# C programavimo kalba

**14 paskaita**

**(Kurso santrauka)**

# *Programavimas*

**Programavimas** – tai problemos sprendimo užrašymas kompiuterinei sistemai suprantama forma, kurį ji gali įvykdyti.

Norint tinkamai parašyti programą:

- Reikia mokėti išspręsti problemą "rankomis"
- Mokėti sudaryti sprendimo kelią (algoritmą)
- Mokėti vieną iš programavimo kalbų, suprantamą kompiuteriui.

*Programming is **not** about mathematics, it is about organisation* !
(Rob Miles "Introduction to C Programming" 1995 )

Programavimo kalba reikalinga, nes:

- Įprastinė kalba per daug sudėtinga (sinonimai, antonimai, šauktukai ir t .t.)
- Trūksta konkretumo ir vienareikšmiškumo

# Programavimo kalba C

C programavimo kalbą sukūrė Dennis Ritchie (Bell Lab.) 1972 m. UNIX OS sukūrimui. Jis rėmėsi tokiomis programavimo kalbomis:

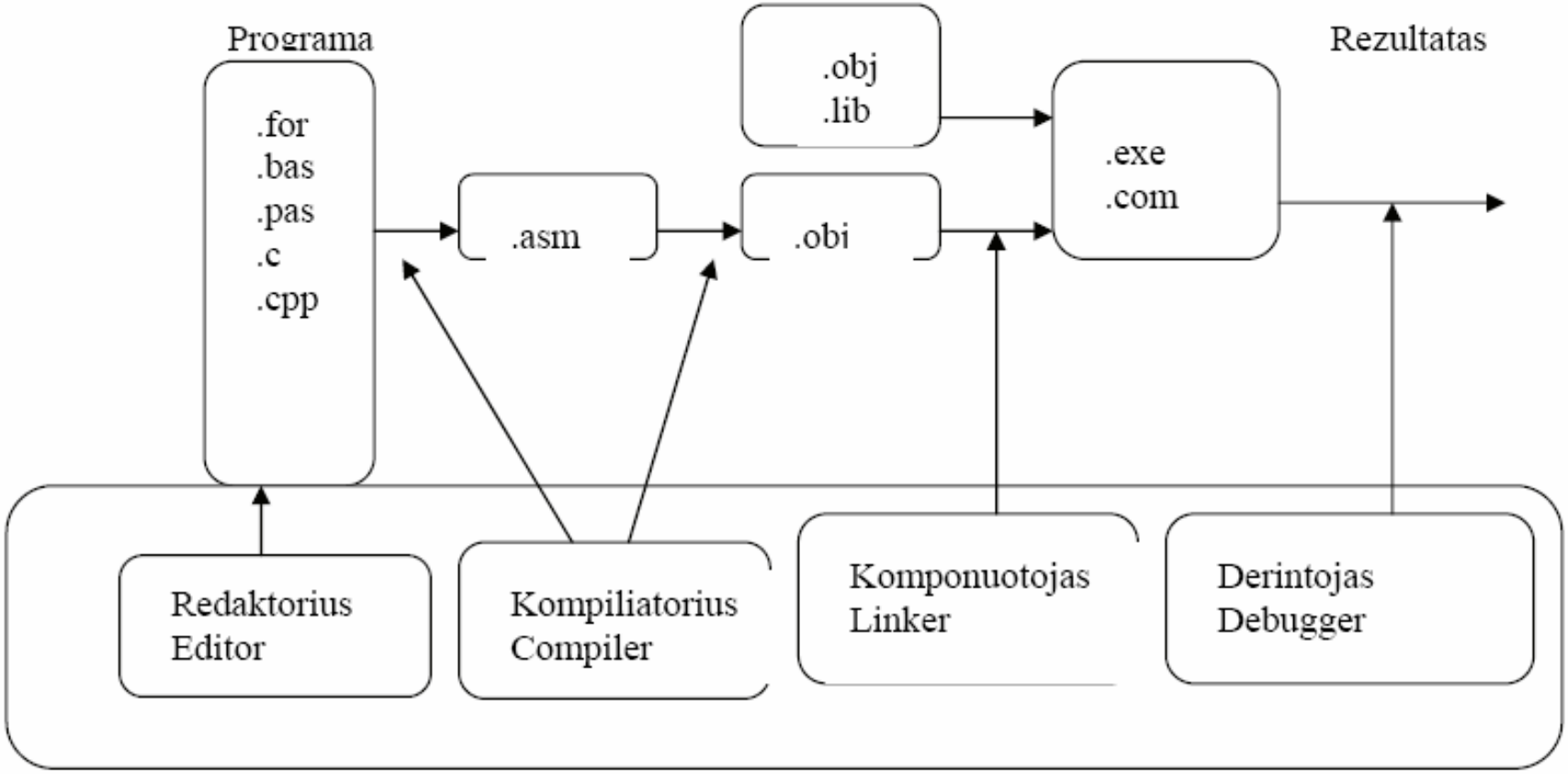
- ALGOL (1960)
- CPL (1963)
- B (1970)

