Curs 4

functii (2), pointeri & tablouri

lucrare de control

```
int fun(int i){   return (i++); }

int main(){

    int i = fun(10);

    printf("%d\n", --i);

    return 0;

}
```

lucrare de control

```
int main(){
        int a=5, b=6;
        printf(“%d+%d=%d\n”,++a,b++,a+b);
    return 0;
}
```

asa nu

```
int a=5, b=6;

int f1();

int suma(){

        return a+b;

}

int main(){

                f1();

                printf(“%d”,suma());

   return 0;
}
```

ce intoarce o functie care nu intoarce nimic?

```
int f1(int param)
{
        if (param%2==1)
                return 1;

}


int main()
{
        int i,r;
        srand(time(NULL));
        for (i=0;i<10;i++)
        {
                r=rand()%2;
                printf("%d %d\n",f1(r),r);
        }
        return 0;

}
```

```
1 1
1 1
1 1
0 0
0 0
1 1
0 0
0 0
0 0
0 0
```

ce intoarce o functie care nu intoarce nimic?

```
int f1(int param)
{
        if (param%2==0)
                return 1;

}
```

```
1 0
1 0
1 0
1 0
1 1
1 1
1 1
1 1
1 1
1 0
```

ce intoarce o functie care nu intoarce nimic?

- R: unspecified behaviour
- cateodata se potriveste sau compilatorul “ghiceste” ce ar trebui intors
- trebuie sa folosim explicit return atunci cand functia trebuie sa intoarca un rezultat
- Raspuns complicat [ADVANCED TOPIC]: http://stackoverflow.com/questions/4644860/function-returns-value-without-return-statement
  - Cand nu se folosesc optimizari, valoarea returnata default este cea din registrul **eax** pe masinile x86

fisiere header

- contin **declaratii de functii** si **definitii de macro-uri** (de ex. constante) partajate de mai multe fisiere sursa
- au **extensia .h**
- pentru proiecte mici, un fisier header este in general suficient
- sunt incluse in fisierele sursa folosind directiva #include
  - **#include “fisierHeader.h”**
    - atentie – numele se pune intre ghilimele pentru fisierele header definite de noi
  - **#include <fisierHeaderSistem.h>**
    - Pentru includerea de fisiere header sistem. Acestea sunt cautate intr-o lista de directoare standard

fisiere header (2)

```
//functii.c
#include “functii.h”
int f1(int a)
{….
}
void f2()
{
….
}
```

```
//progmain.c
#include “functii.h”
int main()
{
 f1(5);
f2();
}
```

```
:~/programare/c4$ gcc progmain.c functii.c
```

```
//functii.h
int f1(int);
void f2();
```

- fisierele header nu sunt mentionate explicit la compilare
  - lipsa prototipului unei functii apelate = eroare la compilare
  - lipsa implementarii unei functii apelate si al carui prototip exista – eroare la linkare
- se compileaza fisierele sursa
- atentie – 1 singur main!

clase de stocare - extern

- extern – pentru variabile folosite in mai multe fisiere
- se declara variabila de tip extern in fisierul header
  - extern int x;
- intr-unul din fisiere se va declara si variabila
  - int x;
- declararea variabilei in fisierul .h nu presupune rezervarea de spatiu de memorie
- acest lucru se face doar in fisier .c prin declarare fara “extern”
- Pentru detalii [ADVANCED TOPIC]: http://stackoverflow.com/questions/1433204/how-do-i-use-extern-to-share-variables-between-source-files-in-c

functii statice

- functiile statice pot fi apelate doar din fisierul in care sunt definite
- un mod prin care pot fi “ascunse” de celelalte fisiere din proiect.

exemplu functii statice

progmain.c

```c
int main()
{
        printf("%d\n", f1(5));
        printf("%d\n", f2(5,2));
        printf("%d\n", f3(5,2));

        return 0;
}
```

functii.h

```c
int f1(int);
int f2(int, int);
static int f3(int, int);
```

```c
int f1(int param)
{
        if (param%2==0)
                return 1;

}

int f2(int p1, int p2)
{
        return f3(p1,p2);
}

static int f3(int x, int y)
{
        return x+y;
}
```

```
/tmp/ccqr8gH4.o: In function `main':
progmain.c:(.text+0x54): undefined reference to `f3'
collect2: ld returned 1 exit status
```

exemplu functii statice (2)

progmain.c

```c
int main()
{
        printf("%d\n", f1(5));
        printf("%d\n", f2(5,2));
//      printf("%d\n", f3(5,2));

        return 0;
}
```

functii.h

```c
int f1(int);
int f2(int, int);
static int f3(int, int);
```

```c
int f1(int param)
{
        if (param%2==0)
                return 1;

}

int f2(int p1, int p2)
{
        return f3(p1,p2);
}

static int f3(int x, int y)
{
        return x+y;
}
```

