Pokazivači

predavač: Nadežda Jakšić

Programiranje programski jezik C

Pokazivači

- koncept pokazivača prisutan je u mnogim programskim jezicima, negde eksplicitno kao u **C**-u, a negde implicitno, kao što je to programskom jeziku **Java** u kome su pokazivači "skriveni" i ne koriste se direktno
- u **C**-u se neke tehnike mogu implementirati lakše korišćenjem pokazivača, a neke se mogu implementirati samo korišćenjem pokazivača (dinamička alokacija memorije)
- **C** koristi pokazivače eksplicitno sa:
  - nizovima
  - strukturama
  - funkcijama
- mogu biti veoma moćan alat koji omogućava pisanje kompaktnijeg i efikasnijeg koda; međutim, njihovo nepravilno korišćenje može ozbiljno narušiti čitljivost i funkcionalnost programa

Statička alokacija memorije

- za statičku alokaciju memorije koristi se **stack** memorija; događa se za vreme kompajliranja programa i koristi se za deklaraciju svake promenljive u programu; količinu memorije unapred određuje programer i ona se za vreme izvršavanja programa ne može dinamički povećati po potrebi; ako je deklarsan niz od 50 članova i programu ne treba više od toga sve je u redu, ali šta ako mu zatreba više? (npr. čitanje iz datoteke za koju se ne zna unapred koliko je velika)

- **stack** je ograničene veličine i manji je od **heap** memorije; statički alocirana memorija se ne može osloboditi, oslobađa se kada promenljiva izađe iz opsega (**scope**)

Dinamička alokacija memorije

- za dinamičku alokaciju memorije koristi se **heap** memorija; ona se koristi kada veličina nečega što treba staviti u memoriju nije unapred poznata, ili kada je potrebno više memorije nego što **stack** može da pruži
- npr.
- //dinamički alociran blok memorije od 1 MB
  char *a = malloc(1048576); //C način, koristeći funkciju **malloc**
  char *a = new char[1048576]; // C++ način, koristeći operator **new**
- dinamički alocirana memorija se može osloboditi tokom izvršavanja programa kada više nije potrebna, kako bi se koristila za neke druge podatke
- npr.
- //oslobađanje dinamički alocirane memorije
  free(a); //C način, koristeći funkciju **free**
  delete[] a; //C++ način, koristeći operator **delete**
-

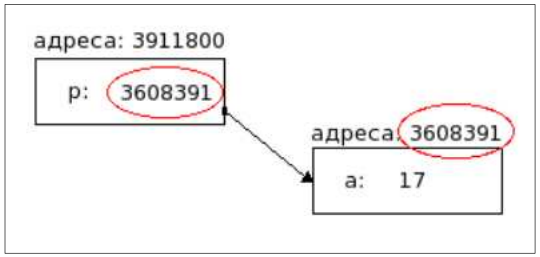
Primer

- u programu postoji struktura koja ima član **imePrezime** koji je statički alociran na 30 znakova ili bajtova (programer pretpostavlja da ime i prezime neće biti duže od 30 znakova uključujući i znak za razmak); ali šta ako se iz datoteke pročita ime koje je duže od toga? kada bi se koristila dinamička alokacija za član **imePrezime** bi se moglo zauzeti tačno onoliko memorije koliko je ime dugačko; time bi program zauzimao manje memorije (ako je ime kraće od 30 znakova ne bi bilo rasipanja memorije jer bi bilo alocirano tačno onoliko koliko je potrebno), a ako je ime duže od 30 znakova program bi alocirao više memorije i ime bi uvek stalo u memoriju

Pokazivač

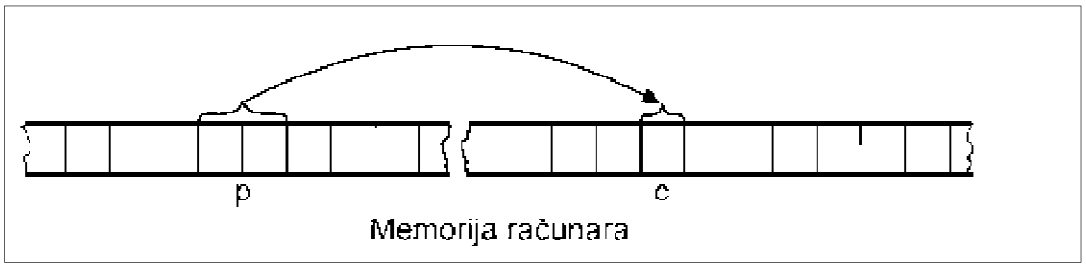
- pokazivač se može definisati kao promenljiva čija je vrednost **adresa** neke druge promenljive

p pokazuje na **a**



- memoriju posmatramo kao niz adresiranih memorijskih ćelija, kojima se može pristupati pojedinačno ili u povezanim grupama; najmanja memorijska jedinica kojoj se može pristupiti je **bajt**