C kalbos tęsinys objektiškai orientuotam programavimui C++ (B.Stroustrup, AT&T Bell Lab. ~1980 m.)

**Standartai:**

- *Tradicinis C* (B.Kernighan, D.Ritchie. The C Programming Language. 1978)
- *Standard C (1989)* **ANSI C** (ANSI X3.159-1989) ISO/IEC 9899:1990
- *Standard C (1995)*
- *Standrad C (1999)* ISO/IEC 9899:1999

# Vykdomosios programos generavimas



# Duomenų tipai

**Duomenų tipai ir jų raktiniai žodžiai.**

*Raktiniai žodžiai:* **int, long, short, unsigned, char, float, double**

| Tipas | Dydis, B | Reikšmė |
|---|---|---|
| Bool | 1 | true arba false |
| unsigned short int | 2 | 0 … 65533 |
| short int | 2 | -32768 … 32767 |
| unsigned long int | 4 | 0 … 4294967295 |
| long int | 4 | -2147483648 … 2147483647 |
| int (16 bit) | 2 | -32768 … 32767 |
| int (32 bit) | 4 | -2147483648 … 2147483647 |
| unsigned int (16 bit) | 2 | 0 …65535 |
| unsigned int (32 bit) | 4 | 0 …4294697295 |
| char | 1 | 0 …256 |
| float | 4 | 1.2e-38 …3.4e+38 |
| double | 8 | 2.2e-308 … 1.8e+308 |
| void | 2 ..4 | |

# Operatorius *typedef*

**typedef** ***egzistuojantis_tipas naujas_tipas;***

**Apibrėžia naują kintamojo tipą, naudojant egzistuojantį (bazinį) duomenų tipą.**

**Argumentai.**
*egzistuojantis_tipas* - bazinis ar anksčiau apibrėžtas duomenų tipas.
*naujas_tipas* - naujai apibrėžtas duomenų tipas.

