Zapiski iz pouka Osnove programiranja II
Programiranje Mikrokontrolerjev

Matej Blagšič

8. marec 2018

Kazalo

1 Osnovno

Pri temu predmetu bomo obravnavali jezik C. Za uporabo lahko preneseš okolje Codeblocks z MinGW inštalacijo ali posebej MinGW compiler in poljubno okolje(Jetbrains).

Pomembno je, da imaš predznanje iz prejšnjega polletja pri Javascriptu, saj so tipi spremenljivk, sintaksa in drugo zelo podobno, tako da v detajle o stvarih, ki so enake ne bom šel.

Vsak dokument začnemo z `#include <stdio.h>` za standardne vhodne in izhodne ukaze. Vsaka koda se izvaja znotraj main funkcije:

```
int main(){
    printf("Hello!\n");
    return 0;
}
```

Prav tako je pomembno uporabiti PODPIČJE za vsakim ukazom/vrstico!!!

Če začnemo na začetku, opazimo `#include` ukaz. Ta se izvrši, preden se karkoli drugega. V temu primeru lahko vnesemo knjižnice. Te nam olajšajo programiranje s tem, da nam en ukaz izvede več ukazov, ki bi jih morali tipkati na roke. To datoteko/knjižnico navedemo lahko z "datoteka"navednicam. Če pa damo v <datoteka>, potem pa išče datoteke v sistemskih mapah okolja. Te datoteke so vrste **header** s končnico **.h**. V našem primeru je knjižnica za pisat in brat podatke - vhodne in izhodne podatke.
To je podobno kot v javascriptu: `<script src="datoteka">`

2 Podatkovni tipi

Si poglejmo zgled:

```
int main(){
    int a;
    float b; //spremenljivka

    printf("Vprisi prvo vrednost");
    scanf("%d", &a);
    printf("Vpisi drugo vrednos");
    scanf("%f", &b);
    printf("%d + %f = %f\n", a, b, a+b);
    return 0;
}
```

C je občutljiv na tip podatkov. Pravimo tudi, da je C statično tipiziran jezik. To pomeni, da moramo vrsto podatka navesti. To pomeni, da se moramo sami odločiti, kakšen tip podatka bo nosila spremenljivka.
Vemo, da v Javascriptu nismo rabili napisati tipa spremenljivke, le `var`, torej je Javascript dinamično tipiziran jezik.

Tipi spremenljivk:

TIP	DOLŽINA(bitov)	FORMATNO DOLOČILO	OBMOČJE
char	8	%d %c	-128 do 127
short, int	16	%d	-65536 do $+65535$
	32		-32768 do $+23767$
long	vsaj 32	%ld	-2.1×10^9 do $+2.1 \times 10^9$
float	običajno 32	%f	-2.1×10^9 do $+2.1 \times 10^9$
double	običajno 64	%lf	-9.2×10^{18} do $+9.2 \times 10^{18}$
void	0		

V C-ju Boolov tip ne obstaja, tako da primerjalni operatorji delujejo enako, le da vračajo 0 za false in 1 za vse, kar je različno od nič. **Ne obstaja TRUE ali FALSE**.

Spoznali bomo tudi, da je pri celoštevilskem tipu pomembna omejitev območja, pri realnem tipu pa natačnost!

Velikokrat bomo srečali izraz `unsigned`. ta nam območje podatkovnega tipa prestavi od 0 do 2x maksimum. Če je char od -128 dp 127, potem je unsigned char od 0 do 255;

2.1 Celoštevilski tip (n bitov)

Obstaja nepredznačen, ki je od 0 do 2^{32}

2.2 Realni tip (IEEE floating point)

p	eksp. (e)	mantisa (m)
1 bit	8 bitov	23 bitov

Ta ima enojno natančnost (single precision) ali `float` in so števila zapisana z 32 bitno velikostjo. Tako so v desetiškem sistemu števila natančna do 7,22 signifikantnih mest, sepravi 7 mest je natančnih, od 8. števila naprej pa je že vprašljivo. Torej, signifikantno pomeni pomembno, tisto, kar je natančno.

