Programski jezik C - kazalci

Tomaž Dobravec

Kazalečna aritmetika

- Kazalčni aritmetični operatorji: +, -, ++, --
- Pomen aritmetičnih operatorjev je vezan na tip:
  - `int *p; p++` ... povečaj naslov v `p` za 4
    `p=p+2;` ... povečaj naslov v `p` za 8
  - `char *p; p++` ... povečaj naslov v p za 1

```
int *p;
p=&x;        // p = 0x7ffee36c;
p++;         // p = 0x7ffee370
p=p+2;       // p = 0x7ffee378
```

Relacijski kazalčni operatorji

- Uporabljamo lahko tudi relacijske operatorje:

  `<` `<=` `>` `>=`

  - `if (p <= q)` ... ali je naslov p manjši od naslova q?

```
int *ptr = tabela;
while ( ptr <= &tab[MAX - 1] ) {
   ...
   ptr++;
}
```

Kazalec na kazalec

Videli smo:

Če želimo, da se vrednost parametra tipa int znotraj funkcije spremeni, moramo parameter prenesti kot *kazalec na int*

Podobno:

Če želimo, da se vrednost parametra tipa kazalec znotraj funkcije spremeni, moramo parameter prenesti kot *kazalec na kazalec*

Kazalec na kazalec

Primer:

```
// nedelujoca koda
rezerviraj(int *x) {
  x = (int *) malloc(100);
}

main() {
  int *a;
  rezerviraj(a);
}
```

```
// delujoca koda
rezerviraj(int **x) {
  *x = (int *) malloc(100);
}

main() {
  int *a;
  rezerviraj(&a)
}
```

Funkcije, ki vračajo niz

Primer:

```
char * preberiBesedo(void)  {
  char *niz = malloc(100);
  scanf("%s", niz);
  return niz;
}
```

- Pozor: pomnilnika, ki ga vrne taka funkcija, nihče ne sprosti! Zelo nevarna praksa!
- Namesto tega raje pišemo:

```
void preberiBesedo(char *beseda)  {

 scanf("%s", beseda);

}
```

Funkcija `strstr`

Funkcija `strstr()` vrne **kazalec** na iskani podniz.

```
char *  strstr(const char *haystack, const char *needle);
```

- Primer uporabe:

```
char *niz = "To je dolg niz";
while (niz!=NULL) {
  printf("%p - %s \n", niz, niz);
  niz++;
  niz = strstr(niz, " ");
}
```

Funkcija `strtok`

Funkcija `strtok()` razbije niz na podnize

- Primer uporabe:

```
char str[] = ”abc:def:ghi”;
char delims[] = ":";
char *result;
result = strtok(str, delims);
while (result != NULL) {
  printf("%s\n", result);
  result = strtok(NULL, delims);
}
```

Prototip funkcije

- Funkcijo lahko napovemo kot ***prototip*** (podamo samo podpis, brez telesa), recimo takole:

```
void uredi(int *t);
```

- Tako funkcijo lahko kličemo v kodi

```
main() {
  int tabela[] = {5,4,7,2};
  uredi(tabela);
}
```

- Funkcija mora biti definirana kasneje v kodi ali v drugi datoteki (ali knjižnici).

Prototip funkcije

- S typedef lahko definiramo not tip funkcije:

```
typedef void urediFunkcija(int *);
```

(definirali smo tip z imenom `urediFunkcija`; tip predstavlja funkcijo, ki prejme tabelo in nič ne vrne)

- Prototip funkcije tipa urediFunkcija definiramo tako:

```
urediFunkcija uredi;
```

- Lahko definiramo tudi kazalec na funkcijo, takole:

```
urediFunkcija * uredi;
```

Funkcija kot parameter

- Parameter funkcije je lahko tudi kazalec na naslov druge funkcije

- Primer:

```
uredi(n, tabela, primerjalnik);
```

- Pozor: **ime funkcije == kazalec** na njen naslov

Primer: kot parameter podam operacijo.

Funkcija `qsort`

V `stdlib.h` je definirana funkcija qsort za urejanje tabel:

```
qsort(void *base, size_t nel, size_t width,
        int (*compar)(const void *, const void *))
```

Primer:

```
int primerjaj(const void *a, const void *b) {
  return *(int *)a - *(int *)b;
}
...
qsort(tab, N, sizeof(int),&primerjaj);
```