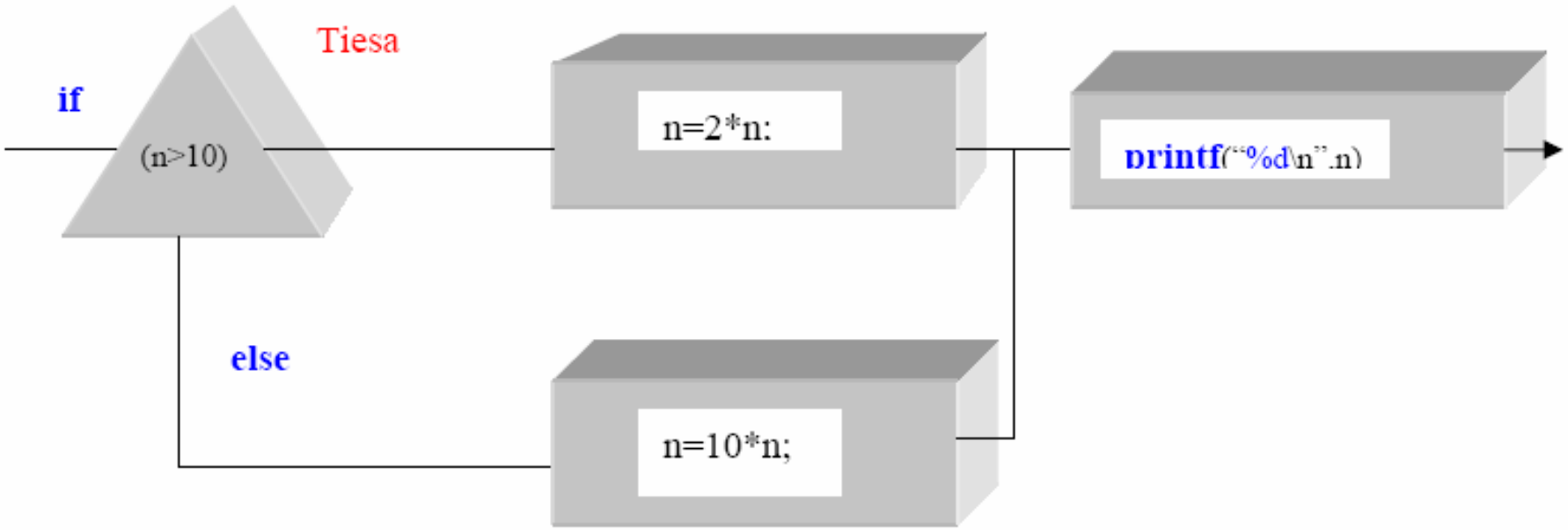
---

```
/*Apibrėžtas naujas tipas */
typedef char C;
typedef unsigned int WORD;
typedef char * string_t;
typedef char field [50];
```

```
/* Panaudojimas */
C achar, anotherchar, *ptchar1;
WORD myword;
string_t ptchar2;
field name;
```

# Loginės operacijos

Operatoriai **if ; if else; switch**
Operacijos > >= <= < == != && || ? :



# Ciklo operatoriai

Operatoriai **while; for; do while**

**while**( sąlyga)
  operatorius(iai);  //veiksmai, kurie bus atliekami

Šio ciklo viduje, turime nurodyti kaip turi keistis sąlygoje įeinantis kintamasis. Kitaip, ciklas bus begalinis.

Cikle **for** iš karto nurodomos ciklo pradinės ir galinės sąlygos, ciklo atlikimo žingsnis.

```
for(a=1; a<=10; a++)
  printf("Man puikiai sekasi!\n");
```

**do**
  operatorius(iai);
**while** (sąlyga)

# Funkcijos

**Funkcija** – tai programos kodo dalis, turinti savo pavadinimą. Ši dalis gali būti iškviesta iš bet kurios kitos programos.

Kodėl naudojamos funkcijos?

- Sukuria aiškia programos struktūrą;
- Skaido programą į dalis;
- Leidžia lengvai panaudoti programos kodo dalį keletą kartų.

Pavyzdys

```
#include <stdio.h>
void starline ();
int main()
{ starline ();
   printf( "Tipas      Skaiciu ribos\n");
   starline ();
   printf( " int        -32768... 32767\n");
   return 0;
}
```

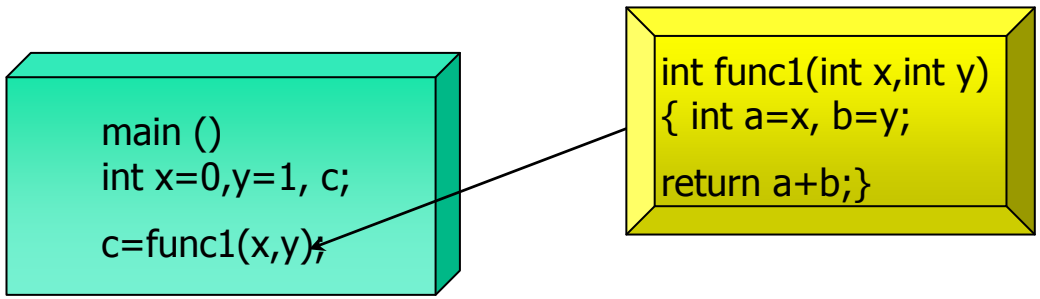
```
void starline ()
{ for (int i =0; i<30;i++)
   printf("*");
   printf("\n");
}
```

# Funkcijos iškvietimas

Funkcija iškviečiama nurodant **funkcijos pavadinimą** ir **argumentų sąrašą**, kurie paprastai būna kintamieji ar konstantos, perduodami funkcijai tolesniam panaudojimui.

