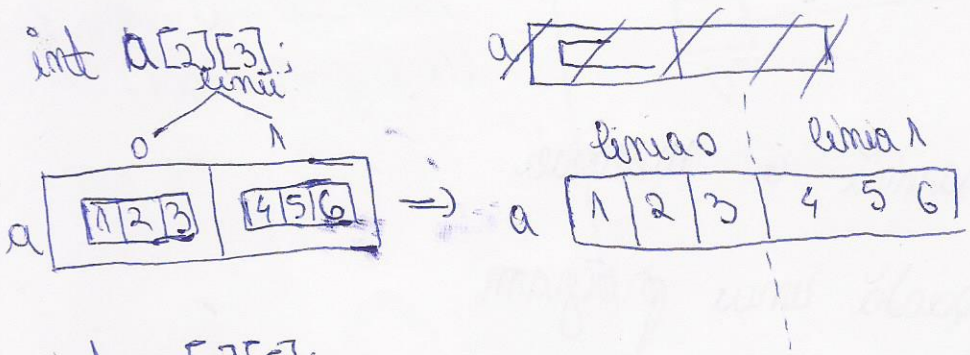
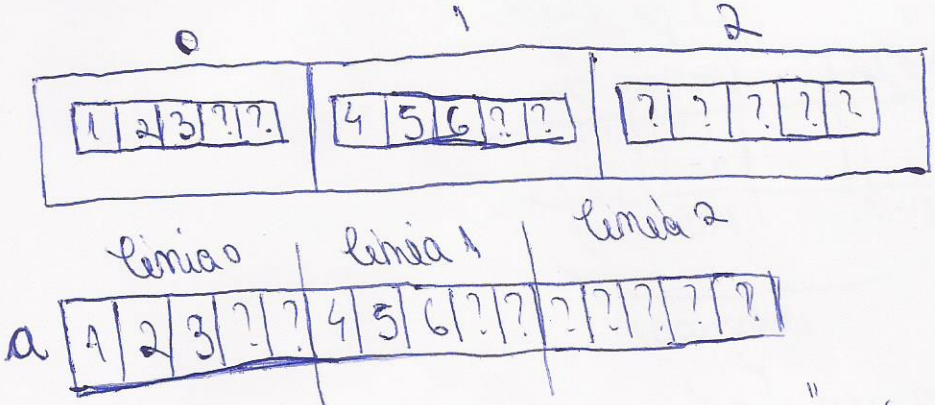
Tablouri unidimensionale și pointeri

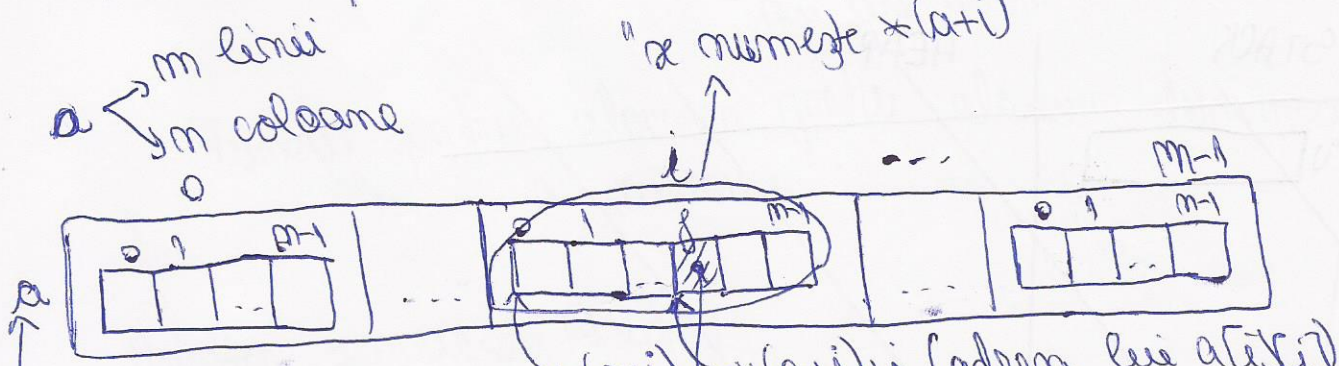
$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

int a[2][3];



int a[3][5];





a ⇐ &a[0] ⇐ &a[0][0]

int **a

void tabloua(int a[], int m)

tabloua(a[i], m)

↳ linie a matricii

a[i][j] ⇐⇒ *(*(a+i)+j)

v[i] ⇐⇒ *(v+i) = *(i+v) =

1) *(*(a+i)[j])

2) *(a[i]+j)