- da bismo dobili adresu neke promenljive u memoriji, koristimo operator **&**



Adresa promenljive

- program ispisuje memorijske adresa promenljivih **p1** i **p2**

```
int main ()
{
int p1;
char p2[10];
printf ("Adresa promenljive p1 je %x\n", &p1 );
printf ("Adresa promenljive p2 je %x\n", &p2 );
return 0;
}
```

- jedan od mogućih izlaza ovog programa je:

  Adresa promenljive p1 je 28ff3c
  Adresa promenljive p2 je 28ff20

oznaka **&p1** predstavlja memorijsku adresu promenljive **p1**; kad se program pokrene za svaku promenljivu se rezerviše određeno mesto u memoriji koje ima svoju adresu; ta adresa je broj koji se može ispisati u bilo kom formatu; ovde je ispis brojeva u heksadekadnom formatu koji predstavlja adrese promenljivih u memoriji

Kako se koriste pokazivači

pokazivač se koristi na sledeći način:

1. deklariše se promenljiva pokazivač
2. dodeli se promenljivoj pokazivač adresa neke promenljive na koju će da pokazuje (referenciranje)
3. pristupa se vrednosti promenljive na koju pokazivač pokazuje (dereferenciranje)

čemu služi pokazivač

- može da menja vrednost (tj. može da pokazuje na razne lokacije u toku trajanja)
- preko njega se može dobiti vrednost promenljive čiju adresu čuva (na koju pokazuje)
- preko njega se može menjati vrednost promenljive na koju pokazuje

Deklaracija pokazivača

sintaksa:

tip *nazivPromenljive;

- **tip** se odnosi na promenljivu na koju pokazuje pokazivač
- znak ***** (zvezdica) označava da se radi o pokazivaču, a naziv promenljive predstavlja naziv pokazivača
- veličina promenljive pokazivač je **4** bajta

primer:

```
int *ip, k; //ip je pokazivač na int a k je običan int (vrednost svake promenljive koja je pokazivač je broj koji predstavlja memorijsku adresu - iako su adrese celi brojevi, pokazivački tipovi se razlikuju od celobrojnih)
double *dp; //dp je pokazivač na double
char *k; //k je pokazivač na karakter
```

Dodela adrese pokazivaču

- kada se pokazivač deklariše on ne pokazuje ni na šta; mora mu se dodeliti na šta da pokazuje pre nego što se upotrebi

  p=&c; //**referenciranje**, tj. pokazivaču dodeljujemo adresu neke memorijske lokacije i tada kažemo da **p** "pokazuje na" **c**

- pokazivački i celobrojni tipovi su različiti; ako je data deklaracija

int *pa, a;
pa = a; //ovaj kôd je neispravan
a=pa;
pa = 1234;

ili
int *ip;
*ip=100; //mora da se navede promenljiva na koju će da pokazuje

Pristupanje promenljivoj

- **dereferenciranjem** se preko pokazivača koji pokazuje na neku memorijsku lokaciju pristupa samoj lokaciji i može da joj se menja vrednost ili jednostavno vidi sadržaj
- unarni operator * (operator **dereferenciranja**) se razlikuje od znaka **zvezdica** koji se koristi pri deklaraciji pokazivača

- simbol * se koristi i za označavanje pokazivačkih tipova i za operator dereferenciranja i treba jasno razlikovati ove njegove dve različite uloge
  primer:
  int a=10, *p; //deklaracija pokazivača
  p = &a; //referenciranje - promenljiva **p** pokazuje na **a**
  *p = 5; //dereferenciranje - u lokaciju na koju ukazuje **p** je upisana vrednost **5** pa je i vrednost promenljive **a** postala **5**