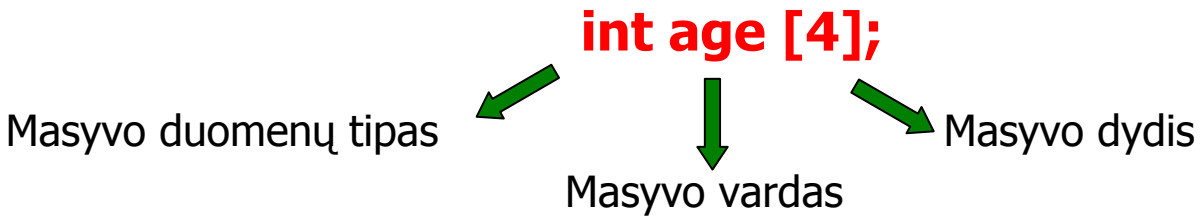
Funkcijos gali iškviesti viena kitą, pagal sudarytą herarchiją. Funkcija gali iškviesti ir pačią save. Tokia konstrukcija vadinama rekursija.

Funkcijos kuria (arba ne) perduodamų duomenų kopiją. Jei kopija kuriama, toks duomenų perdavimas vadinamas **reikšmės perdavimu**, jei ne – **nuorodos perdavimu**.

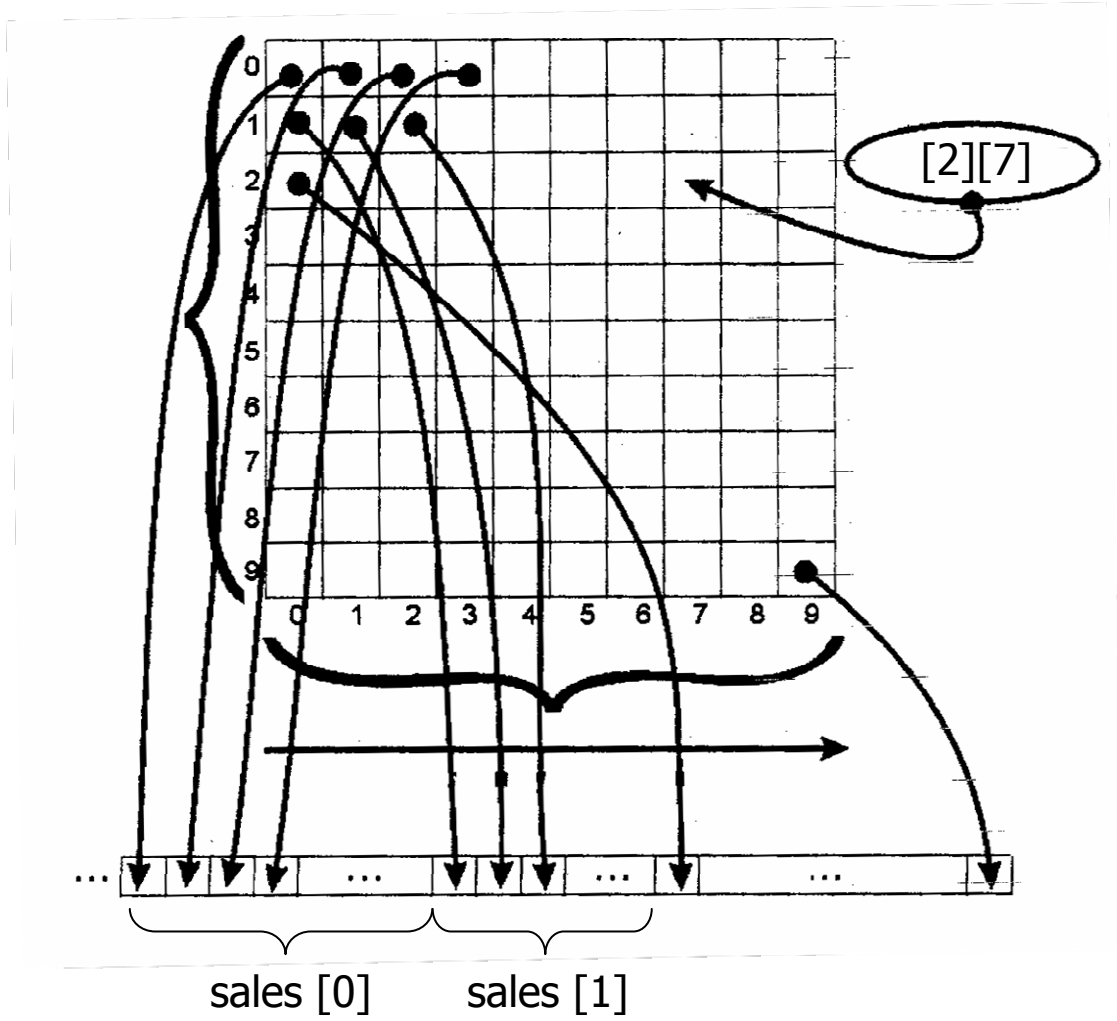


