Pokazivači u C-u

by Inazuma

Uvod

Većini kolega na PiPI-u glavobolju zadaju pokazivači (engl. *pointer*).

Pointeri, zapravo, nisu nikakav bauk i, ako se dobro savladaju, služe kao savršena podloga za daljnje učenje i savladavanje gradiva PiPI-a i ASP-a (na ASP-u tehnički sve ide preko pointera). Da ne govorim o tome kako su jako moćan alat.

Da ne duljim više s uvodom, nadam se da će vam ovaj kratak tutorial pomoći u razumjevanju i savladavanju pointera.

Sretno!

P. S.
Ne volim hrvatski naziv „pokazivači" pa ću u daljnjem tekstu stalno pisati „pointer".
I ako kopirate kod u neki IDE, obavezno pretipkajte navodnike jer nije isti encoding!

Za adrese su uzete neke random vrijednosti (nisu zbilja tako zauzete).

Varijable i adrese

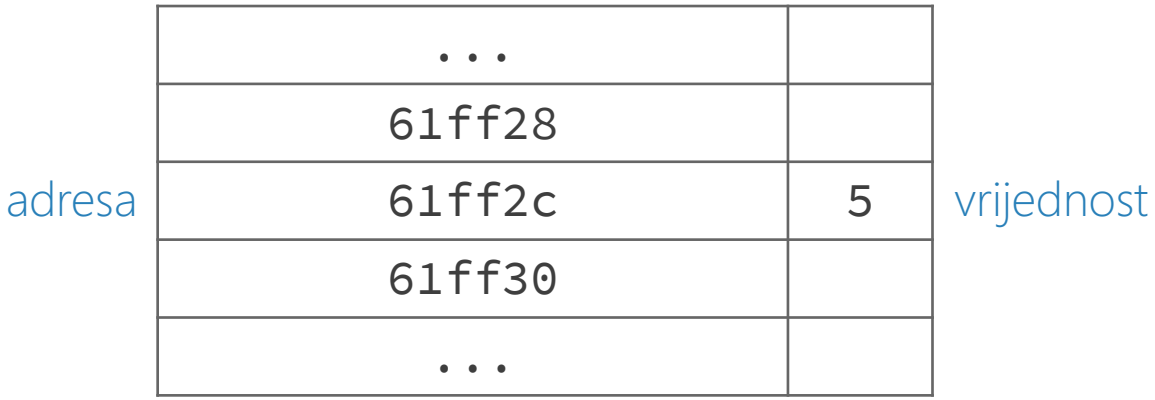
Svaka definirana varijabla je spremljena na nekoj memorijskoj lokaciji. Svaka ta memorijska lokacija ima svoju adresu. Adresi neke varijable možemo pristupiti operatorom `&`.
Pogledajmo primjer:

```
int main() {
    int a = 5;
    printf(„Adresa varijable a je %x.”, &a);
}
```

Ako pokrenemo gore navedeni kod, dobit ćemo ispis sličan ovome:

```
Adresa varijable a je 61ff2c.
```

Oznaka `%x` znači da argument treba ispisati u heksadekadskom sustavu, no to ovdje nije toliko bitno.
Kao ispis smo dobili adresu varijable `a`. U memoriji to izgleda ovako:



Što su zapravo pointeri?

Pointer je vrsta varijable koja sadrži adresu neke druge varijable, tj. direktnu adresu memorijske lokacije gdje je ta druga varijabla spremljena.

Pointer može biti bilo kojeg tipa, ovisno o varijabli na koju pokazuje.

Definira se tako da se ispred naziva varijable stavi `*` (zvjezdica).

Pogledajmo neke primjere pointera:

```
int main() {
    int *ip;        // Pointer na integer.
    double *dp;     // Pointer na double.
    float *fp;      // Pointer na float.
    short *sp;      // Pointer na short.
    char *cp;       // Pointer na char.
    long *lp;       // Pointer na long.
    unsigned *up;   // Pointer na unsigned (unsigned je isto što i unsigned int).
}
```

Općenita definicija: `tip *ime-varijable;`

Pridruživanje vrijednosti pointerima

Pridruživanje vrijednosti pointerima se odvija u 2 koraka:

1. Definira se pointer sa `tip *ime-varijable;` ovisno o varijabli kojoj se pristupa.

2. Pridruži mu se adresa neke varijable sa operatorom `&` pri čemu tip varijabli mora biti isti (ne može se adresa od `int` pridružit pointeru za `double`)!