Primer 1

```
int main ()
{
int  a=20;   // deklaracija promenljive a
int  *p;     //deklaracija pokazivača p
p = &a;  //referenciranje – promenljivoj pokazivač se dodeljuje adresa
                                promenljive na koju će da pokazuje
printf ("Adresa promenljive a je %x\n",&a);
printf ("Adresa koja se nalazi u pokazivacu je: %x\n", p);
printf ("Vrednost promenljive na koju pokazuje pokazivac, tj.
    promenljive a je: %d\n", *p);
*p=30;
printf ("Nova vrednost promenljive a je: %d\n",a);
return 0;
}
```

Primer 2

promenljiva **x** se nalazi na memorijskoj lokaciji 100, **y** na 200 i **ip** na 1000

```
int x=1, y=2;
int *ip; //ip je pokazivač na ceo broj
ip=&x //x=1; y=2; u ip se upisuje vrednost 100, tj. adresa promenljive x
y=*ip; //promenljivoj y se dodeljuje vrednost promenljive na koju
    pokazuje ip, tj. promenljiva y dobija vrednost promenljive x, tj. y=1
x=ip; //trenutna vrednost pokazivača ip se dodeljuje promenljivoj x, tj.
    x dobija vrednost 100 - prijaviće gresku pri kompajliranju
*ip=3; //dodeljujemo vrednost promenljivoj na koju pokazuje pokazivač
    ip, tj. promenljiva x dobija vrednost 3
```

Primer 2

```
int main ()
{
int x=1, y=2;
int *ip;
ip=&x;
printf ("%d  %d  %d\n", x, y, ip);
y=*ip;
printf ("%d  %d  %d\n", x, y, ip);
x=ip;
printf ("%d  %d  %d\n", x, y, ip);
*ip=3;
printf ("%d  %d  %d\n", x, y, ip);
getch();
return 0;
}
```

Pokazivačka aritmetika

- pošto je pokazivač broj, tj. adresa promenljive, nad njim se mogu vršiti aritmetičke operacije: ++, -- , +, -

primer: inkrementacija (povećavanje za jedan)

- int *pok //deklarisanje pokazivača
- ako je vrednost pokazivača **pok** jednaka 1000 (u decimalnom brojnom sistemu) i ako su brojevi tipa **int** na računaru predstavljeni sa 4 bajta (32 bita), pozivom aritmetičke operacije **++** dobija se sledeća memorijska lokacija, odnosno preskače se dužina **int**-a
- naredba **pok++** će promeniti vrednost promenljive **pok** na

  1000+4 = 1004
- ako bi **pok** pokazivao na karakter, koji je veličine jednog bajta, operacija **pok++** bi promenila vrednost **pok**-a na **1001**

Inkrementiranje pokazivača

```
#include <stdio.h>
const int MAX = 3;

int main () {
   int  niz[] = {10, 100, 200};
   int  i, *pok;

   pok = niz;
   for (i=0;i<MAX;i++) {

      printf ("Adresa elementa niz[%d] = %x\n", i, pok);
      printf ("Vrednost elementa niz[%d] = %d\n", i, *pok);
      pok++; //pokazivač pokazuje na sledeći element niza
   }
   return 0;
}
```

vrednost promenljive kojom se označava niz je upravo adresa nultog elementa niza; nakon izvršavanja linije **pok = &niz[0]; pok** i **niz** imaju jednake vrednosti, tako da prethodna linija može biti napisana i kao **p = a;**

pokazivacu **pok** se dodeljuje adresa nultog elementa niza

Dekrementiranje pokazivača

```
#include <stdio.h>
const int MAX = 3;

int main () {
   int  niz[] = {10, 100, 200};
   int  i, *pok;

   pok = &niz[MAX-1]; //pokazivač pokazuje na poslednji element niza

   for (i=MAX;i>0;i--)
{
      printf ("Adresa elementa niz[%d]=%x\n", i-1,pok);
      printf ("Vrednost elementa niz[%d]=%d\n", i-1, *pok);
      pok--; //pokazivač pokazuje na prethodni element niza
   }

   return 0;
}
```

