Kazalci

Kazalec na spremenljivko je poseben tip spremenljivk. Vrednost tega tipa nam pove, kje v pomnilniku je zapisana vrednost neke druge spremenljivke (pokaže na tisto spremenljivko). Kazalcu na spremenljivko drugače rečemo tudi *naslov spremenljivke*.

Če je `x` celoštevilska spremenljivka, potem kazalec `p`, ki kaže na spremenljivko `x`, definiramo kot je prikazano na spodnjem primeru. Če je `p` kazalec, pa lahko pogledamo, kakšna je vrednost spremenljivke, na katero ta kazalec kaže:

```
int x = 5;
int *p = &x;
int y = *p;
```

Znak `&`, ki ga zapišemo pred imenom spremeljivke, je operator, ki vrne naslov spremenljivke. Ta operator poznamo že iz funkcije `scanf`, kjer smo ga pisali pred imeni številskih spremenljivk. Naslov spremenljivke `x` shranimo v spremenljivko `p`, ki je tipa `int *` (kazalec na celo število). Eniški operator `*` (zvezdica), ki ga napišemo pred kazalcem, nam vrne vrednost spremenljivke, na katero kaže ta kazalec. Ker je `p` kazalec na celo število, nam operator vrne celoštevilsko vrednost. Ta vrednost je enaka 5, saj je `p` kazalec na spremenljivko `x`, ki ima vrednost 5.

Kazalci kot parametri funkcij

V poglavju o funkcijah smo videli, da sprememba parametra v funkciji navzven ni vidna. Če torej iz funkcije A pokličemo funkcijo B, in v funkciji B spremenimo vrednost katerega od njenih parametrov, tega v funkciji A ne opazimo. Zaradi tega do sedaj nismo znali napisati funkcije, ki bi znala zamenjati vrednosti dveh spremenljivk. Z uporabo kazalcev pa je možno tudi to, samo namesto spremenljivk, ki jima želimo zamenjati vrednosti, moramo za parametra dobiti naslova teh dveh spremenljivk (kazalca na spremenljivki).

```
void zamenjaj(int *a, int *b)
{
   int t = *a;
   *a = *b;
   *b = t;
}

int main(void)
{
   int x = 1;
   int y = 2;
   zamenjaj(&x, &y);
   ...
```

Funkcija `zamenjaj` za parametra dobi kazalca na spremenljivki, ki jima želimo zamenjati vrednosti. Ti dve spremenljivki definiramo v neki drugi funkciji (na primer v funkciji `main`), ko pa pokličemo funkcijo `zamenjaj`, ji za parametra podamo njuna naslova. Znotraj funkcije `zamenjaj` definiramo celoštevilsko spremenljivko `t`, ki dobi vrednost, na katero kaže kazalec `a` (v našem primeru `a` kaže na spremenljivko `x` iz funkcije `main`, torej `t` dobi vrednost 1). Nato spremenljivki, na katero kaže `a`, dodelimo vrednost, na katero kaže `b` (v našem primeru spremenljivki `x` dodelimo vrednost spremenljivke `y`, torej `x` dobi vrednost 2). Na koncu pa spremenljivki, na katero kaže `b`, dodelimo vrednost spremenljivke `t` (v našem primeru spremenljivka `y` dobi vrednost 1).

Na tak način lahko pišemo tudi funkcije, ki bi morale vračati več rezultatov. Kot smo videli v poglavju o funkcijah, lahko funkcija vrne samo eno vrednost (to vrednost vrne s stavkom `return`). Ostale rezultate pa lahko vrne preko kazalčnih parametrov. Ko želimo takšno funkcijo poklicati, si najprej deklariramo spremenljivke, v katere bo funkcija zapisala rezultate, potem pa poleg ostalih parametrov podamo funkciji še kazalce na te spremenljivke. Tako lahko iz funkcije dostopamo do spremenljivk, deklariranih v drugi funkciji, in v njih zapišemo naračunane rezultate. Primer takšne funkcije je `scanf`, ki preko kazalčnih parametrov v spremenljivke shrani prebrane vrednosti.

```
int minmax(int tab[], int n, int *min, int *max)
{
   if (n <= 0) return 0;
   *min = *max = tab[0];
   for (int i = 0; i < n; ++i) {
      if (tab[i] < *min) *min = tab[i];
      if (tab[i] > *max) *max = tab[i];
   }
   return 1;
}

int main(void)
{
   int t[] = {8, 5, 2, 7, 9, 4, 0, 4};
   int min, max;
   int ok = minmax(t, SIZEOF(t), &min, &max);
   ...
```

Funkcija `minmax` v zgornjem primeru dobi za parametra tabelo `tab` in njeno dolžino `n`, ter izračuna najmanjši in največji element tabele. Oba rezultata vrne preko parametrov `min` in `max`, ki sta kazalca na celoštevilski spremenljivki, deklarirani v funkciji `main`. Poleg tega funkcija `minmax` vrne še uspešnost. To je vrednost 0 ali 1, ki pove, ali je najmanjši in največji element sploh možno določiti.