Nakon toga, vrijednost pointera, tj. **vrijednost na adresi na koju pointer pokazuje (tj. vrijednost na adresi koja je spremljena u pointer, tj. vrijednost na adresi koja je vrijednost pointera, sve su sinonimi)** dobivamo operatorom `*` (zvjezdica). Dobivanje vrijednosti pointera zove se **dereferenciranje** pointera.

Primjer – pridruživanje adrese pointerima

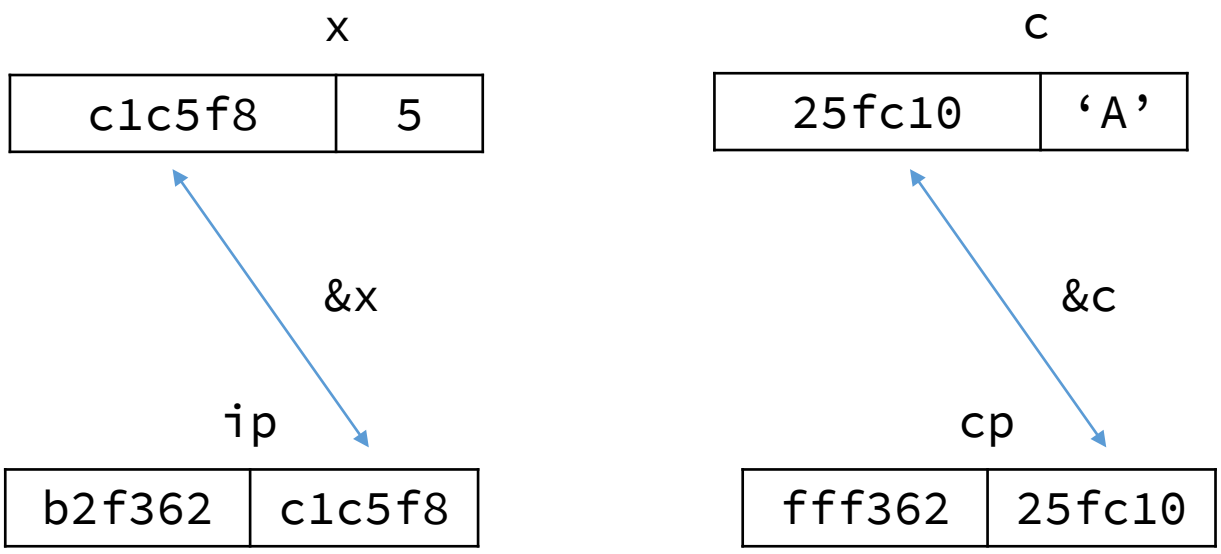
```
int main() {
    // Definiranje pointera.
    int *ip;
    char *cp;

    // Definiranje varijabli.
    int x = 5;
    char c = ‘A’;

    // Pridruzivanje adresa pointerima.
    ip = &x;
    cp = &c;

    // Ispis
    printf(„Vrijednost varijable x = %d.\n”, x);
    printf(„Vrijednost varijable c = %c.\n”, c);
    printf(„Vrijednost varijable x preko pointera = %d.\n”, *ip);
    printf(„Vrijednost varijable c preko pointera = %c.”, *cp);
}
```

Ako pokrenemo gore navedeni kod, ispis će biti:

```
Vrijednost varijable x = 5.
Vrijednost varijable c = A.
Vrijednost varijable x preko pointera = 5.
Vrijednost varijable c preko pointera = A.
```

Primjer – pridruživanje adrese pointerima

Grafički (u memoriji), to izgleda ovako (lijevo su adrese, desno vrijednosti varijabli):



Dakle, kad bi ispisali `ip`, ispisali bi vrijednost pointera i dobili bi `c1c5f8`, a kad bi ispisali `*ip`, dobili bi vrijednost na koju pokazuje pointer, tj. `5`.
Isto vrijedi i za `cp`.

Operacije s pointerima

Nad pointerima možemo raditi operacije isto kao nad običnim varijablama.

Recimo da imamo dvije varijable `int x = 5;` i `int y = 10;` i recimo da imamo dva pointera koji pokazuju na te varijable, `px = &x` i `py = &y`.

