Uvod v vaje in osnove programskega jezika C

Pravila predmeta

Vaje pri predmetu izvajava asistenta:

- Rok Češnovar (rok.cesnovar@fri.uni-lj.si),
- Ratko Pilipović (ratko.pilipovic@fri.uni-lj.si).

Vsa vprašanja v zvezi s snovjo predmeta zastavljajte na forumu e-učilnice, saj boste s tem pomagali tudi ostalim kolegom. Zdravniška opravičila urejajte preko študentskega referata.

Vaje se izvajajo s pomočjo predlog za izvedbo vaj. Asistenti bodo občasno objavili krajše videoposnetke kjer bo to potrebne. V terminih za vaje tako ne bomo ponovno razlagali snovi iz predavanj. Termini za vaje bodo namenjeni zgolj za vaša vprašanja. Pred vajo si obvezno preberite gradivo za vajo, ki bo objavljeno na učilnici, najkasneje v petek v tednu pred vajo. Predloge za vaje vsako leto malo osvežimo in pregledamo, se nam pa zagotovo v tekst ali primere prikrade kakšna napaka. Vse napake, ki jih opazite, prosimo sporočite na e-učilnico ali na napako opozorite asistenta na vajah. Hvala!

Želimo vam uspešen in zdrav semester in veliko pridobljenega znanja!

Oprema za vaje

Na vajah bomo uporabljali razvojno ploščo STM32F4 Discovery. Ploščo bomo podrobno spoznavali čez celoten semester. Na e-učilnici je na voljo dokumentacija razvojne plošče.

Za razvoj programov, razhroščevanje in programiranje razvojne plošče bomo uporabljali orodje STM32CubeIDE, ki se lahko uporablja za razvoj poljubnega mikrokrmilnika proizvajalca STMicroelectronics. Orodje je brezplačno in na voljo za vse operacijske sisteme, naložite si ga lahko na https://www.st.com.

Osvežitev programskega jezika C

Gradivo, ki ga berete, ni mišljeno kot celostna ponovitev programskega jezika C. Namenjeno je osvetlitvi določenih podrobnosti, ki jih bomo pogosto

uporabljali in tistih tematik na katerih v dosedanjem času študija ni bilo dosti poudarka. V primeru, da menite, da je vaše znanje C-ja šibko, vam predlagamo, da pred naslednjim tednom pregledate katerega izmed spletnih tečajev ali prelistate katero izmed množice knjig o programskem jeziku C.

Podatkovni tipi

Klasični primitivni podatkovni tipi v C-ju so, upajmo da, dobro znani vsakemu izmed vas. Za predstavitev celih števil uporabljamo **char**, **short int** (ali krajše **short**), **int**, **long int** (krajše **long**) ter **long long int** (krajše **long long**). Vsi omenjeni tipi imajo tudi nepredznačene oblike, ki jih dobite s predpono **unsigned**, na primer **unsigned int**. Tabela 1 prikazuje število bitov, ki jih običajno zasede posamezen podatkovni tip.

Tabela 1: Podatkovni tipi.

Podatkovni tip	Število bitov
char	8
short	16
int	32
long	32
long long	64

Nimamo pa zagotovila, da bodo ti podatkovni tipi točno takšnih dolžin. Za podatkovni tip **char** vemo zgolj, da bo dolžina vsaj 8 bitov, lahko pa je tudi več. Pogosto bomo želeli bolj natančno določiti oziroma bolj jasno pokazati, koliko bitov naj zasede naša spremenljivka. Takrat bomo raje uporabili podatkovne tipe **int8_t**, **int16_t**, **int32_t**, **uint8_t**, **uint16_t**, **uint32_t**, ki so definirani v `<stdint.h>`. Dolžina tipov v bitih je tu podana kar v imenu, predpona **u** pa označuje nepredznačene tipe. Za te podatkovne tipe imamo zagotovilo, da če jih je na nekem sistemu mogoče uporabiti, bodo vedno pričakovanih dolžin.

Bitne operacije

C omogoča sledeče operacije nad biti:

- bitni in `x & y`,

- bitni ali `x | y`,
- bitni xor `x ^ y`,
- bitna negacija $\sim x$,
- pomik bitov v desno `x >> 2`,
- pomik bitov v levo `x << 3`.

Pri pomiku bitov v levo, se izpraznjeni bit vedno postavi na vrednost 0. Pri pomiku v desno se na izpraznjeno mesto pri nepredznačenih številih vpiše ničla, pri predznačenih pa se na izpraznjeno mesto vpiše vrednost zadnjega bita pred pomikom (predznaka).