# Masyvo pavyzdys

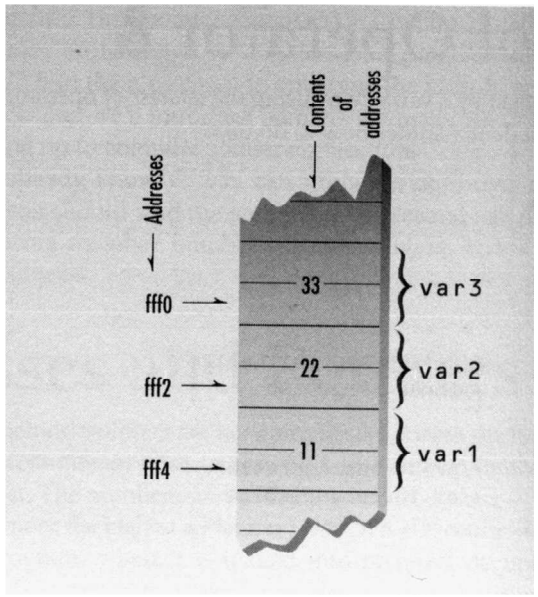
```
#include <stdio.h>
int main ()
{ int age[4];                    // sukuriamas masyvas iš 4 int tipo elementų
     for (int  j=0; j<4; j++)
       { printf ("Iveskite amziu: ");
          scanf ("%d", &age[j]);
        }
     for ( j=0; j<4; j++)
        printf ("Ivesktas amzius: %d ", age[j]);
 return 0;

}
```