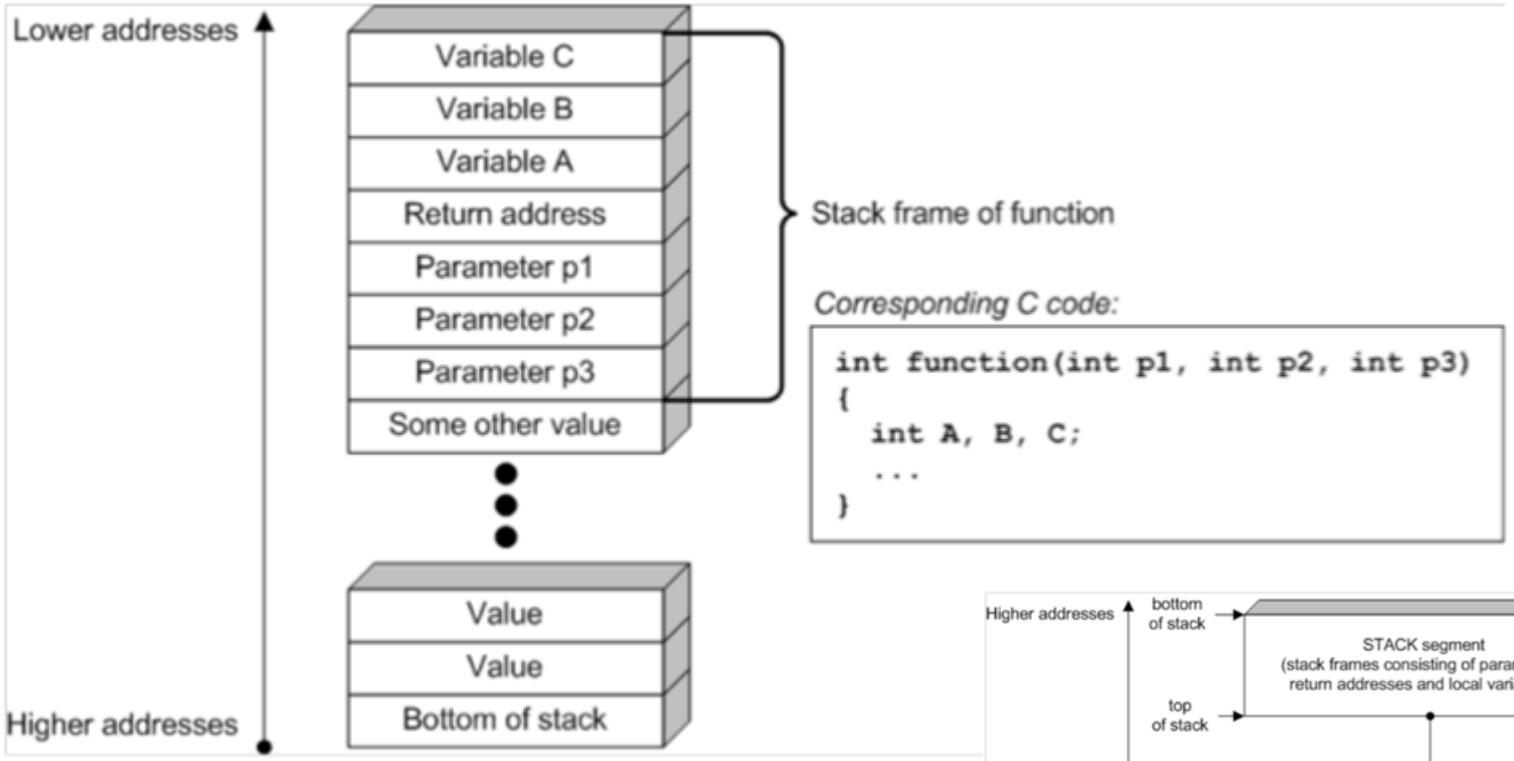
```
1
7
vlad@hoth:~/programare/c4$ gcc progm
```

