PROGRAMIRANJE EMBEDDED SISTEMA U PROGRAMSKOM JEZIKU C

Rekapitulacija programskog jezika C

C progremski jezik se najčešće koristi za programiranje embedded sistema. Neke od njegovih prednosti su:

* Nizak nivo pristupa hardveru (npr.direktno adresiranje registara).
* Brzina izvršavanja.
* Kontrola nad memorijom.
* Kompatibilnost sa većinom mikrokontrolera i toolchain-a.

Tipična struktura programa u C-u za mikrokotrolere možemo predstaviti na sledeći način:

#include “math.h“ *//uključivanje biblioteka*

*/\*Globalne varijable – sve funkcije mogu da im prostupaju\*/*

int count; *//globalna (statička) varijabla*

*/\*Definicije funkcija \*/*

int imeFunkcije1( float i){ *//parameter i se prosleđuje funkciji, a fukcija vraća podatak tipa int*

int i,j; *//lokalne promenljive, alociraju se na steku*

**--instrukcije za imlementaciju funkcije**

 }

*/\* Glavni program\*/*

void main(void){

unsigned int pr1; *// lokalna promenljiva*

int i1; *// lokalna promenljiva*

*/\*Deo za inicijalizaciju\*/*

**--Instrukcije za inicijalizaciju promenljivih, I/O portova, uređaja...**

*/\*Beskonačna petlja\*/*

while(1){

**--instrukcije koje se ponavljaju**

} */\*ponavlja se zauvek\*/*

}

C program za mikrokontroler ima malo drugačiju strukturu od standardnog C programa – fokusiran je na direktnu kontrolu hardvera i obično nema operativni sistem. Umesto toga, koristi se beskonačna petlja za neprekidno izvršavanje koda.

Tipovi podataka u C-u

U zavisnosti od platforme na kojoj se radi može da dođe do nekih odstupanja u dužini koju zauzima određeni tip podataka, ali uvek možemo na svakoj platformi da proverimo koliko bajtova zauzima određeni podatak korišćenjem naredbe *sizeOf(tip\_podaka)*.

U sledećoj tebeli je dato predstavljanje celobrojnog tipa podataka u C-u.

|  |  |  |
| --- | --- | --- |
| Deklaracija tipa podatka | Broj bitova | Opseg vrednosti |
| char x;unsigned char x;uint8\_t x;  | 8 | 0÷255 |
| short x;signed short x;int16\_t x; | 16 | -32768÷+32767 |
| unsigned short x;uint16\_t x; | 16 | 0÷65535 |
| int x;signed int x;int32\_t x; | 32 | -21474836648÷+21474836647 |
| unsigned int x;uint32\_t x; | 32 | 0÷4294967295 |

Logički tip podataka predstavljamo sa **bool** tipom podataka, širina je 1 bit, a može da uzima vrednosti **true** ili **false** (**1** ili **0**).

Za predstavljanje realnih podataka koriste se tipovi **float** (jednostruka preciznost, zauzima 4 bajta) i **double** (dvostruka preciznost, zauzima 8 bajtova).

Promenljive u programskom jeziku C se moraju deklarisati da bi se naznačio tip i dužina informacije koja će se čuvati na tim lokacijama. Simbiličko ime se daje da bi se referencirala informacija:

 int a, b, c; //deklarisane su tri promenljive tipa int

 char x,y; //deklarisane su dve promenljive tipa char

Memorijski prostor koji se alocira za promenljive može se nalaziti u RAM memoriji, ili ROM/Flash memoriji (u njoj se smeštaju konstante).

Promenljive mogu biti automatske i static. Automatske promenljive se deklarišu u okviru funkcije/procedure, promenljiva je vidljiva samo u okviru te funkcije. Memorijki prostor za automatsku promenljivu se alocira na vrhu steka kada se pozove funkcija, a nakon izvršenja funkcije zauzeti prostor se dealocira, odnosno oslobađa. Pošto se svaki put oslobađa memorija, to indicira da promenljiva između dva poziva funkcije ne čuva vrednost.