Poređenje pokazivača

```
#include <stdio.h>
const int MAX = 3;

int main () {
   int  niz[] = {10, 100, 200};
   int  i,*pok;

   pok=niz;
   i=0;
   while (pok<=&niz[MAX-1] )
  {
      printf ("Adresa elementa niz[%d] = %x\n", i, pok);
      printf ("Vrednost elementa niz[%d] = %d\n", i, *pok );
      pok++;
      i++;
   }
   return 0;
}
```

Korišćenje pokazivača

```
int x = 5, y = 1, z[20];
int *p;  //p je pokazivac na int
p = &x;  //p sada pokazuje na x
y = *p;  //y sada ima vrednost 5
*p = 0;  //x sada ima vrednost 0
p = &z[0]  //p sada pokazuje na z[0]
```

- unarni operatori **&** i ***** imaju veći prioritet od aritmetičkih operatora, tako da **y = *p + 5;** //prvo uzima vrednost na koju pokazuje **p**, uvećava je za **5** i dodeljuje promenljivoj **y**

```
*p += 1;   //uvećava *p za jedan
++*p;      //uvećava *p za jedan
(*p)++; //uvećava *p za jedan
```

- pošto su pokazivači promenljive kao i sve druge, moguće je njihovo korišćenje i bez dereferenciranja; jedan od primera je i dodeljivanje vrednosti jednog pokazivača drugom: **q = p;** čime se vrši kopiranje sadržaja pokazivača **p** u pokazivač **q**, tako da pokazivač **q** pokazuje na istu memorijsku lokaciju na koju pokazuje i **p**

Korišćenje pokazivača

- vrednost pokazivaču se može dodeliti i u samoj deklaraciji sa
  int *ip = &var
- ova naredba dodeljuje vrednost promenljivoj **ip**, i radi istu stvar kao kad napišemo
  int *ip; ip = &var;

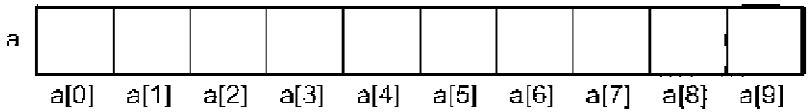
- ako ne postoji adresa koja bi se dodelila pokazivaču, dobro je dodeliti mu vrednost **NULL** – memorijska lokacija sa oznakom 0; ona označava da pokazivač ne pokazuje ni na šta

```
int main () {
   int  *ip = NULL;
   printf ("Vrednost pokazivaca je: %x\n", ip  );
   return 0;
}
```

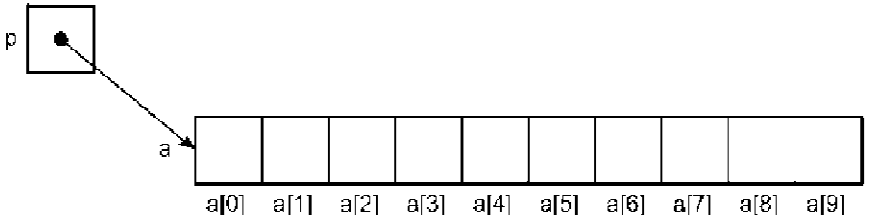
- provera da li pokazivač ima vrednost **NULL**
  if (pok) //tačno ako pok nije NULL
  if (!pok) //tačno ako pok jeste NULL

Nizovi i pokazivači

- pokazivači i nizovi su povezani; svaka operacija koja se može obaviti korišćenjem nizova, može se obaviti i pomoću pokazivača;
- deklaracija **int a[10];** definiše niz **a** veličine **10**, odnosno niz od 10 uzastopnih celih brojeva **a[0], a[1], ..., a[9**]

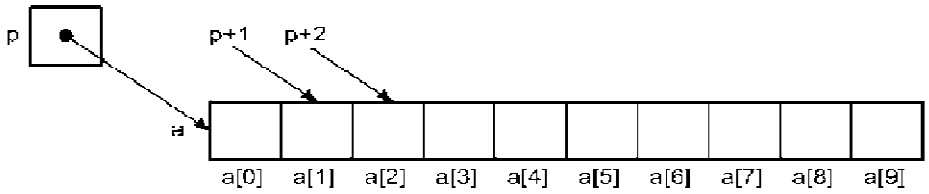


- ako je **p** pokazivač na ceo broj, deklarisan kao **int *p;** onda **p = &a[0];** postavlja pokazivač **p** da pokazuje na nulti element niza **a** ili drugačije rečeno pokazivač **p** sadrži adresu elementa **a[0]**