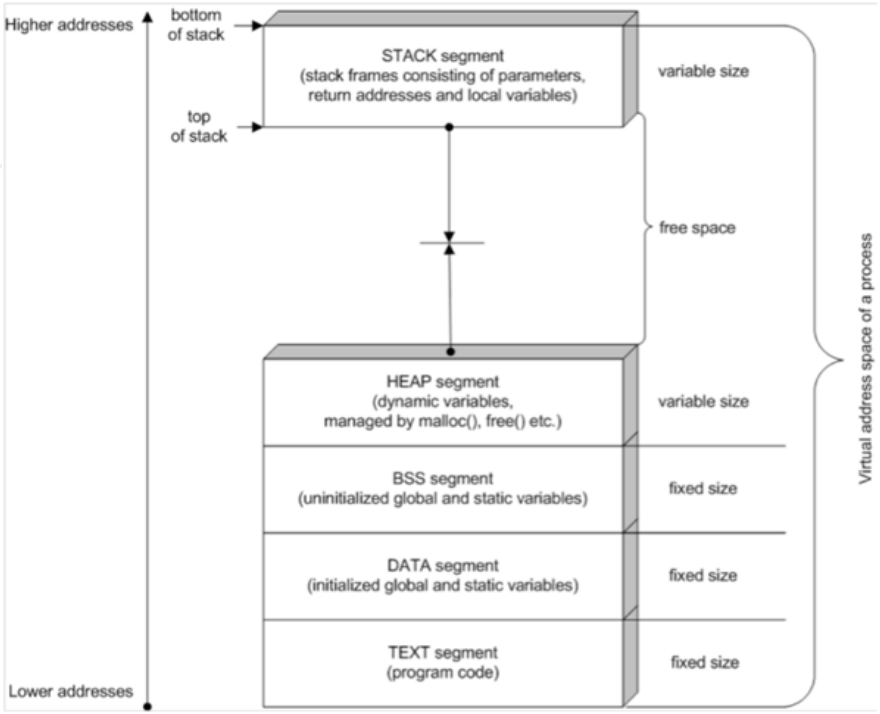
Functii recursive

- functie recursivă = funcție care se auto-apelează
- necesită minim un punct de ieșire (ramură la care se ajunge sigur și pe care funcția nu se mai apelează)
- tipuri de recursivitate
  - head – apelul recursiv se face la inceputul functiei
  - tail – apelul recursiv se face la sfarsit
- exemple de funcții recursive
  - factorial, fibonacci, putere (v. la tablă ☺)
- Exista si recursivitate cu mai mult de 2 functii: recursivitate mutuala (https://en.wikipedia.org/wiki/Mutual_recursion)

Ce se intampla cand apelam o functie?

- se rezervă spațiu pe stivă pentru argumentele funcției și variabile locale
- se copiază argumentele funcției în această zonă de memorie
- se execută funcția
- rezultatul execuției este copiat într-o valoare ce va fi întoarsă
- se aduce stiva la starea precedentă
- programul revine in locul în care a fost apelată funcția





http://www.drdobbs.com/security/anatomy-of-a-stack-smashing-attack-and-h/240001832

Recursivitate vs. iterativitate

- recursivitate
  - pro: de multe ori cod mai elegant, mai lizibil
  - contra: overhead mare datorat de apelurile repetate ale functiilor

Pointeri

- pointer = variabila ce contine adresa unei variabile
- declaratie:
  - tip * nume_variabila;
  - char * adrC;
    - declara o variabila de tip adresa care va puncta catre un element de tip char

Pointeri – operatori

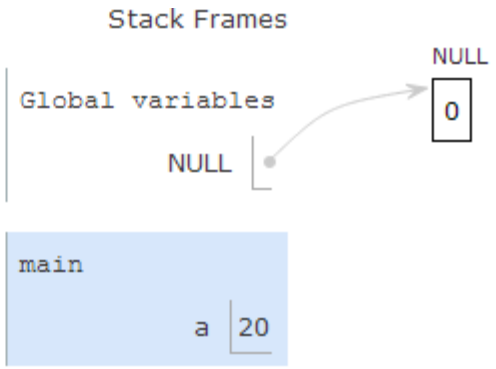
- operatorul de referentiere/adresare &
  - se aplica asupra unei variabile.
  - rezultatul este un pointer
  - int x,*p;
  - p=&x;