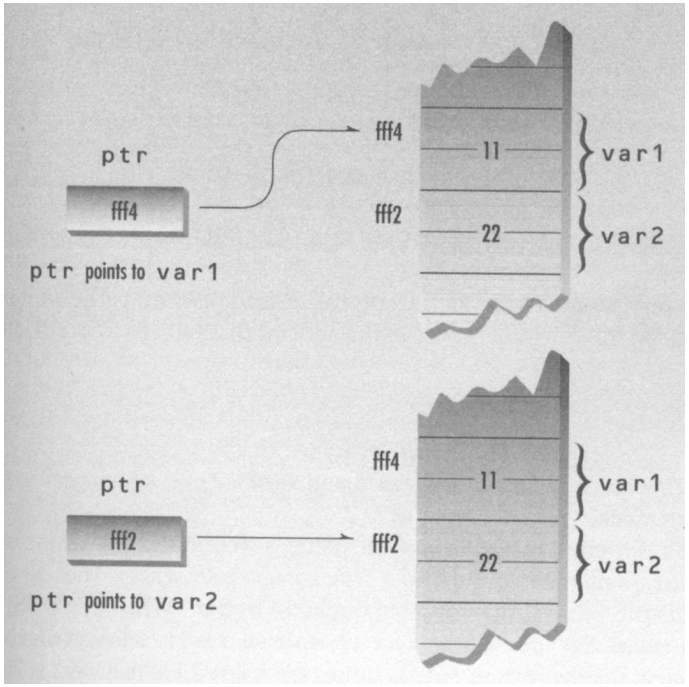


# Dvimačiai masyvai



# Rodyklės (pointers) ir adresai




*Kintamasis* – tai atminties segmentas turintis vardą. Pvz. var1 = 11

Kompiuteris pasiekia savo atmintį ne per kinamųjų vardus, o per **adresus**.

**Rodyklė** – tai kintamasis, kuriame saugomas atminties adresas.
Kitaip dar sakoma, kad rodyklė rodo į kintamąjį, kurio adresą ji saugo.

Rodyklė turi būti apibrėžiama, kaip ir bet kuris kitas kintamasis, pvz. **int *ptr;**

# Masyvai ir rodyklės

Ryšys tarp masyvų ir rodyklių:

&a[0] < &a[1] < ... < &a[9]

&a[1] = &a[0] + 1
&a[i] = &a[i-1] + 1

Kadangi *a* tai rodyklė:

&a[i] tas pats kaip a+i
a[i] tas pats kaip *(a+i)

**Pavyzdys**

```
int i, a[10] ;
for (i = 0; i < 10; i++)
   printf("%d  ", a[i]);

for (i = 0; i < 10; i++)
   printf("%d  ", *(a+i));
```

# Rodyklės ir funkcijos

Duomenys funkcijoms perduodami per parametrų sąrašus tokiais būdais:

- kopijuojant **reikšmę**;
- naudojant **rodykles**;
- naudojant **nuorodas**.

```
#include <stdio.h>
void main()
{  void centm (double&);
   double var = 10.0;
   printf ("var = %lf  cnt\n", var);
   centm (var);
   printf ("var = %lf  coliu\n", var);
}

void centm(double &v)
{ v = v/2.54; }
```

```
#include <stdio.h>
void main()
{  void centm (double*);
   double var = 10.0;
   printf ("var = %lf  cnt\n", var);
   centm (&var);
   printf ("var = %lf  coliu\n", var);
}
void centm(double *ptr)
{ *ptr = *ptr/2.54; }
```

# Masyvų perdavimas funkcijose

Masyvų perdavimui į funkcijas dažniausiai naudojamos rodyklės.

**Masyvo pavadinimas – tai rodyklė į 1-ąjį masyvo elementą**, kurios reikšmė 1-ojo elemento adresą.

```
#include <stdio.h>
void main()
{  void centm (double*);
   double array[5] = {10.0, 12.4, 3.5, 7.2, 22.4};
   centm (array);
   for (int i = 0; i  < 5; i++)
       printf ("array[%d] = %lf  coliu\n", i, array[i]);
}

void centm(double *ptr)
{ for (int j = 0; j < 5; j++)
      ptr[j] /=2.54;        // masyvo elementą daliname
 }
```

# Dinaminis atminties išskyrimas

Funkcijos **malloc()** ir **calloc()** grąžina rodyklę į nurodyto tipo atmintį, todėl naudojant šią funkciją nurodomas rodyklės tipas.

```
int* pInt =(int*) malloc(40);   float* pfl= (float*) calloc(4, 4);
```

Atmintis atlaisvinama naudojant funkciją **free().**

**Sintaksė:**

```
void free (void *block);
```

čia **block* – rodyklė į atminties bloką, kuris turi būti atlaisvintas.