Če hočemo večjo natančnost, uporabimo `double` oz. dvojna natačnost (double precision). Ima kapaciteto 64 bitov, torej v desetiškem do 15,95 mest natančno. Po 15. mestu je že vprašljivo natančno.

2.3 Modifikatorji formatna določila

`%d` vemo, da stoji za cela števila. Če vrinemo neko število "N-> `"%Nd"`, potem povemo, na koliko mest se izpiše število, deluje na desno poravnavo.
Če vrinemo ničlo -> `"%0Nd"`, potem zapolne prazna mesta z ničlam.
Če vrinemo "N.Mf-> `"%N.Mf"`, potem izpiše N mest število z M mesti za decimalno piko.

```
int x = 15;
float y = 3.141592;

printf("%5d",x); --> Izpise _ _ _1 5
printf("%.2f",y); --> Izpise 3.14
printf("%05d",x); --> Izpise 00015
```

2.4 Znaki

Imamo več standardov znakov. Najbolj osnoven in razširjen je `ASCII` (American standard code for information Interchange). Ta zapis je 8 - biten. Lahko najdemo tabelco, ki nam pokaže kodo za vsak znak.

0	NUL	16	DLE	32	SPC	48	0	64	@	80	P	96	`	112	p
1	SOH	17	DC1	33	!	49	1	65	A	81	Q	97	a	113	q
2	STX	18	DC2	34	"	50	2	66	B	82	R	98	b	114	r
3	ETX	19	DC3	35	#	51	3	67	C	83	S	99	c	115	s
4	EOT	20	DC4	36	$	52	4	68	D	84	T	100	d	116	t
5	ENQ	21	NAK	37	%	53	5	69	E	85	U	101	e	117	u
6	ACK	22	SYN	38	&	54	6	70	F	86	V	102	f	118	v
7	BEL	23	ETB	39	'	55	7	71	G	87	W	103	g	119	w
8	BS	24	CAN	40	(	56	8	72	H	88	X	104	h	120	x
9	HT	25	EM	41	)	57	9	73	I	89	Y	105	i	121	y
10	LF	26	SUB	42	*	58	:	74	J	90	Z	106	j	122	z
11	VT	27	ESC	43	+	59	;	75	K	91	[	107	k	123	{
12	FF	28	FS	44	,	60	<	76	L	92	\	108	l	124	|
13	CR	29	GS	45	-	61	=	77	M	93	]	109	m	125	}
14	SO	30	RS	46	.	62	>	78	N	94	^	110	n	126	~
15	SI	31	US	47	/	63	?	79	O	95	_	111	o	127	DEL

Slika 1: ASCII tabela

V C-ju je pomembno, da damo en znak v enojne navednice. S tem pomeni, da program zaznava ASCII kodo. Torej, če izpišemo `printf("%d", '0');`, nam program izpiše 48. Pri znakih uporabimo torej spremenljivko `char`.

2.5 Psevdonaključna števila

Ena metoda za pridobivanje naključnih števil je metoda srednjih kvadratov. Pri tem kvadriramo dvomestna števila. Pri tem je statistično gledano naključnost zelo podobna realni naključnosti, kot da bi metali kocke.

3 Branje podatkov

Da nam program prebere podatek, uporabimo funkcijo:

```
scanf("formatno_dolocilo", &spremenljivka);
```

Vidimo, da moramo najprej deklarirati tip podatka, ki ga pričakuje operator Scanf. Potem pa določimo naslovni operator & in nato za njim spremenljivko, ki naj sprejme podatek.

4 Pisanje podatkov

Za pisanje podatkov uporabimo funkcijo:

```
printf("formatni_niz", izrazi)
```

Pomembne so tudi ubežne sekvence. To so `\r \n \t`, ki povejo, kaj se zgodi, ko se text izpiše. `\n` naredi novo vrstico(new line) po besedilu, `\t` je tabulator...

Tako v našem primeru, se a izpiše tam, kjer je njegov %d in b, kjer je %f ter vsota a+b tam, kjer je %f(glej izsek programske kode).