Operatorul de dereferentiere

- operatorul de dereferentiere
  - se aplica unui pointer
  - intoarce valoarea variabilei spre care puncteaza pointerul
  - int x,*p,y; p=&x;
  - *p=5;//scrie 5 la adresa p
  - y=*p;//atribuie variabilei y valoarea de la adresa p

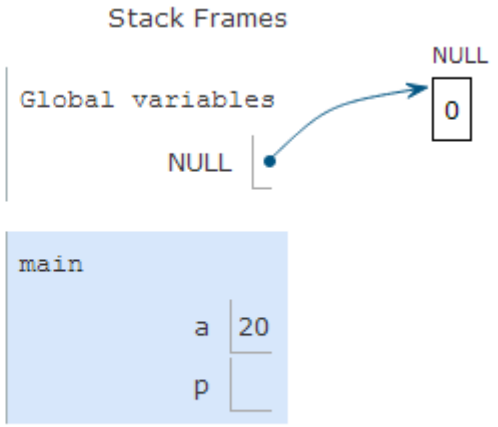
Exemplu operatori

```
1  #include<stdio.h>
2
3  int main()
4  {
5      int a = 20;
6      int *p;
7      p = &a;
8      *p = 30;
9      return 0;
10 }
```

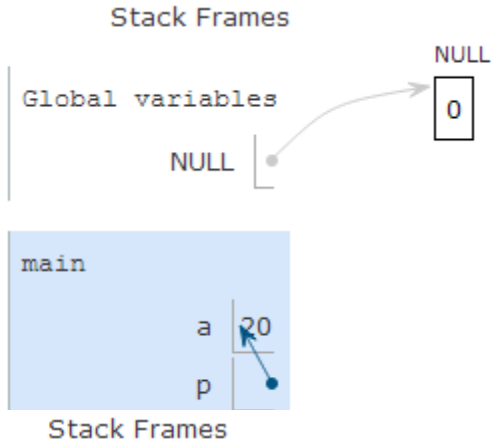


- p este neinitializat=poate puncta catre orice adresa de memorie
- nu putem folosi pointerii care nu sunt initializati

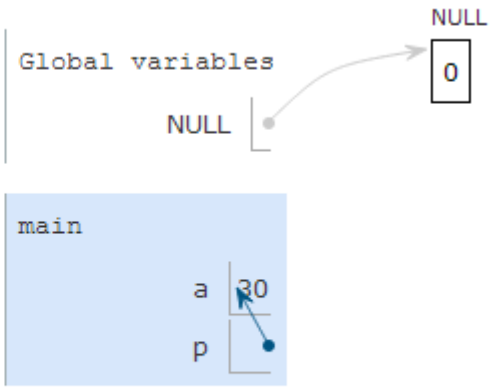
```c
#include<stdio.h>

int main()
{
    int a = 20;
    int *p;
    p = &a;
    *p = 30;
    return 0;
```




```c
#include<stdio.h>

int main()
{
    int a = 20;
    int *p;
    p = &a;
    *p = 30;
    return 0;
```




```c
#include<stdio.h>

int main()
{
    int a = 20;
    int *p;
    p = &a;
    *p = 30;
    return 0;
}
```

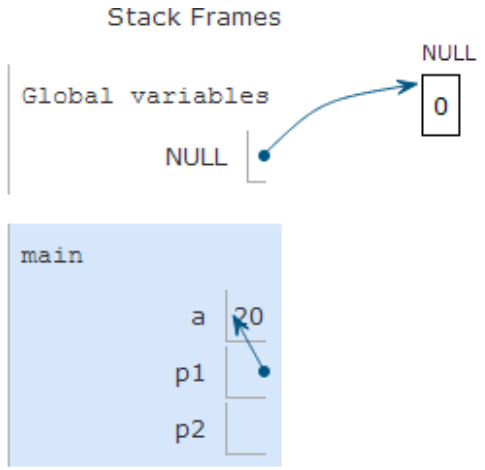


Edit code

Aritmetica pointerilor

- putem aduna/scadea numere intregi cu pointeri
  - adunarea a x unitati unui pointer semnifica deplasarea in memorie cu un numar de octeti= x*sizeof(tip pointer)
  - int *p1,*p2,x; p1=&x;p2=p1+1;
  - p2 va puncta catre o zona de memorie aflata cu 4 octeti dupa p1.

```
1  #include<stdio.h>
2
3  int main()
4  {
5      int a = 20;
6      int *p1,*p2;
7      p1 = &a;
8      p2=p1+1;
9      printf("p1=%p\np2=%p\n",p1,p2);
10     return 0;
11 }
```



Edit code

Program output:

```
p1=0x7fb564e1d1c0
p2=0x7fb564e1d1c4
```

diferenta intre p1 si p2 este de 4 octeti – dimensiunea unui int.