Kazalci in tabele

Kazalci in tabele so si na nek način zelo podobni, tako da je včasih kar malo težko ločiti med njimi. Tabelo lahko deklariramo kot kazalec, torej je vseeno, ali pišemo `int t[]` ali `int *t`. Tabela je predstavljena s kazalcem na njen prvi element, kar pomeni, da je kazalec enak naslovu prvega elementa. Kazalcu lahko prištejemo celo število `i` in tako dobimo kazalec na `i`-ti element. V spodnji tabeli so po vrsticah zbrani enakovredni zapisi:

`int t[] = {1, 2, 3};`	`int *t = {1, 2, 3};`
`&t[0]`	`t`
`&t[i]`	`t + i`
`t[0]`	`*t`
`t[i]`	`*(t + i)`

Uporaba kazalcev nam omogoča, da tudi podtabelo vidimo kot čisto navadno tabelo. To največkrat uporabimo pri klicih funkcij, ki tabelo dobijo za parameter.

```
int t[] = {8, 5, 3, 7, 9, 4, 0, 4};
izpis(t, SIZEOF(t));                    // izpis cele tabele
izpis(t + 2, 5);                        // izpis podtabele {3, 7, 9, 4, 0}
```

Pogosto kazalec uporabimo tudi namesto indeksa, ko sestavljamo zanko po nizu. Za primer poglejmo zanko, ki izpiše vse znake niza `s`, ki niso presledki.

```
for (int i = 0; s[i] != '\0'; ++i) {
```

```
   if (s[i] != ' ') printf("%c", s[i]);
}
```

Takšno zanko bi z uporabo kazalcev napisali takole:

```
for (char *t = s; *t != '\0'; ++t) {
   if (*t != ' ') printf("%c", *t);
}
```

Kazalcem pa se ne moremo izogniti, kadar želimo sestaviti funkcijo, ki bi morala vrniti tabelo. Nove tabele v taki obliki, kot smo jo poznali do sedaj, funkcija ne more vrniti. Lahko pa v funkciji ustvarimo novo tabelo in vrnemo kazalec na njen prvi element. Ker k tabeli spada tudi podatek o njeni dolžini, moramo velikokrat vrniti tudi to (razen, če ni dolžina že določena z vhodnimi podatki). Edina možnost je preko dodatnega parametra. Kot primer poglejmo funkcijo, ki poskrbi za vnos tabele:

```
int *vnosTabele(char *dolzina, char *element, int *n)
{
   printf(dolzina);
   scanf("%d", n);
   int *tab = (int *)malloc(*n * sizeof(*tab));
   for (int i = 0; i < *n; ++i) {
      printf(element, i + 1);
      scanf("%d", tab + i);
   }
   return tab;
}
```

V funkciji `vnosTabele` smo uporabili kazalce povsod, kjer je bilo možno. Funkcija za parametra dobi dva niza, ki določata, kakšno besedilo se izpiše uporabniku pred vnosom dolžine tabele in vnosom posameznega elementa. Oba niza uporabimo kot formatna niza pri funkcijah `printf`. Pri drugem v nizu pričakujemo tudi formatno določilo za izpis celega števila. Niza smo namesto kot tabeli deklarirali kot kazalca. Funkcija bo vrnila kazalec na prvi element nove tabele, preko parametra `n` pa še njeno dolžino.

Pri prvem klicu funkcije `scanf` pred imenom spremenljivke nismo napisali znaka `&` kot smo ga običajno vedno napisali pri številskih spremenljivkah. To je zato, ker `n` ni celoštevilska spremenljivka, pač pa kazalec na celo število. Pri drugem klicu funkcije `scanf` pa smo namesto `&tab[i]` napisali kazalčno verzijo `tab + i`.

Novo tabelo smo ustvarili s klicem funkcije `malloc`, ki je definirana v `stdlib.h`. Funkcija rezervira zahtevano količino pomnilnika in vrne kazalec na začetek rezerviranega bloka. Ta kazalec pretvorimo v pravi tip in shranimo v spremenljivko `tab`. Velikost pomnilnika moramo podati v zlogih (bytih). Dolžina tabele v zlogih je enaka številu njenih element pomnoženo z velikostjo enega elementa. Velikost elementa ugotovimo z operatorjem `sizeof`. Funkcija na koncu vrne kazalec na začetek rezerviranega bloka, ki ga bomo uporabili kot tabelo.

```
int main(void)
{
   int n;
   int *t = vnosTabele("Vnesi dolzino: ", "Vnesi %d. element: ", &n);

   // s tabelo t delamo kot z običajno tabelo

   free(t);
   return 0;
}
```

Pred klicem funkcije `vnosTabele` moramo deklarirati celoštevilsko spremenljivko, v katero nam bo funkcija shranila dolžino tabele. Naslov te spremenljivke podamo kot tretji parameter

pri klicu funkcije. Funkcija nam vrne kazalec na prvi element tabele, ki si ga shranimo v kazalčno spremenljivko. Tabelo potem uporabljamo kot čisto navadno tabelo, ko pa je ne potrebujemo več, moramo sprostiti pomnilnik, ki ga je tabela zasedala. To storimo s klicem funkcije `free`, ki je tudi definirana v `stdlib.h`.