Primer automatske promenljive:

void delay(){

 int i,j; //automatske promenljive, vidljive samo u okviru funkcije delay

 for(i=0;i<10;i++){ //promenljive moraju da se inicijalizuju svaki put

 for(j=0;j<10000;j++){

}

}

}

Za razliku od automatskih, statičke (static) promenljive ne menjaju momorijski prostor tokom izvršavanja programa. Statičke promenljive mogu da se deklarišu bilo unutar ili izvan funkcije. Ukoliko se definišu izvan funkcije, za takve varijable kažemo da su globalne i mogu da se koriste na bilo kom mestu u programu. Ukoliko se definiše unutar funkcije, ispred tipa promenljive mora da se se napiše ključna reč *static*, pri čemu je opseg promenljive lokalni, ali između poziva funkcija promenljiva zadržava svoju vrednost.

Primer static promenljive:

unsigned count; //globalna promenljiva je statička, alocira se fiksna lokacija u RAM memoriji

 //count može da se koristi bilo gde u programu

void func1(){

 int i; //automatska promenljiva, alocira se memorija na vrhu steka

 static int j; //statička promenljiva, alocira se fiksna memorijska lokacija u RAM-u

 if (count==0) //test vrednost globalne promenljive count

 i=count; //statička promenljiva se inicijalizuje pri prvom ulasku u func1

 i=count; //inicijlizuje se automatksa varijabla svaki put kad se uđe u func1

 j=j+i; //menja se statička promenljiva j, vrednost se čuva do sledećeg poziva func1

} //dealocira se memorijski prostor koji je zauzimala automatska promenljiva i

void main(void){

 count=0; //inicijalizuje se globalna promenljiva count

 while(1){

func1();

count++; //inkrementira se globalna promenljiva count

}

}

U programskom jeziku C, aritmetičke operacije su osnovne matematičke operacije koje se koriste za manipulaciju brojevima. C jezik podržava standardne aritmetičke operacije kao što su sabiranje, oduzimanje, množenje, deljenje i ostatak pri deljenju.

Evo spiska aritmetičkih operatora u C:

**Aritmetički operatori u C:**

1. Sabiranje +
2. Oduzimanje -
3. Množenje \*
4. Deljenje /
5. Ostatak pri deljenju %

**Primer korišćenja aritmetičkih operacija u C:**

#include <stdio.h>

int main() {

 int a = 10, b = 3;

 // Sabiranje

 printf("Sabiranje: %d + %d = %d\n", a, b, a + b);

 // Oduzimanje

 printf("Oduzimanje: %d - %d = %d\n", a, b, a - b);

 // Množenje

 printf("Množenje: %d \* %d = %d\n", a, b, a \* b);

 // Deljenje

 printf("Deljenje: %d / %d = %d\n", a, b, a / b); // Rezultat je celobrojni, ostatak se odbacuje

 // Ostatak pri deljenju

 printf("Ostatak pri deljenju: %d %% %d = %d\n", a, b, a % b)

 return 0;

}

**Objašnjenje:**

* **a + b**: Sabira vrednosti a i b.
* **a - b**: Oduzima vrednost b od a.
* **a \* b**: Množi vrednosti a i b.
* **a / b**: Deli vrednost a sa b. Rezultat je celobrojni, pa će ostatak biti zanemaren (ako su a i b celobrojne promenljive).
* **a % b**: Ostatak pri deljenju vrednosti a sa b. Na primer, 10 % 3 daje rezultat 1, jer 10 podeljeno sa 3 daje količnik 3, a ostatak je 1.

**Rezultat izvođenja ovog programa:**

makefile

Copy

Sabiranje: 10 + 3 = 13

Oduzimanje: 10 - 3 = 7

Množenje: 10 \* 3 = 30

Deljenje: 10 / 3 = 3

Ostatak pri deljenju: 10 % 3 = 1

**Napomena:**

* **Deljenje sa nulom**: Deljenje sa nulom (npr. a / 0) je nevaljano i može izazvati grešku u programu.
* **Celobrojno deljenje**: Kada delite dva celobrojna broja, rezultat je takođe celobrojni, a decimalni deo se odbacuje. Ako želite decimalni rezultat, morate koristiti tipove sa pomičnim zarezom (float ili double).