Aritmetica pointerilor

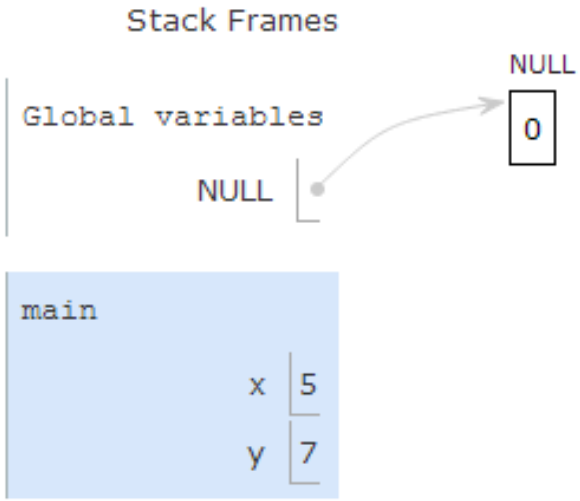
- putem scadea 2 pointeri **de acelasi tip**
- valoarea intoarsa reprezinta numarul de unitati dintre cei doi pointeri
- **nu putem aduna 2 pointeri**
- tipul rezultatului
  - pointer=pointer+intreg
  - pointer=pointer-intreg
  - intreg=pointer-pointer
  - eroare de compilare = pointer+pointer

Transmiterea parametrilor prin adresa (2)

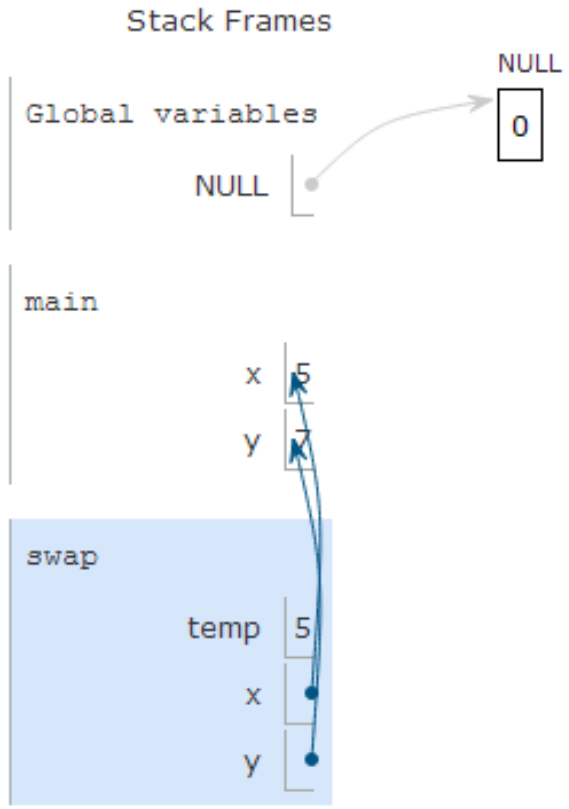
- reminder:
  - int f1(int *p);
  - se transmite functiei adresa unei variabile
  - se poate modifica valoarea de la adresa data
  - valoarea ramane modificata la parasirea functiei
- functia primeste ca parametru un pointer
- pointerul trebuie sa fie initializat inainte de a fi utilizat

exemplu (swap schimba 2 valori intre ele)