Nizovi i pokazivači

- pokazivač **p** pokazuje na prvi element niza **p=&a[0]**
- vrednost promenljive kojom se označava niz (**a**) je zaparavo adresa nultog elementa niza
- sledi da promenljive **p** i **a** imaju iste vrednosti, tj **p=a**
- ako **p** pokazuje na prvi element niza, onda **p+1** pokazuje na sledeći element, **p+i** pokazuje na **i**-ti element iza **p**
- ako **p** pokazuje na **a[0]**, onda ***(p+1)** predstavlja sadržaj elementa **a[1]**, a ***(p+i)** sadržaj elementa **a[i]**



- ovakav način pristupanja elementima niza može se koristiti bez obzira na tip elemenata niza
- sadržaj elementa **a[i]** se može napisati kao ***(a+i)**
- slično važi i za pokazivač **p**, pa je **p[i]** isto što i ***(p+i)**

Primer

```
//ispisivanje elemenata niza na tri načina
int main()
{
int niz[5]={3,4,5,6,7};
int i;
int *pok;

for (i=0;i<5;i++) //prvi način
     printf ("%d\n",niz[i]);

pok=niz;
for (i=0;i<5;i++) //drugi način
     printf ("%d\n",*(niz+i));

for (pok=niz;pok<niz+5;++pok) //treći način
     printf ("%d\n",*pok);
  getch ();
  return 0;
}
```

Primer

```
double bilans[5] = {1000.0, 2.0, 3.4, 17.0, 50.0};
double *p;
int i;
p = bilans;
printf ("Ispis vrednosti niza preko pokazivaca p\n");
for (i = 0; i < 5; i++)
    {printf ("*(p + %d) : %.1f\n", i, *(p + i) );}
printf ("Ispis vrednosti elemenata niza preko promenljive bilans\n");
for ( i = 0; i < 5; i++ )
     {printf ("*(bilans + %d) : %.1f\n", i, *(bilans + i) );}
```

- kreirana je promenljiva **bilans** koja je tipa **niz**, zatim je pokazivaču **p** dodeljena vrednost promenljive **bilans**, odnosno sada imamo dve promenljive, to su **bilans** i **p** koje pokazuju na prvi element niza
- u dve **for** petlje ilustrovan je prikaz elemenata niza preko ove dve promenljive; bitno je primetiti da je pokazivač **p** deklarisan kao **double**, što odgovora tipu elemanata niza

Šta ispisuje sledeći program

```
//pokazivači i stringovi
#include <stdio.h>
#include <conio.h>
int main()
{
char film[]="Tango argentino";
char *pok;
pok=film;
puts (pok);
puts (++pok);
film[5]='\0';
puts (film);
puts (++pok);
getch ();
return 0;
}
```

Pokazivač - parametar funkcije

- **C** funkcijama šalje argumente preko vrednosti, ne postoji direktan način da funkcija koja je pozvana promeni promenljive u funkciji koja ju je pozvala
- funkcija kopira vrednosti koje su joj poslate u svoje privremene lokalne promenljive; iako funkcija vrši promenu podataka u tim promenljivim, ta promena ne utiče na prosleđene promenljive
- funkcije mogu imati pokazivače kao ulazne parametre ili povratne vrednosti

int funkcija (int *param1, char *param2);

- funkcija ima dva ulazna parametra to su **param1** koji je pokazivač na **int** i **param2** koji je pokazivač na **char**

Pokazivač - parametar funkcije

primer:

```
int zameni (int *pa, int *pb);
int main()
        {int a = 4, b = 7;
        printf ("Pre zamene: a=%d, b=%d\n", a, b);
        zameni (&a, &b); //prosleđujemo adrese promenljivih
        printf ("Posle zamene: a=%d, b=%d", a, b);
return 0;}
int zameni (int *broj1, int *broj2)
        { int pom;
        pom = *broj1;
        *broj1 = *broj2;
        *broj2 = pom;
return 0;}
```