**Pavyzdys**

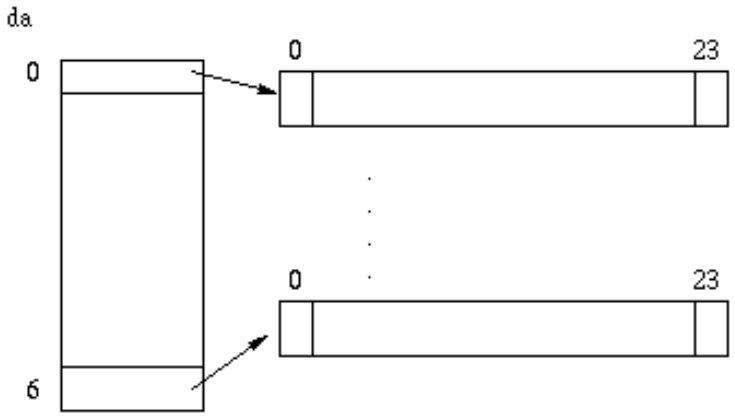
```
#include <stdlib.h>
  int *ip, ar[100];
  ip = (int *) malloc( sizeof ar );
  free (ip);
```

# Dinaminis masyvas

```
#include <stdio.h>
#include <stdlib.h>
int main()
{ int N, *ptr;
  printf( "Iveskite masyvo elementu skaiciu: ");
  scanf ("%d", &N);
  ptr = (int*) malloc(N*sizeof(int) );      // išskiriama atmintis masyvui
   if ( ptr == NULL)
    {  printf ("Truksta atminties \n");
     return 1;}

  for (int i = 0; i < N; i++)
     *(ptr+i) = i;                          // galima naudoti ir prt[i]

  for ( i = 0; i < N; i++)
     {printf( "%d elementas = %d\n", i, *(ptr) );
       ptr++; }
  free (ptr);
  return 0; }
```

# Dinaminis dvimatis masyvas

```
#include <stdio.h>
#include <stdlib.h>

// užduotis: išskirti atmintį masyvui da [n][m]
void main()
{  int **da;

// tegul m ir n reikšmės nuskaitomos
// pvz. iš failo ir n=7; m=24;

   da = (int**) malloc(n*sizeof(int*));

for (int i = 0; i< n; i++)
   da[i] = (int*) malloc(m*sizeof(int));
}
```



# Pavyzdys

**Užduotis**: Suskaičiuojuoti matricos kiekvienos eilutės elementų suma. Rezultatai surašomi į vienmatį masyvą.

```
#include <stdio.h>

typedef int mas [10];
typedef int matr[10][15];

int Duomenys(matr, int *, int *);    // Duomenų įvedimas
void Sp (mas, int);                  // Spausdinimas
int Suma (mas, int);
```

# Pavyzdys

```
void main() {
matr A; mas S;
int n, m, i, j;
Duomenys(A, &n, &m);                    // Duomenų įvedimas klaviatūra

for (i=0; i<n; i++)                     // Matricos spausdinimas
{ for (j=0; j<m; j++)
     printf( "%d ", A[i][j] );
 printf( "\n" );
}
for (i=0; i<n; i++) {                   // Matricos spausdinimas su funkcija
printf("Eilute Nr.%d", (i+1));
Sp(A[i], m); }                          // Vienos eilutės spausdinimas

for (i=0; i<n; i++) S[i] = Suma(A[i], m);
Sp (S, n); }                            // Suformuoto masyvo spausdinimas
}
```

# Pavyzdys

```
int Duomenys(matr D, int *n, int *m)
{  FILE  F;
 F = fopen( "duomenys.txt", "r" );
 if ( F == NULL ) printf( "Failas neatidarytas\n");
  else{
      printf(" Failas sekmingai atidarytas !!\n");
      fscanf(F,"%d%d\n", n, m);                  //eilučių ir stulpelių skaičius
    for(int i=0; i<*n; i++)
      {for(int j=0; j<*m; j++)
          fscanf(F,"%d", &D[i][j]);
       }
fclose(F);
      }
}
```

# Pavyzdys

```
// Spausdinimas
void Sp(mas x, int n) {
int i = 0;
while (i < n)
 printf("%d ", x[i++]);
printf ("\n"); }
```

```
// Masyvo elementų sumos skaičiavimas
int Suma(mas D, int k) {
int i, Sk=0;
for (i=0; i<k; i++) Sk += D[i];
return Sk; }
```