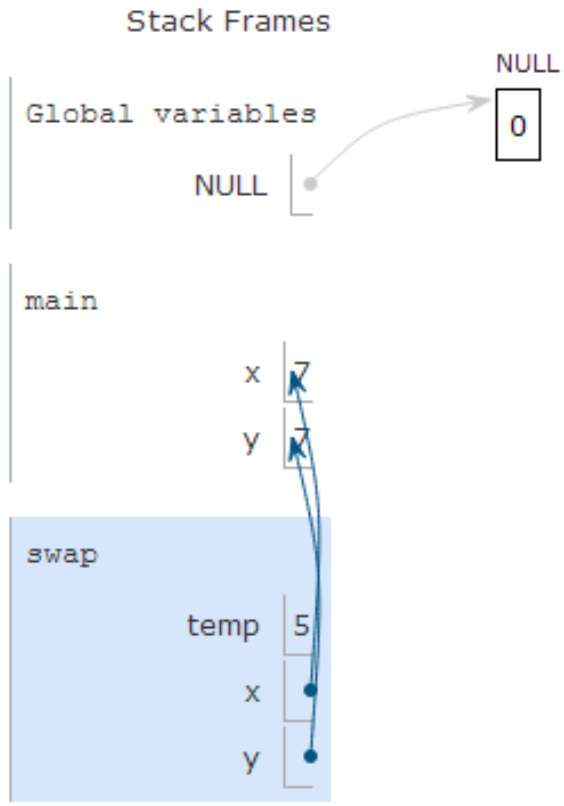
```
#include <stdio.h>

void swap(int *x, int*y)
{
    int temp=*x;
    *x=*y;
    *y=temp;
}

int main()
{
    int x=5,y=7;
    printf("x=%d &x%p\ny=%d &y=%p\n",x,&x,y
    swap(&x,&y);
    printf("x=%d &x%p\ny=%d &y=%p\n",x,&x,y
    return 0;

}
```



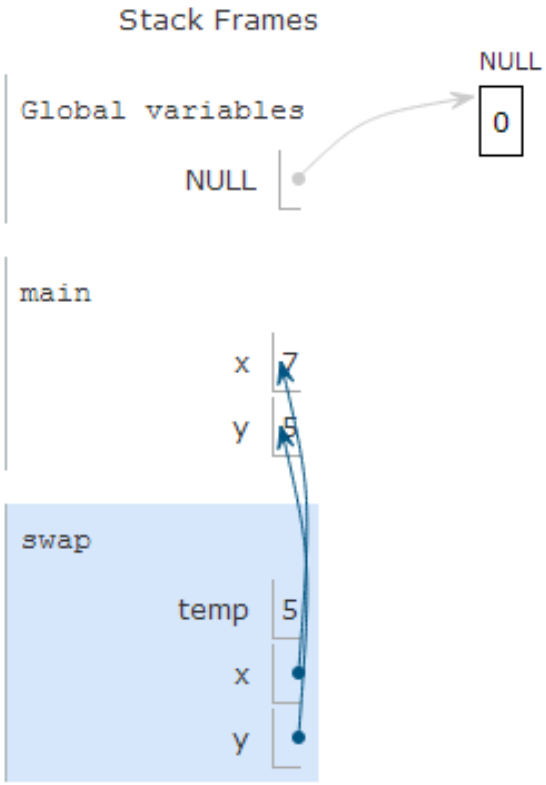
```
1  #include <stdio.h>
2
3  void swap(int *x, int*y)
4  {
5      int temp=*x;
6      *x=*y;
7      *y=temp;
8  }
9
10 int main()
11 {
12     int x=5,y=7;
13     printf("x=%d &x%p\ny=%d &y=%p\n",x,&x,y
14     swap(&x,&y);
15     printf("x=%d &x%p\ny=%d &y=%p\n",x,&x,y
16     return 0;
17
18 }
```