**Bit-paralelni logički operatori u C:**

1. **& (bitwise AND)**: Vraća 1 ako su oba bita na istoj poziciji 1, inače vraća 0.
2. **| (bitwise OR)**: Vraća 1 ako je bar jedan bit na istoj poziciji 1, inače vraća 0.
3. **^ (bitwise XOR)**: Vraća 1 ako su bita različita (jedan je 0, a drugi 1), inače vraća 0.
4. **~ (bitwise NOT)**: Inverzija svih bitova (pretvara 1 u 0, a 0 u 1).
5. **<< (bitwise left shift)**: Pomera bitove ulevo za određeni broj pozicija.
6. **>> (bitwise right shift)**: Pomera bitove udesno za određeni broj pozicija.

**Primer korišćenja bit-paralelnih operatora:**

#include <stdio.h>

int main() {

 int a = 5, b = 3; // a = 0101, b = 0011 u binarnom formatu

 // Bitwise AND (&)

 int and\_result = a & b; // 0101 & 0011 = 0001

 printf("a & b = %d\n", and\_result);

 // Bitwise OR (|)

 int or\_result = a | b; // 0101 | 0011 = 0111

 printf("a | b = %d\n", or\_result);

 // Bitwise XOR (^)

 int xor\_result = a ^ b; // 0101 ^ 0011 = 0110

 printf("a ^ b = %d\n", xor\_result);

 // Bitwise NOT (~)

 int not\_result = ~a; // ~0101 = 1010 (u 32-bitnom formatu biće -6)

 printf("~a = %d\n", not\_result);

 // Bitwise Left Shift (<<)

 int left\_shift\_result = a << 1; // 0101 << 1 = 1010

 printf("a << 1 = %d\n", left\_shift\_result);

 // Bitwise Right Shift (>>)

 int right\_shift\_result = a >> 1; // 0101 >> 1 = 0010

 printf("a >> 1 = %d\n", right\_shift\_result);

 return 0;

}

**Objašnjenje:**

1. **Bitwise AND (&)**:
	* a = 5 → 0101 (u binarnom formatu)
	* b = 3 → 0011 (u binarnom formatu)
	* a & b = 0001 (što je 1 u decimalnom formatu)
2. **Bitwise OR (|)**:
	* a = 5 → 0101
	* b = 3 → 0011
	* a | b = 0111 (što je 7 u decimalnom formatu)
3. **Bitwise XOR (^)**:
	* a = 5 → 0101
	* b = 3 → 0011
	* a ^ b = 0110 (što je 6 u decimalnom formatu)
4. **Bitwise NOT (~)**:
	* a = 5 → 0101
	* ~a = 1010 (u 32-bitnom formatu može biti -6, jer se koristi binarni komplement)
5. **Bitwise Left Shift (<<)**:
	* a = 5 → 0101
	* a << 1 pomera bitove ulevo za jedan korak: 1010 (što je 10 u decimalnom formatu)
6. **Bitwise Right Shift (>>)**:
	* a = 5 → 0101
	* a >> 1 pomera bitove udesno za jedan korak: 0010 (što je 2 u decimalnom formatu)

**Rezultat izvođenja programa:**

a & b = 1

a | b = 7

a ^ b = 6

~a = -6

a << 1 = 10

a >> 1 = 2

**Dodatne napomene:**

* **Bitwise NOT (~)** se koristi za inverziju svih bitova broja. Ako je broj pozitivan, rezultat može biti negativan zbog upotrebe **2's complement** za predstavljanje negativnih brojeva.
* **Shift operacije (<<, >>)** pomeraju bitove u levo ili desno. Pri pomeranju u levo, prazna mesta se popunjavaju sa 0, dok pri pomeranju udesno mogu postojati varijacije u ponašanju (zavisi od implementacije i da li je broj pozitivan ili negativan).