5 Operatorji

Pri C-ju so enaki operatorji, kot v JS, le da z nekimi izjemami: Operator === in !=== ne obstajata.
Prav tako operator za deljenje ne zapišemo kot "/"ne deluje enako. Problem prihaja iz tipa spremenljivk. Če obsoječo spremenljivko x, ki je tipa int, deljimo ali spreminjamo tako, da bi postala ta spremenljivka kateregakoli drugega tipa, kot prvotni int, potem vrne program 0. Primer:

```
int maint(){
    x = 7;
    x = x / 8 * 8

    printf("%d", x);
    return 0;
}
```

Če pa spremenimo prvo 8 z 8.0, potem bo program jo vzel za realno število in deljil in nato nazaj množil z 8, tako se te pokrajšata in program vrne 7.

6 Funkcije

Funkcije deklariramo:

```
float imeFunkcije(){/*telo funkcije*/return 0;}
```

Opazimo, da funkcijo deklariramo kot float oz. funkcijo, ki vrne realno število. V resnici lahko funkcije definiramo kot karkoli hočemo, glede na to, kaj naj bi vrnila.

Prav tako vidimo, da glavna zanka, v kateri se koda izvaja, je `main`. v tej kodi se izvajajo vsi programi in funkcije. Tako se koda, ki je napisana tu notri, se prevede in spremeni v izvršilno kodo(executable).

Primer funkcije je iz poglavja Psevdonaključna števila. Tam smo spoznali definiranje funkcije: `int dogodek(float verjetnost);` Vidimo, da moramo za razliko od JS definirati vrsto spremenljivke, ki gre v vhodne podatke, tj. verjetnost. Poleg tega, ker se konča z podpičjem, imenujemo ta del prototip. Nič ne naredi. Nato definiramo šele funkcijo.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int dogodek(float verjetnost);
int dogodek(float verjetnost){
    if((float)rand() / RAND_MAX <= verjetnost){}
        return 1;
    }
}

int main(){

    int x, stevec = 0;
    srand(time(NULL));
    for(x=0; x < 100000;x++){
        stevec += dogodek(0.5);
    }
    printf("%d", stevec);

    return 0;
}
```

V funkciji je pomembno, da pretvorimo rand() v float tip spremenljivke, ker drugače gre za celoštevilsko deljenje, kar potem pomeni le 0 ali 1. Problem je, da nam potem vsakič vrne enako vrednost okoli 50 000, ker je ta random le psevdonaključna. Zato srednjo verednost definiramo z ukazom `srand`(oz seme) in vanj vnesemo čas, ki pa nikoli ni enak. Zato tako vsakič generira zares naključno število. Naredimo primer na bolezni.

Izračunajmo, koliko % ljudi, ki so bolani zares, zanje test pokaže, da so res bolani. Testiramo 100 000 ljudi in vemo, da bolezen ubije 0.5% ljudi. Prav tako vemo, da test pokaže z natančnostjo 99%, da je oseba bolana. 1%, da je oseba zdrava. Ampak ali je res, da je 1% zdravih ali napačno diagnosticirano.

```
int main(){

    unsigned long i;
    float pBolan = 0.005; //verjetnost da ubije
    float pPozitBolan = 0.99; //resnicno bolan verjetnost
    float pPozitZdrav = 0.01;//verjetnost da pokaze da je bolan, ceprav je zdrav
    unsigned long pozit = 0;
    unsigned long pozitBolan = 0;

    srand(time(NULL));
    for(i = 0; i<100000;i++){
        if(dogodek(pBolan)){//vemo da je bolan
            if(dogodek(pPozitBolan)){//testiramo kako dobro izmerimo,ce je bolan
                pozitBolan++;//dodamo ga med bolane in pozitivno testirane
                pozit++;
            }
        }
        else{//testiramo zdravega
            if(dogodek(pPozitZdrav)){//tu se znajde zdrav in pozitivno testiran
                pozit++;
            }
        }
    }
    printf("%f", (float)pozitBolan/pozit);// rezultat je bolni/testirane
        pozitivno
}
```