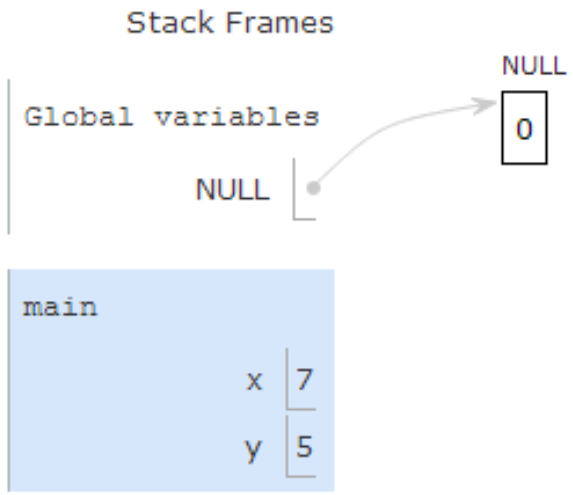
```
1  #include <stdio.h>
2
3  void swap(int *x, int*y)
4  {
5      int temp=*x;
6      *x=*y;
7      *y=temp;
8  }
9
10 int main()
11 {
12     int x=5,y=7;
13     printf("x=%d &x%p\ny=%d &y=%p\n",x,&x,y
14     swap(&x,&y);
15     printf("x=%d &x%p\ny=%d &y=%p\n",x,&x,y
16     return 0;
17
18 }
```



```
1  #include <stdio.h>
2
3  void swap(int *x, int*y)
4  {
5      int temp=*x;
6      *x=*y;
7      *y=temp;
8  }
9
10 int main()
11 {
12     int x=5,y=7;
13     printf("x=%d &x%p\ny=%d &y=%p\n",x,&x,y
14     swap(&x,&y);
15     printf("x=%d &x%p\ny=%d &y=%p\n",x,&x,y
16     return 0;
17
18 }
```



```
1  #include <stdio.h>
2
3  void swap(int *x, int*y)
4  {
5      int temp=*x;
6      *x=*y;
7      *y=temp;
8  }
9
10 int main()
11 {
12     int x=5,y=7;
13     printf("x=%d &x%p\ny=%d &y=%p\n",x,&x,y
14     swap(&x,&y);
15     printf("x=%d &x%p\ny=%d &y=%p\n",x,&x,y
16     return 0;
17
18 }
```



Tablouri/vectori

- vector = colectie de elemente identificate prin indici
- in C primul element are indicele 0;
- indicii sunt numere intregi intre 0 .. (length(vector) – 1)
- elementele vectorului au acelasi tip
- int V[10];
  - declaratia unui vector de 10 elemente de tip intreg

declarare vectori

- numetip numevariabila[dimensiune_vector]
- dimensiune_vector trebuie sa fie o constanta de tip intreg
- exemplu:
  - int v[20]; char s[15]; float nr[100];

declarare vectori

- se recomanda utilizarea constantelor definite cu #define
  - #define MAX 100
  - …
  - int v[MAX];

Caracteristici vector

- int v[MAX];
- v – adresa primului element al vectorului
- MAX – nr maxim de elemente pe care il poate avea vectorul
- Nr efectiv de elemente utilizate – se retine intr-o alta variabila;

Vectori – accesul la elemente

- numeVector[indice] acceseaza valoarea elementului cu indicele = indice
- pentru ca indexarea incepe de la 0 numeVector[indice] va fi al indice+1 element
- numeVector[3]= al 4-lea element
  - (dupa numeVector[0], numeVector[1], numeVector[2])

Initializarea vectorilor

- la declarare vectorii se pot initializa in 3 moduri
  - int v[10]={0};
    - toate elementele vectorului vor fi initializate cu 0;
  - int v[10]={0,1,2,3,4,5,6,7,8,9};
    - se initializeaza explicit toate elementele vectorului
    - elementele vor fi initializate cu numerele de la 0-9 (v[0]=0, v[1]=1,…

Initializarea vectorilor

- int v[10]={valoare_0,…, valoare_n}
  - v[0]=valoare_0, …, v[n]=valoare_n, celelalte sunt initializate cu 0

vector neinitializat

```
#include<stdio.h>
int main()
{
        int a[10],i;
        for (i=0;i<10;i++)
                printf("a[%d]=%d\n",i,a[i]);
        return 0;
}
```

```
a[0]=468814728
a[1]=32626
a[2]=4195616
a[3]=0
a[4]=0
a[5]=0
a[6]=4195296
a[7]=0
a[8]=797824192
a[9]=32767
```

vector initializat cu 0

```
#include<stdio.h>
int main()
{
        int a[10]={0},i;
        for (i=0;i<10;i++)
                printf("a[%d]=%d\n",i,a[i]);
        return 0;
}
```

```
a[0]=0
a[1]=0
a[2]=0
a[3]=0
a[4]=0
a[5]=0
a[6]=0
a[7]=0
a[8]=0
a[9]=0
```

vector cu toate elementele initializate explicit

```
#include<stdio.h>
int main()
{
        int a[10]={0,1,2,3,4,5,6,7,8,9},i;
        for (i=0;i<10;i++)
                printf("a[%d]=%d\n",i,a[i]);
        return 0;
}
```

```
a[0]=0
a[1]=1
a[2]=2
a[3]=3
a[4]=4
a[5]=5
a[6]=6
a[7]=7
a[8]=8
a[9]=9
```