Bitne operacije so pri delu z mikrokrmilniki nepogrešljive. Z njimi bomo izvajali predvsem sledeč operacije:

Postavi bit (Set Bits)

Pogosto bomo želeli bit b_i v n-bitni spremenljivki spremeniti v enico, pri tem pa ostale bite ohraniti na istem stanju. Na ta način bomo na primer prižgali LED diodo, zagnali motorček, itd. To najlažje dosežemo z operacijo ali (`a = a | x;`), kjer je x konstanta v kateri je zgolj bit b_i nastavljen na 1. Takšno število najlažje ustvarimo tako, da enico pomaknemo za i mest v levo. Ukaz, ki v spremenljivki a nastavi bit b_i je tako:

```
1    a = a | (1 << i);
```

Pobriši bit (Reset Bits)

Seveda bomo želeli na enak način bite tudi pobrisati oz. ponastaviti, torej vrednost bita b_i v n-bitnem številu spremeniti v ničlo, ostale bite pa ohraniti v istem stanju. Najenostavneje to dosežemo z operacijo **in** (`a = a & x;`), kjer je x konstanta v kateri je zgolj bit b_i nastavljen na 0, ostali biti pa na 1. Takšno konstanto najlažje ustvarimo tako, da enico pomaknemo za i mest, kot v prejšnjem primeru, nato pa dobljeno konstanto negiramo. Ukaz, ki spremenljivki a pobriše bit b_i je tako:

```
1    a = a & ~(1 << i);
```

Negiraj bit (Toggle Bits)

Z operacijo negacije obrnemo vrednost posameznega bita pri čemer stanje ostalih bitov ohranimo. To dosežemo na enak način kot pri postavljanju bita, le da namesto operacije ali uporabimo operacijo ekskluzivni ali (**xor**):

```
1       a = a ^ (1 << i);
```

Testiraj bit (Test Bits)

Velikokrat nas bo zanimala vrednost bita b_i v n-bitni spremenljivki. Na ta način bomo na primer ugotavljali ali je gumb pritisnjen ali spuščen. Najenostavnejši postopek za testiranje vrednosti bita b_i je:

```
// testiraj bit i
a & (1 << i);

// primer testiranja bita j
if (a & (1 << j)) {
    // ce je bit j enica
} else {
    // ce je bit j nicla
}
```

Kazalci

Kazalci v jeziku C so pogosto trn v peti vsakega študenta. Pri predmetu Organizacija računalniških sistemov s kazalci ne bomo ustvarjali kompleksnih podatkovnih struktur, ampak bomo uporabljali njihov primarni namen – kazanje na specifični naslov.

Kazalec je namreč zgolj spremenljivka, katere vrednost predstavlja naslov. Tip kazalca pa nam pove kakšnega tipa je spremenljivka katere vrednost je v kazalcu.

Kazalec določenega tipa definiramo tako, da poleg tipa dodamo zvezdico. **int*** predstavlja kazalec na spremenljivko tipa **int**. Naslov torej *kaže* na 4 bajte, katerih vrednost predstavlja predznačeno celo število. Če imamo spremenljivko a definirano kot `int a = 4;`, kazalec na spremenljivko dobimo z ukazom `int *p = &a;`. Znak & pred imenom spremenljivke namreč vrača njen naslov. Spodaj je podana koda običajnih ukazov, ki jih izvajamo s kazalci. Razlaga je podana v komentarjih:

```
// definiramo kazalec
int *p;
// definiramo spremenljivko
int a = 26;

p = &a; // kazalec kaze na a, recimo da se nahaja na naslovu
0x200
```

```
7       *p = 12; // vrednost na naslovu 0x200 nastavimo na 12, kar
        pomeni da je vrednost a 12
8
9       *(p+1) = 7; // vrednost na naslovu 0x204 nastavi na 7
10      // v zgornjem primeru s povecavo stevca za 1 pridemo do
        naslova 0x204 (0x200 + 0x4), do stirice pridemo, ker je na
        naslovu 0x200 spremenljivka tipa int, ki zaseda 4 bajte
11
12      // cesa ne smemo poceti s kazalci
13      // vse od spodaj nastetega je slaba praksa ali pa neveljaven
         ukaz
14      p = a; // TEGA NE POCNITE! S tem bi naslov kazalca
        spremenili na 12. Kaj je na naslovu 12 obicajno ne vemo!
15      *a = 6; // TUDI TEGA NE POCNITE! S tem ukazom bi vrednost na
         naslov 12 nastavili na 6.
16
17      &a = 4; // neveljaven ukaz
18      &p = 6; // neveljaven ukaz
```