Prosleđivanje po referenci

- funkcija prima dve adrese **int** promenljivih, zato joj se kao parametri u pozivu prosleđuju adrese promenljivih **a** i **b**, odnosno **&a** i **&b**
- kada se u funkciji promene vrednosti promenljivih čije su adrese prosleđene kao parametri, menjaju se direktno vrednosti na memorijskim lokacijama i ove vrednosti ostaju sačuvane i posle izvršavanja funkcije
- ovaj princip se u **C**-u zove **prosleđivanje po referenci** jer se ne prosleđuju promenljive već adrese tih promenljivih odnosno referenca na njih; prosleđivanje po referenci se može implementarati jedino korišćenjem pokazivača i ne predstavlja pravi pojam prosleđivanja po referenci već njegovu simulaciju pomoću pokazivača (neki programski jezici imaju pojam prosleđivanje po referenci, u C-u se ovaj koncept simulira korišćenjem pokazivača)
- da li se funkcija zamene vrednosti brojeva može implementirati bez pokazivača? odgovor je ne, osim ako se koriste neke složenije strukture podataka koje sadrže dva elementa koje će se vratiti kao rezultat izvršavanja funkcije

Prosleđivanje po referenci

primer: funkcija računa količnik i ostatak dva broja; predaju se vrednosti dva broja (**x** i **y**) i adrese dve promenljive na kojima će se smestiti rezultati

```
void divAndMod (int x, int y, int* div, int* mod) //pri pozivu adrese idu u
                                                    pokazivače
{printf ("Kolicnik postavljam na adresu : %p\n", div); //div je vrednost
                                                        pokazivača *div
 printf ("Ostatak postavljam na adresu : %p\n", mod); //mod je vrednost
     *div = x / y;                                     pokazivača *mod
     *mod = x % y; }
int main()
     {int div, mod;
     printf ("Adresa promenljive div je %p\n", &div);
     printf ("Adresa promenljive mod je %p\n", &mod);
     divAndMod (5, 2, &div, &mod);
     printf ("Vrednost promenljive div je %d\n", div);
     printf ("Vrednost promenljive mod je %d\n", mod); return 0;}
```

Pokazivač, niz i funkcija

kada se niz prosleđuje funkciji ono što se, u stvari, prosleđuje je lokacija u memoriji za njegov početni član

f (a,n) ≡ f (&a[0],n)

ovo je razlog zbog koga se funkcija deklariše sa:

int f (int a[], int n)

ekvivalentna deklaracija je:

int f (int *a, int n) s obzirom da je **int a[] ≡ int *a**

Primer

//program koji ciklično pomera niz celih brojeva dužine **n** za **m** mesta ulevo
//funkcija za unos niza, funkcija za ispis niza i funkcija za pomeranje jednog mesta ulevo

```
void citajNiz (int *x, int n) //može se pisati (int x[], int n)
{
int i;
for (i=0;i<n;i++)
            {
            printf ("x[%d]=",i);
             scanf ("%d",x+i);
            }
}
void pisiNiz (int *x, int n)
{int i;
for (i=0;i<n;i++)
            {printf ("x[%d]=%d",i,*(x+i));
            printf ("\n");}
}
void LevoZa1(int *x, int n)
{int pom,i;
pom=*x;
for (i=1;i<n;*(x+i-1)=*(x+i),i++);
       x[n-1]=pom;}
```

Primer - nastavak

```
int main()
{
int n,m,i,x[50];
printf ("Unesite broj elemenata niza:\n");
scanf ("%d",&n);
printf ("Unesite elemente niza:\n");
citajNiz (x,n); //funkcija za unos elemenata niza
printf ("Za koliko mesta pomerate ulevo elemente niza:\n");
scanf ("%d",&m);
for (i=0;i<m;i++)
LevoZa1 (x,n); //funkcija za premeštanje niza za jedno mesto ulevo
printf ("Novi niz je:\n");
pisiNiz (x,n); //funkcija za ispis niza
return 0;
}
```

Povratna vrednost funkcije

- pokazivač može da bude i povratna vrednost iz funkcije, što se definiše na sledeći način:

int * funkcija();

- ovako deklarisana funkcija vraća pokazivač na **int**
- kod implementacije ove funkcije, može nastati problem ako vraćamo pokazivač na neku lokalnu promenljivu koju smo napravili u funkciji
- postoje i pokazivači na funkcije, a jedan primer deklaracije je

void (*funkcija) (int)

- ovde je deklarisan pokazivač na funkciju **funkcija** koja nema povratnih vrednosti i ima jedan ulazni parametar tipa **int**