Sada vrijednostima varijabli `x` i `y` možemo pristupati preko pointera i također, raditi operacije nad njima kao što bi i nad samim varijablama. Na primjer:

```
int main() {
    int x = 5, y = 10, z, *px, *py;
    px = &x; py = &y;

    // z = x + y = 15
    // z = *px + *py = 15
    // z = x + *py = 15
    // z = 2 * (*px) = 10
    // z = *py / *px = 2
    // z = *px - *py = -5

    // Vrijednost neke varijable možemo promjeniti preko pointera.
    // *px = 7; - sada je x = 7
    // *px = *py - sada je x = 10.
    // (*px)++; - sada je x = 6 (ako je originalno bio 5). Zagrade su obavezne!
    // Preporučam da svugdje pišete zagrade oko pointera, npr. z = (*px) + (*py);
    // kako bi znali da su i zvjezdica i naziv zapravo varijabla.
}
```

Operacije s pointerima

No što bi se desilo ako bi napisali `px = px + 1;`?

Ovdje ne povećavamo vrijednost na koju pokazuje pointer (`*px`) nego vrijednost adrese koja je spremljena u pointer.

Međutim, on **neće** adresu povećati za fizički broj 1, npr. sa `ffff12` na `ffff13`, nego će ju povećati **za veličinu bajtova koju zauzima taj tip pointera**. Veličinu u bajtovima koju zauzima neki tip podatka možemo dobiti sa naredbom `sizeof(tip)`. Važno je napomenuti da veličine nisu jednake na svim sustavima! Ispod je tablica uobičajenih veličina tipova podataka (u bajtovima) u C-u:

tip	sizeof(tip)
char	1
short	2
int	4
float	4
double	8
long	4

Operacije s pointerima

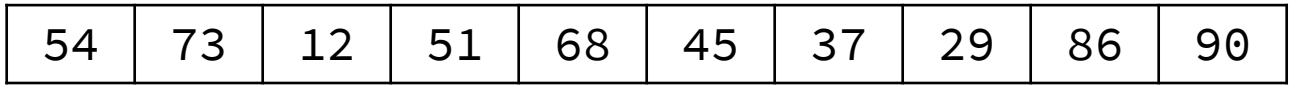
Dakle, ako je npr. zadano da je `int *px = &x; px = px + 1;`, onda bi se vrijednost adrese koja je spremljena u pointer povećala za 4 bajta, npr. sa `ffff10` na `ffff14`.

NO TO U OVOM SLUČAJU NIJE DOZVOLJENO I MOŽEMO SE SAMO DOVESTI U PROBLEME! Zašto? Zato što mi ne znamo što se nalazi na toj adresi koju smo dobili povećavanjem pointera i mjenjanjem vrijednosti na koju pokazuje ona pokazuje može dovesti do rušenja programa!

Pa zašto onda to postoji i kada smijemo to koristiti? Kod svima dragih polja.

Polja i pointeri

Dosad ste naučili da polje nekog tipa varijable možete inicijalizirati sa npr. `int polje[10]`. Time ste rezervirali memoriju za 10 elemenata tipa integer. Svaki taj integer je veličine 4 bajta. U memoriji je to ovako:



prvi element (indeks 0)

zadnji element (indeks 9)

Ako želimo pristupiti npr. prvom elementu polja, jednostavno bi napisali `int x = polje[0];`. Ove uglate zagrade su ustavri samo drugi (lakši) način zapisa pointera. Cijelom ovom polju i njegovim elementima se zapravo pristupa preko pointera.

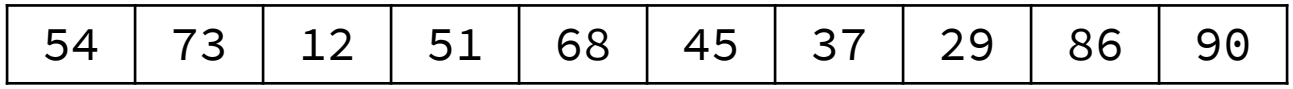
```
int main() {
    int polje[10], x;

    // x = polje[0]; -> x = 54. Istu ovu naredbu možemo zapisati kao:
    // x = *polje; -> x = 54. Skroz je isto.
    // x = polje[3]; -> x = 51. Ovako preko pointer zapisa:
    // x = *(polje + 3) -> x = 51.
}
```

Polja i pointeri

Pa kako to da sad smijemo napisati `x = *(polje + 3)`? Smijemo zato jer kad smo rezervirali polje, memorija je zauzeta „jedna za drugom" (to nije totalno točno, ali recimo da je).

Dakle, znamo da će od `*polje` do `*(polje + 9)` biti naše polje pa mu smijemo pristupati na način pointera (onaj zapis sa uglatom zagradom to ustvari i radi ispod haube).

