קלט/פלט אוטומטי:

אפשרות 1

```
fscanf(fin,"%i",&i);
fprintf(fout,"%i\n",i);
```

אפשרות 2

```
scanf("%i",&i);
printf("%i\n",&i);
```

ואז: a.out < infile > outfile

אפשרות 3

קלט בפקודת ההרצה

(command line arguments)

במקום: main()

```
main(int argc, char *argv[])
```

a.out infile 53 ⟶

```
argc=3
argv[0]="a.out"
argv[1]="infile"
argv[2]="53"
argv[3]=NULL
```

```
sscanf(argv[2],"%i",&i);
```

numerical recipes -ל הקדמה

nrerror("bad input in function integrate");

```
void nrerror(char error_text[])
{
  fprintf(stderr,"Numerical Recipes run-time error...\n");
  fprintf(stderr,"%s\n",error_text);
  fprintf(stderr,"...now exiting to system...\n");
  exit(1);
}
```

void nrerror(char *error_text)

#include<stdlib.h>

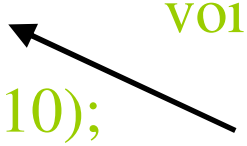
<u>אינטגרל:</u>

float qromb(float (*func)(float), float a, float b);

הקדמה ל- numerical recipes

```
float *v1;
v1=vector(5,10);
free_vector(v1,5,10);
```

```
float *vector(long nl, long nh);
void free_vector(float *v, long nl, long nh);
```



v1[5], …, v1[10]

v1[5] ⇔ *(v1+5)

```
float v[6],*v1;
v1=v-5;
```

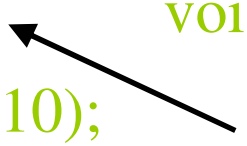
v1[5] ⇔ v[0]

v1[10] ⇔ v[5]

הקדמה ל- numerical recipes

```
float *v1;
v1=vector(5,10);
free_vector(v1,5,10);
```

```
float *vector(long nl, long nh);
void free_vector(float *v, long nl, long nh);
```



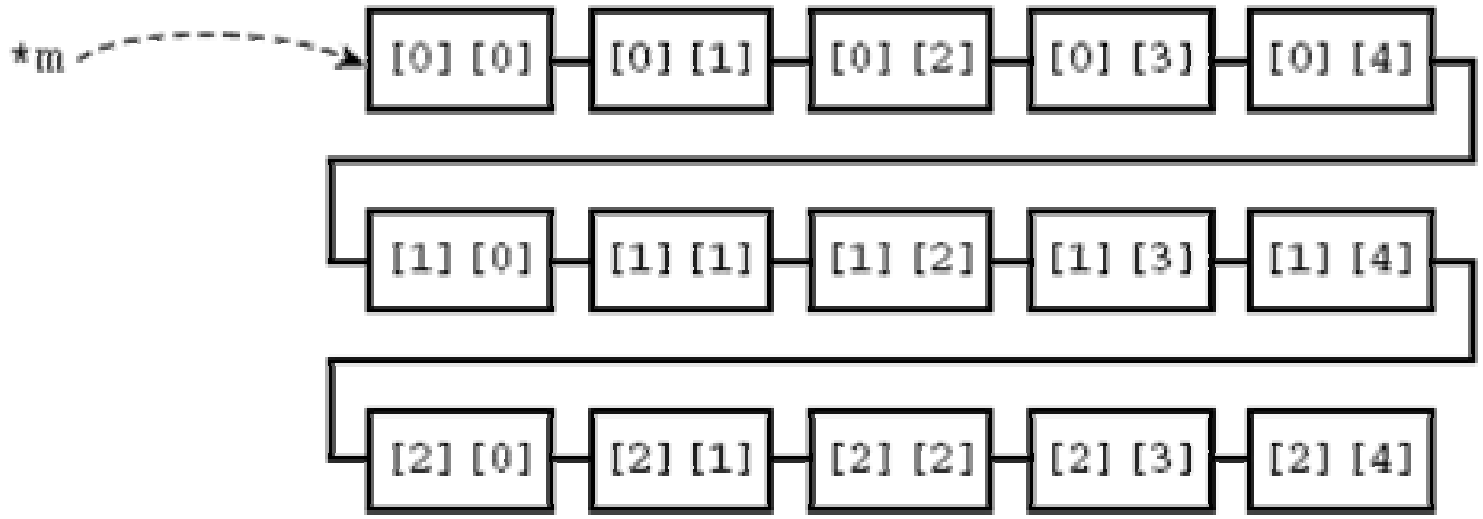
v1[5], …, v1[10]

```
float *vector(long nl, long nh)
{
  float *v;
  v=(float *) malloc( (nh-nl+1)*sizeof(float) );
  if (!v) nrerror("allocation failure in vector()");
  return v-nl;
}
```

```
void free_vector(float *v, long nl, long nh)
{  free(v+nl);  }
```