Kazalce bomo pri vajah predmeta ORS uporabljali predvsem takrat, ko bomo želeli pisati na točno določene naslove. Drug primer pa je, ko bomo želeli da funkcija spreminja vrednost argumenta. Primer takšne uporabe je prikazan spodaj. Funkciji `fun_value` podamo vrednost spremenljivke x, ki jo potem v funkciji spremenimo. Ker je bila spremenljivka podana kot vrednost, se ta sprememba ne pozna v funkciji, ki kliče `fun_value`. V primeru, da si želimo takšnega obnašanja, moramo parameter x podati kot kazalec, v funkciji pa spreminjati vrednost na naslovu na katerega kaže ta kazalec. Tak primer je prikazan v funkciji `fun_reference`.

```
1       void fun_value(int a) {
2           // ...
3           a = 4;
4           // ...
5       }
6
7       void fun_reference(int* a) {
8           // ...
9           *a = 7;
10          // ...
11      }
12
13      int main() {
14          int x = 1;
```

```
15          fun_value(x);
16          // x je tu se vedno 1
17          fun_reference(&x);
18          // x je sedaj 7 tudi v main funkciji
19      }
```

Strukture

Struktura v jeziku C je zbirka spremenljivk različnih osnovnih tipov.

```
1       struct circle {
2           int x;
3           int y;
4           uint16_t z;
5           float r;
6       };
7
8       struct circle a;
9       a.x = 4;
10      a.y = 6;
11
12      struct circle *b;
13      b->x = 3;
14      b->y = 7;
```

Posebnost struktur, ki jo bomo s pridom izkoriščali tekom semestra, je ta, da se elementi strukture v pomnilniku vedno nahajajo zaporedno. Če se zgornja struktura začne na naslov 200, potem vemo, da se element x nahaja na naslovu 0x200, element y na naslovu 0x204, element z na naslovu 0x208, element r pa na naslovu 0x20A.

Druga posebnost struktur je njihova uporaba v kombinaciji s kazalci. Kazalec na strukturo definiramo enako kot vsak drug kazalec, kar lahko vidite v zgornjem primeru. Naivno bi tako lahko pričakovali, da bomo do posameznih elementov dostopali z `*b.x = 3;`. Težava nastopi, ker ima pika najvišjo prioriteto med operatorji zato bi to morali zapisati kot `(*b).x`, kar pa je precej neberljivo. Namesto tega lahko uporabimo bolj pregledno obliko s puščico: `b->x`.

Naloge

1. Predpostavimo, da program v C-ju prevajamo za procesor HIP, ki ste ga spoznali pri predmetu Arhitektura računalniških sistemov. V kateri ukaz se prevede pomik v desno, če je spremenljivka tipa *uint8_t*? Kaj pa, če je spremenljivka tipa *int8_t*?

2. Kakšna je razlika med bitno operacijo in (npr. 7&3) ter logično operacijo in (npr. 7&&3). Sta rezultata enaka?

3. Zakaj je zaporedje ukazov `int i = 5; &i = 3;` neveljavno? Kaj bi dejansko storili, če bi prevajalnik to zaporedje ukaza dovolil?

4. Implementirajte naslednje funkcije:

   - `reset_bit(uint32_t* x, uint8_t p)` – resetiraj p-ti bit števila na katerega kaže kazalec x.
   - `reset_two_bits(uint32_t* x, uint8_t p)` – resetiraj bita p in p+1 števila na katerega kaže kazalec x.
   - `set_bit(uint32_t* x, uint8_t p)` – p-ti bit števila na katerega kaže kazalec x nastavi na 1.
   - `set_two_bits_to(uint32_t* x, uint8_t p, uint8_t n)` – bita p in p+1 števila na katerega kaže x nastavi na vrednost n (00, 01, 10, 11)
   - `vector_length(struct vector a*)`, ki prejme kazalec na strukturo `struct vector` s tremi elementi: `x`, `y`, `length`. Prva dva elementa sta predznačeni celi števili, `length` pa število v plavajoči vejici. Funkcija `vector_length` na podlagi vrednosti `x` in `y` izračuna dolžino vektorja ($length = \sqrt{x^2 + y^2}$) in jo zapiše v `length`. Funkcija nič ne vrača.

Primeri in namigi

Primeri za testiranje vaših rešitev 4. naloge:

```
1   uint32_t a;
2   a = 0xF;     reset_bit(&a, 2) -> 0xB
3   a = 0xA;     reset_bit(&a, 0) -> 0xA
4   a = 0xFF;    reset_two_bits(&a, 3) -> 0xE7
5   a = 0xB7;    reset_two_bits(&a, 3) -> 0xA7
6   a = 0xB;     set_bit(&a, 0) -> 0xB
7   a = 0xE;     set_bit(&a, 2) -> 0xE
8   a = 0xEF;    set_two_bits_to(&a, 3, 1) -> 0xEF
9   a = 0xB7;    set_two_bits_to(&a, 3, 2) -> 0xB7
10  struct vector b;
11  b.x = 4;
12  b.y = -2;
13  vector_length(&b); b.length = 4.47
```

Števila v šestnajstiški obliki izpišete z ukazom `printf(``%X``,x);` Za reševanje nalog uporabite enega izmed spletnih simulatorjev C-ja, na primer https://www.onlinegdb.com/online_c_compiler.