vector cu elemente initializate partial explicit si restul elementelor initialiate cu 0

```
#include<stdio.h>
int main()
{
        int a[10]={4,5,6},i;
        for (i=0;i<10;i++)
                printf("a[%d]=%d\n",i,a[i]);
        return 0;
}
```

```
a[0]=4
a[1]=5
a[2]=6
a[3]=0
a[4]=0
a[5]=0
a[6]=0
a[7]=0
a[8]=0
a[9]=0
```

vectori si pointeri

- vector in C = adresa primului element

```c
int main()
{
        int v[10]={0};
        int *p=v;
        printf("p=%p, v=%p\n",p,v);
        return 0;
}
```

```
p=0x7fffd0017670, v=0x7fffd0017670
```

adresarea elementelor din vector poate fi facuta cu ajutorul pointerilor

```c
int main()
{
        int v[10]={0};
        int *p=v,i;
        srand(time(NULL));
        printf("p=%p, v=%p\n",p,v);
        for(i=0;i<10;i++)
                v[i]=rand()%100;

        for(i=0;i<10;i++,p++)
                printf("v[i]=%i,*p=%i, v+i=%p, p=%p\n",v[i],*p,v+i,p);


        return 0;
```

```
p=0x7fff61a64b70, v=0x7fff61a64b70
v[i]=11,*p=11, v+i=0x7fff61a64b70, p=0x7fff61a64b
v[i]=25,*p=25, v+i=0x7fff61a64b74, p=0x7fff61a64b
v[i]=84,*p=84, v+i=0x7fff61a64b78, p=0x7fff61a64b
v[i]=99,*p=99, v+i=0x7fff61a64b7c, p=0x7fff61a64b
v[i]=89,*p=89, v+i=0x7fff61a64b80, p=0x7fff61a64b
v[i]=32,*p=32, v+i=0x7fff61a64b84, p=0x7fff61a64b
v[i]=20,*p=20, v+i=0x7fff61a64b88, p=0x7fff61a64b
v[i]=26,*p=26, v+i=0x7fff61a64b8c, p=0x7fff61a64b
v[i]=97,*p=97, v+i=0x7fff61a64b90, p=0x7fff61a64b
v[i]=12,*p=12, v+i=0x7fff61a64b94, p=0x7fff61a64b
```

parcurgerea vectorilor

- Cu indici
  - Intre 0 si nr efectiv de elemente

```
int I,n, v[10];
srand(time(NULL));
n=1+rand()%9; //generam un numar intre 1 si 10
//n va fi lungimea efectiva a vectorului
for (i=0;i<n;i++)
//parcurgerea tipica – intre 0 si lungimea efectiva
    //generam aleator elementele vectorului
    v[i]=rand()%100;
```

Parcurgerea vectorilor

- Cu pointeri
  - Se identifica adresa aflata la sfarsitul vectorului (=v+n)
  - Se parcurge intre v si acea adresa

```
int n, v[10], *p,*s;
//vector initializat
for (p=v, s=v+n;p<s;p++)
      *p=rand()%100;
```

Vectori vs. pointeri

```c
int main()
{
        int x[5],*p=x;
        printf("x=%p p=%p\n",x,p);
        printf("x+1=%p p+1=%p\n",x+1,p+1);
        printf("&x=%p &p=%p\n",&x,&p);
        printf("&x+1=%p &p+1=%p\n",&x+1,&p+1);
        printf("sizeof(x)=%d, sizeof(p)=%d\n",sizeof(x), sizeof(p));
        return 0;
}
```

```
x=0x7fff43ff60d0 p=0x7fff43ff60d0
x+1=0x7fff43ff60d4 p+1=0x7fff43ff60d4
&x=0x7fff43ff60d0 &p=0x7fff43ff60c8
&x+1=0x7fff43ff60e4 &p+1=0x7fff43ff60d0
sizeof(x)=20, sizeof(p)=8
```

&x=x

x nu poate fi modificat

x++ - err de compilare

Vectori de lungime variabila

```
int n;
printf(“introdu n\n”);
scanf(“%d”,&n);
int v[n];
// sintaxa corecta in c99/c11
//nu exista in ANSI C
// se rezerva spatiu pe stiva in momentul executiei
```

// [ADVANCED TOPIC] http://www.drdobbs.com/the-new-cwhy-variable-length-arrays/184401444