Dakle `x` će postati ona vrijednost koja se nalazi na adresi na koju pokazuje `*polje` uvećano za 3, tj. za 3 * 4 = 12 bajtova (jer je int), a to je 51. `*polje` je zapravo nulti element polja.



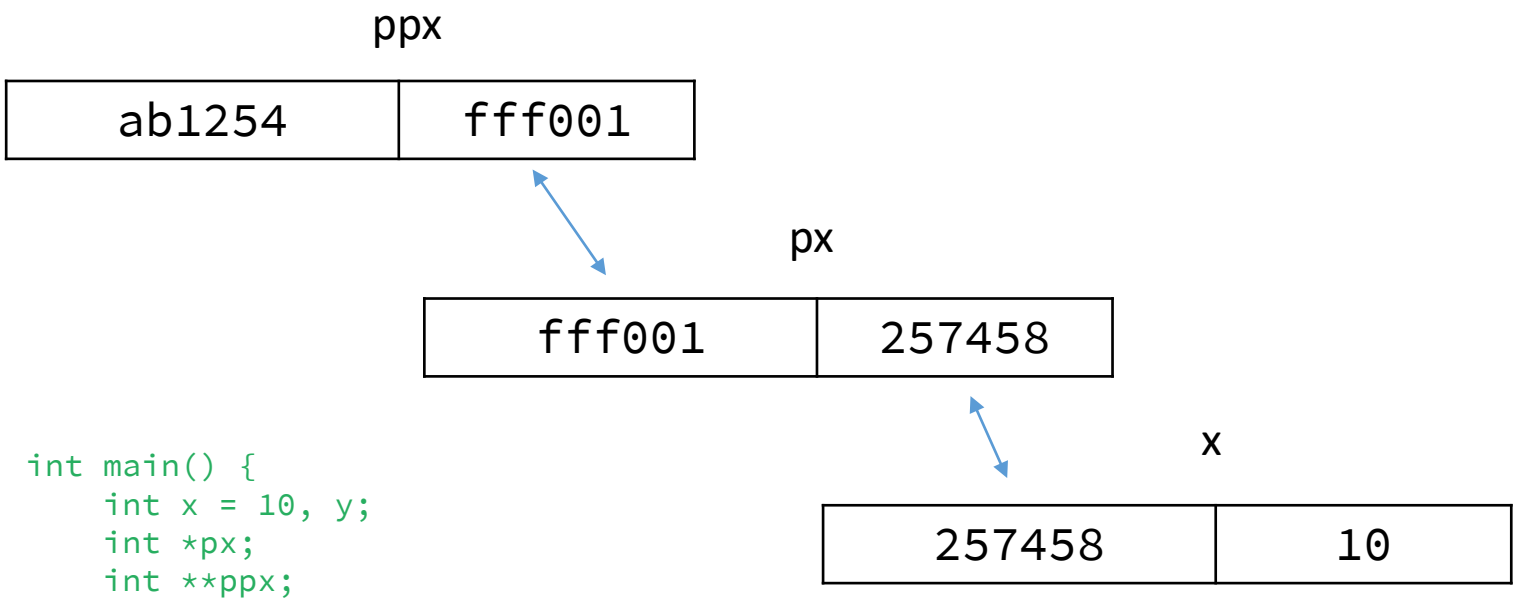
prvi element (indeks 0)

zadnji element (indeks 9)

Pointer na pointer...na pointer?

Da, moguće je. Pointer može pokazivati na pointer koji pokazuje na neki treći itd.

To nije ništa drugo nego pointer čija je vrijednost adresa nekog drugog pointera koji pokazuje na neku varijablu (lijevo je adresa, desno vrijednost).




```
int main() {
    int x = 10, y;
    int *px;
    int **ppx;

    px = &x;
    ppx = &px;

    // y = *px; -> y = 10.
    // y = **ppx; -> y = 10.
}
```

Zadnji savjet i zašto pointeri?

Crtajte. Crtajte, crtajte i samo crtajte. Bit će vam lakše se snaći i skužiti što, gdje i kako na što pokazuje.

A zašto si komplicirat život sa pointerima? Funkcije u C-u ne mogu direktno mjenjati vrijednosti varijabli, nego im se mora pristupati preko pointera, tzv. call-by-reference. Postoji i call-by-value, no obadva se rade na predavanjima i nisu toliko opširni tako da samo ukratko: call-by-reference – argumenti funkcije su pointeri, call-by-value – argumenti funkcije su obične varijable. Ako ih treba mjenjati u funkciji, koristi se call-by-reference i zato pointeri.

To je (ukratko) to.

Sretno na ispitima!