3) j[*(a+i)]

4) i[a][j]

5) j[a[i]] = j[i[a]]

(j+(a+i))

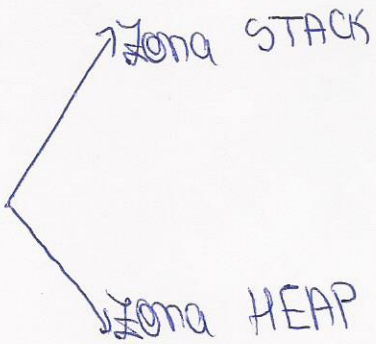
~~*(*[a]+j)~~

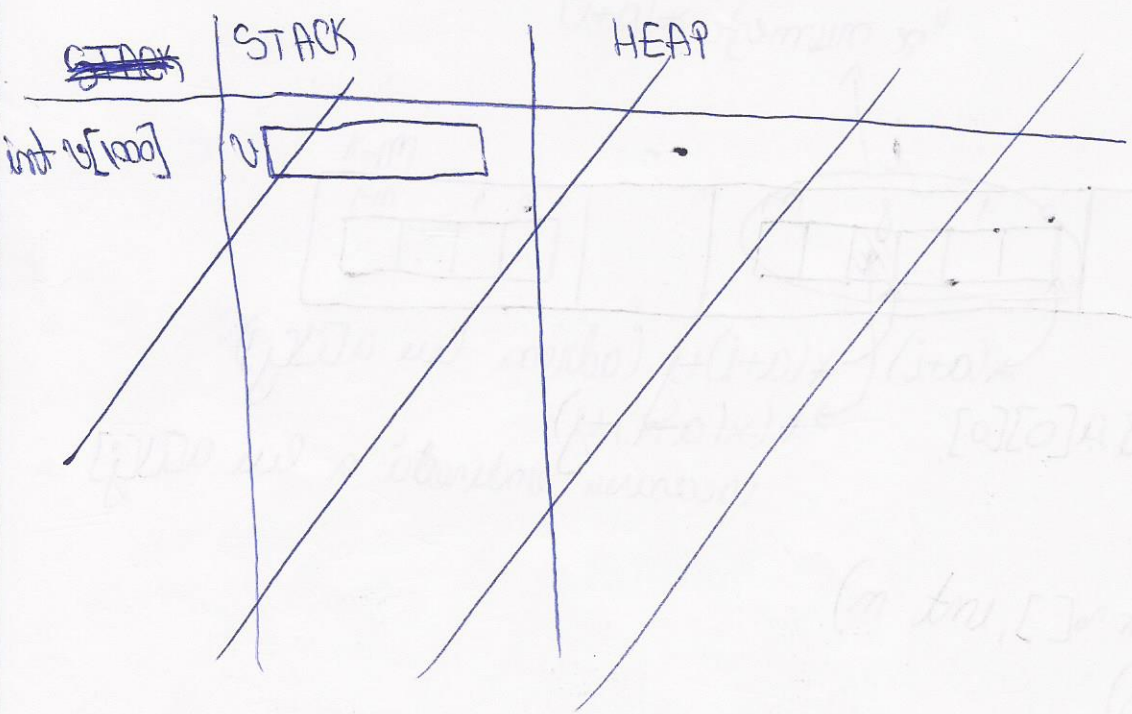
((i+a)+j)