numerical recipes -הקדמה ל

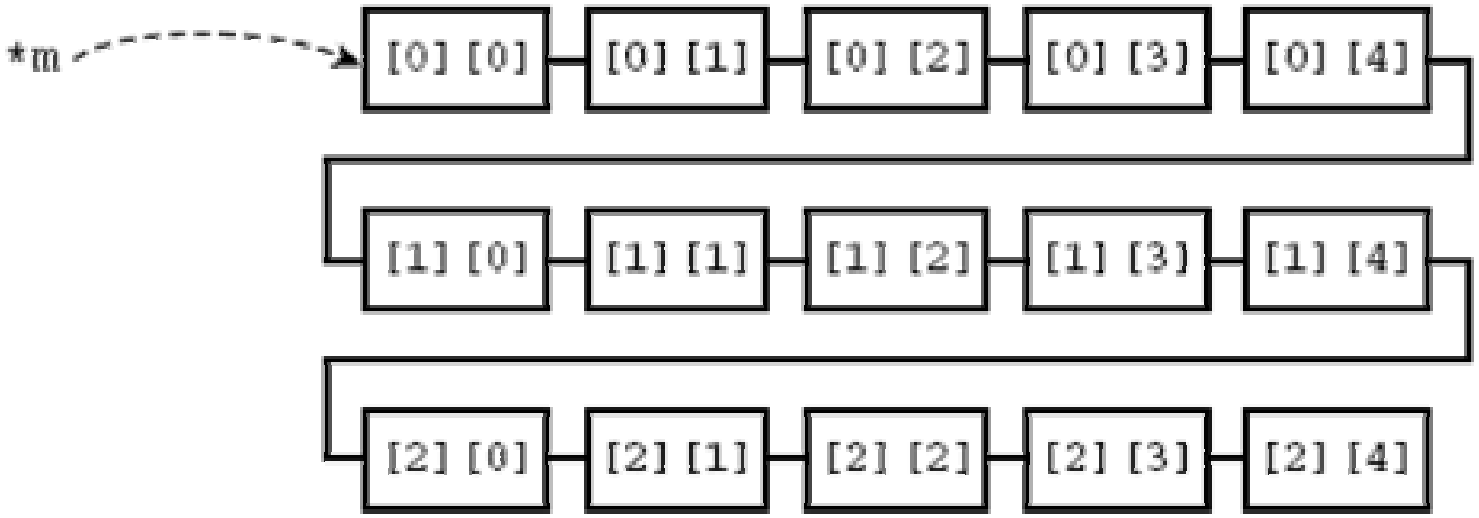
```
float m[3][5];
```



m[2][3] ⇔ *(*m+2*5+3)

```
float f=func(m);
void func(float v[][5])
{ return v[3][5];  }
```

