[i]!='\0';i++){
		if(text[i]==ch) contor++;
	}
	return contor;
}
char celMaiDesIntalnit(char text[]){
	//returneaza cel mai des intalnit caracter din text,
	//daca sunt mai multe, il returneaza pe primul din stanga
	char chMax='\0', chActual;
	int valMax=0, valActual;
	for(int i=0;text[i]!='\0';i++){
		chActual=text[i];
		valActual=deCateOri(text,chActual);
		if(valActual>valMax){
			valMax=valActual;
			chMax=chActual;
		}
	}
	return chMax;
}
```

```
int main(){
	char text[]="xyz yz ayz";
	cout<<"In stringul"<<endl;
	cout<<text<<endl;
	cout<<"primul caracter dintre cele mai des intalnite este"<<endl;
	cout<<'\''<<celMaiDesIntalnit(text)<<'\''<<endl;
	return 0;
}
```

5. Principiul primului contra-exemplu

Când avem de stabilit dacă toate elementele unei mulţimi finite au o anumită proprietate, parcurgem mulţimea şi căutăm *primul contra-exemplu*, adică primul element care nu are acea proprietate.

Dacă îl găsim ne oprim şi răspundem: "fals, nu toate elementele au proprietatea cerută", altfel continuăm căutarea. Dacă am parcurs toată mulţimea şi nu am găsit nici un contra-exemplu răspunsul este "da, toate elementele au proprietatea cerută".

```
#include<iostream>
using namespace std;
bool toateSuntPozitive(int tab[], int n){
	//decide daca toti tab[i]>0
	//cautam primul contra-exemplu
	for(int i=0; i<n;i++){
		if(tab[i]<=0) return false;
	}
	return true;
}
int main(){
	int tab[3]={1,0,3};
	if(toateSuntPozitive(tab,3))
		cout<<"DA, toate sunt strict pozitive"<<endl;
	else
		cout<<"NU, nu toate sunt strict pozitive"<<endl;
	return 0;
}
```

În cazul când nu putem opri căutarea prin ieşirea dintr-o funcţie de decizie, utilizăm o variabilă de decizie:

```
#include<iostream>
using namespace std;
int main(){
	const int n=3;
	int tab[n]={1,-2,3};
	//decidem daca toate elementele tab[i]>0
	//cautam primul contra-exemplu
	bool amGasitNegative=false; //variabila de decizie
	for(int i=0; i<n;i++){
		if(tab[i]<=0) {
			amGasitNegative=true;
```

```
                break;
            }
        }
        if(!amGasitNegative)
            cout<<"DA, toate sunt strict pozitive"<<endl;
        else
            cout<<"NU, nu toate sunt strict pozitive"<<endl;
        return 0;
}
```

Variabila de decizie o iniţializăm cu sensul logic "nu am găsit încă un ceea ce căutăm, adică un contra-exemplu" şi o schimbăm când găsim primul contra-exemplu, ocazie cu care oprim şi căutarea.

6. Principiul primului exemplu

Când avem de stabilit dacă într-o mulţime finită există un element cu o anumită proprietate, parcurgem mulţimea şi căutăm *primul exemplu*, adică primul element cu acea proprietate.

Dacă îl găsim ne oprim şi răspundem: "adevarat, există un element cu proprietatea cerută", altfel continuăm căutarea. Dacă am parcurs toată mulţimea şi nu am găsit nici un exemplu răspunsul este "fals, nu există elemente cu proprietatea cerută"

```
#include<iostream>
using namespace std;

bool exista2egale(int a[], int n){
        //decide daca tabloul in a[] exista o pereche de componente egale

        //formam toate perechile (a[i],a[j]) cu i<j
        //si cautam primul exemplu de a[i]==a[j]
        for(int i=0;i<n;i++){
            for(int j=i+1;j<n;j++){
                if(a[i]==a[j]) return true;
            }
        }
        return false;
}

int main(){
        int tab[4]={1,0,3,0};
        if(exista2egale(tab,4))
            cout<<"DA, exita componente egale"<<endl;
        else
            cout<<"NU, nu exita componente egale"<<endl;
        return 0;
}
```

Iată şi varianta "inline", cu variabilă de decizie:

```
#include<iostream>
using namespace std;
int main(){
	const int n=4;
	int a[n]={1,0,3,0};
	//decidem daca exista o pereche de componente egale

	//formam toate perechile (a[i],a[j]) cu i<j
	//si cautam primul exemplu
	bool amGasit2Egale=false;
	for(int i=0;i<n;i++){
		for(int j=i+1;j<n;j++){
			if(a[i]==a[j]) {
				amGasit2Egale=true;
				break;//oprim cautarea
			}
		}
		if(amGasit2Egale) break; //oprim cautarea !
	}
	if(amGasit2Egale)
		cout<<"DA, exita componente egale"<<endl;
	else
		cout<<"NU, nu exita componente egale"<<endl;
	return 0;
}
```

Observăm că şi în acest caz valoarea iniţială a variabilei de decizie are sensul “nu am găsit încă ceea ce căutăm”.

Alt exemplu:

```
#include<iostream>
using namespace std;
const int dimMax=3;
bool exitaLinieNula(int A[dimMax][dimMax],int n){
	//decide daca in matricea A de tip n x n
	//exita o linie cu toate elementele nule

	//cautam prima linie nula
	for(int i=0;i<n;i++){
		//stabilim daca pe linia i
		//toate elementele sunt egale cu 0
		//cautam primul contra-exemplu
		bool amGasitNenule=false;
		for(int j=0; j<n; j++){
			if(A[i][j]!=0){
				amGasitNenule=true;
				break;
			}
		}
		if(!amGasitNenule) return true;
	}
	//nu am gasit nici un exemplu de linie nula
	return false;
```

```
}
int main(){
	int A[dimMax][dimMax]={{1,0,3},{0,1,0},{1,0,3}};
	if(exitaLinieNula(A,3))
		cout<<"DA, are o linie nula"<<endl;
	else
		cout<<"NU, nu are nici o linie nula"<<endl;
	return 0;
}
```

In final, un exemplu de nota zece:

```
#include<iostream>
using namespace std;
const int dimMax=3;
bool toateLiniileAuZece(int A[dimMax][dimMax],int n){
	//decide daca in matricea A de tip n x n
	//toate liniile au macar cate un element egal cu 10

	//cautam prima linie contra-exemplu (o linie fara nici nu 10)
	for(int i=0;i<n;i++){
		//stabilim daca pe linia i exista macar un 10
		//cautam primul exemplu
		bool amGasit1Zece=false;
		for(int j=0; j<n; j++){
			if(A[i][j]==10){
				amGasit1Zece=true;
				break;
			}
		}
		if(!amGasit1Zece) return false;
	}
	//nu am gasit nici o linie contra-exemplu
	return true;
}
int main(){
	int A[dimMax][dimMax]={{1,10,3},{10,0,0},{10,0,3}};
	if(toateLiniileAuZece(A,3))
		cout<<"DA, pe fiecare linie exista macar un 10"<<endl;
	else
		cout<<"NU, nu pe fiecare linie exista un 10"<<endl;
	return 0;
}
```