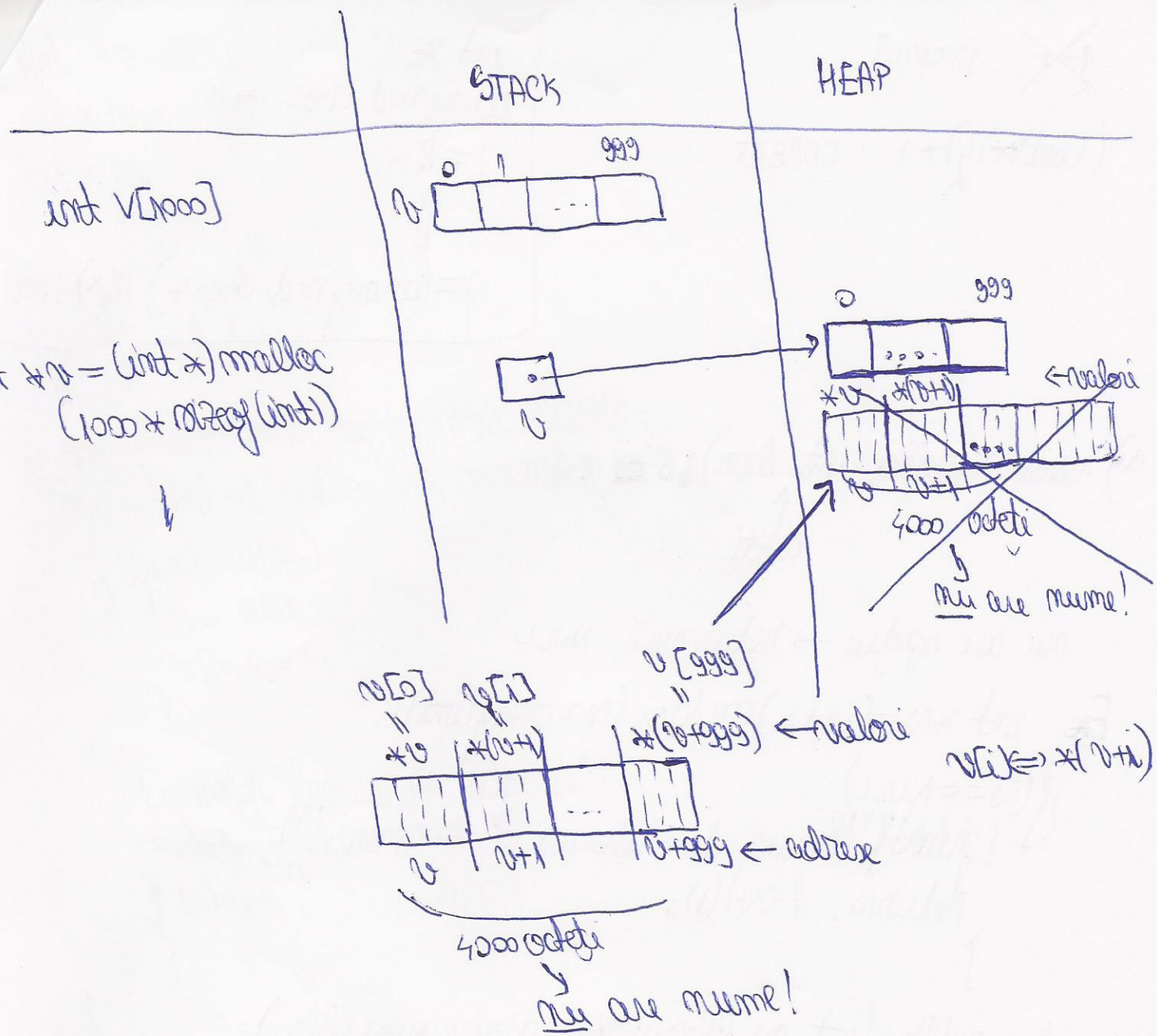
*(i[a]+j)

Alocarea dinamică a memoriei

Zona de memorie asociată unui program







Funcții pentru alocarea dinamică a memoriei

Pointer generic ⇔ void *

~~Ex~~ int x, *p; ~~nu are aritmetică~~
void *q;
p = &x;
q = p;

- pointerii generici nu au aritmetică!!

~~g++;~~ incorect

((int*)g)++ CORECT

```
int x;
unsigned char *p;
p=&x;
⇕
p=(unsigned char*)(&x);
```

a) void *malloc(int dim);

dată

nu are spațiu → returnează NULL

Ex int *v=(int*)malloc(n*sizeof(int));

```
if(v==NULL)
{printf("Eroare de alocare a memoriei!");
 return; // exit(1);
}
```

b) void *calloc(int nr blocuri, int dimensiune bloc);

dată

Ex: int *v = (int*) calloc(n, sizeof(int));

* inițializează toată zona de memorie cu 0!

c) void *realloc (void *pointer, int dim_nouă);

size_t ⇐⇒ int

Obs

1) pointer==NULL ⇐⇒ malloc

2) dim_nouă==0 ⇐⇒ free(pointer)

! dacă nu poate realoca ⇒ v=NULL (distruge vectorul)

~~v=(int*)realloc(v,1000);~~

```
aux=(int*)realloc(v,1000);
if(aux!=NULL)
 v=aux;
...
```

else
v nu a fost realocat, dar este intact.
}

```
int *v=NULL;
int *aux=NULL;
int x,n=0;

do
{scanf("%d",&x);
 n++;
 if(x!=0)
 aux=(int*)realloc(v,n*sizeof(int));
   if(aux!=NULL)
     {v=aux;
      v[n-1]=x;
     }
   else
     {printf("Eroare de alocare");
      free(v); // ???
      return; //???
     }
}
while(x!=0);
```

d) void free(pointer)
eliberează zona de memorie