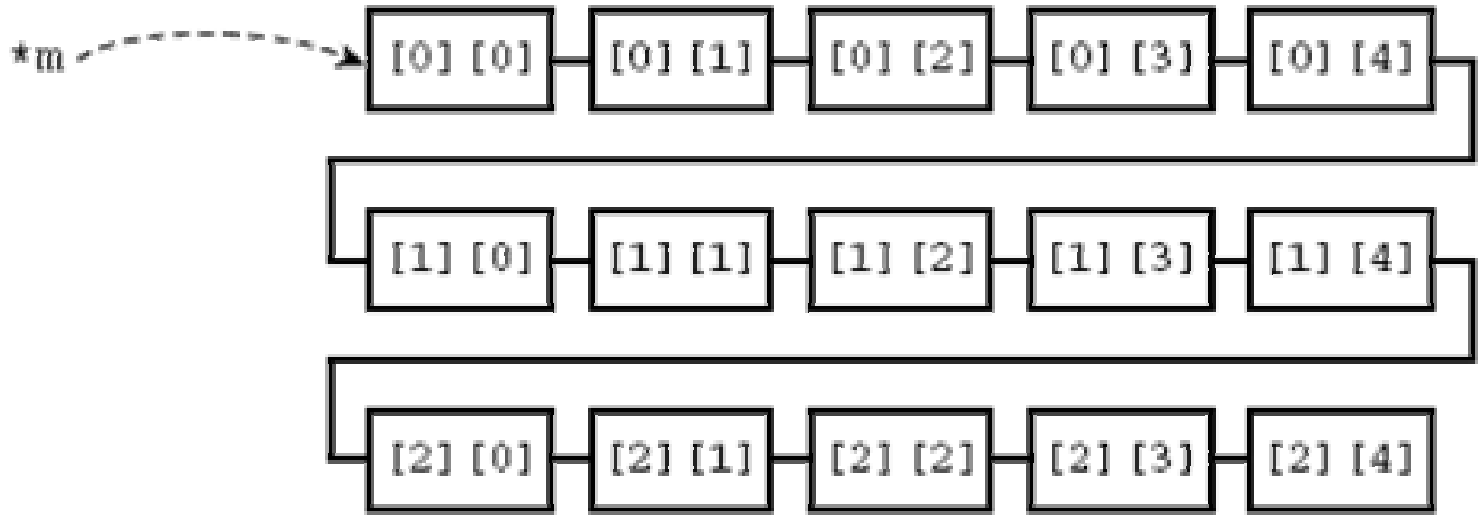
הקדמה ל- numerical recipes

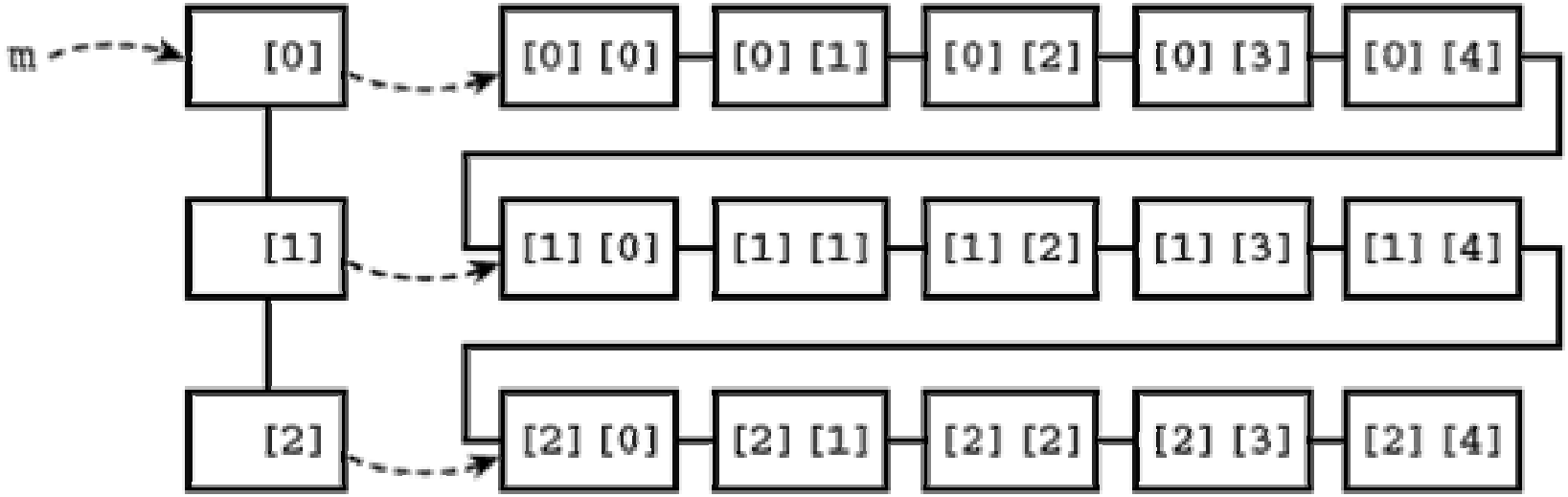


~~void matadd(float a[][], float b[][], float c[][], int n, int m) { … }~~

~~void matadd(float a[][m], float b[][m], float c[][m], int n, int m) { … }~~

void matadd(float a[][5], float b[][5], float c[][5], int n, int m) { … }






```
float **m, *a[3], *b;
m=a;                    m[2] ⇔ a[2]=b      m[2][3]=(m[2])[3] ⇔ b[3]
b=a[2];          void matadd(float **a, float **b, float **c, int n, int m)
```

```
float **matrix(long nrl, long nrh, long ncl, long nch)
{
  long i, nrow=nrh-nrl+1,ncol=nch-ncl+1;
  float **m;

  /* allocate pointers to rows */
  m=(float **) malloc( nrow*sizeof(float*) );
  m -= nrl;

  m[nrl]=(float *)  malloc( (nrow*ncol)*sizeof(float) );
  m[nrl] -= ncl;

  for (i=nrl+1; i<=nrh; i++) m[i]=m[i-1]+ncol;
  return m;
}
```

m[nrl..nrh][ncl..nch]

```
void free_matrix(float **m, long nrl, long nrh, long ncl, long nch)
{
  free( m[nrl]+ncl );
  free( m+nrl );
}
```

שימוש:

```
float func(float *v)
{
   return v[5];
}

float *v1;
v1=vector(5,10);
func(v1);
free_vector(v1,5,10);
```

```
float func(float **m)
{
   return m[4][15];
}

float **m1;
m1=matrix(1,5,10,20);
func(m1);
free_vector(m1,1,5,10,20);
```