Bit-paralelni operatori se često koriste u situacijama kada radite sa binarnim podacima ili kada želite efikasno manipulisati bitovima za optimizaciju programa.

**Logički operatori u C jeziku:**

1. **&& (logičko AND)**: Vraća true (1) ako su oba uslova istinita.
2. **|| (logičko OR)**: Vraća true (1) ako je barem jedan od uslova istinit.
3. **! (logičko NOT)**: Vraća true (1) ako je uslov netačan (inverzuje vrednost).

**Primer korišćenja logičkih operatora u C:**

#include <stdio.h>

int main() {

 int a = 5, b = 10;

 // Logičko AND (&&)

 if (a > 0 && b > 0) {

 printf("Oba broja su pozitivna.\n");

 }

 // Logičko OR (||)

 if (a > 0 || b < 0) {

 printf("Bar jedan broj je pozitivan.\n");

 }

 // Logičko NOT (!)

 if (!(a < 0)) {

 printf("Broj 'a' nije negativan.\n");

 }

 return 0;

}

**Objašnjenje:**

1. **&& (AND)**:
Uslov a > 0 && b > 0 će biti true samo ako su oba broja pozitivna. U ovom slučaju, a i b su oba pozitivna, pa će se ispisati poruka "Oba broja su pozitivna."
2. **|| (OR)**:
Uslov a > 0 || b < 0 će biti true ako je bilo koji od uslova tačan. U ovom slučaju, a > 0 je tačno (jer je 5 pozitivno), pa će se ispisati poruka "Bar jedan broj je pozitivan."
3. **! (NOT)**:
Uslov !(a < 0) je suprotan od a < 0. Pošto je a pozitivno, a < 0 je false, a negacija !false je true, pa će se ispisati poruka "Broj 'a' nije negativan."

**Rezultat izvođenja programa:**

Oba broja su pozitivna.

Bar jedan broj je pozitivan.

Broj 'a' nije negativan.

**Specijalne napomene:**

* **Logičke vrednosti**: U C jeziku, logičke vrednosti su predstavljene celim brojevima. 0 znači false, a svaki broj različit od nule znači true.
* **Kratenje u evaluaciji**: U C jeziku, kada koristite logičke operatore, C primenjuje *kratko-cuts* (short-circuiting). To znači da:
	+ U && (logičko AND), ako je prvi uslov false, C neće ni proveravati drugi uslov jer rezultat sigurno neće biti true.
	+ U || (logičko OR), ako je prvi uslov true, C neće proveravati drugi uslov jer rezultat sigurno neće biti false.

Ove operacije su veoma korisne u svim vrstama logičkih uslova i upravljanja tokom izvršavanja programa.

**Operatori za pomeranje bitova u C-u**

U programskom jeziku C, **shift operatori** (<< i >>) omogućavaju pomeranje bitova u levo ili desno. Ovi operatori se koriste za efikasno manipuliranje sa bitovima brojeva, a posebno su korisni u bit-paralelnim operacijama, optimizaciji i radu sa binarnim podacima.

### Tipovi shift operatora:

1. **Levo pomeranje (<<)**: Pomera bitove broja ulevo.
2. **Desno pomeranje (>>)**: Pomera bitove broja udesno.

Pomeranje bitova ulevo znači da se svi bitovi broja pomeraju za određeni broj pozicija u levo. Na kraju, novi bitovi sa desne strane se popunjavaju sa 0. Svako pomeranje ulevo za jedan korak je ekvivalentno množenju broja sa 2.

**Sintaksa**: rezultat = broj << broj\_pozicija;

gde:

* broj je broj čije bitove pomeramo,
* broj\_pozicija je broj pozicija za koje želimo pomeriti bitove ulevo.

Pomeranje bitova udesno znači da se svi bitovi broja pomeraju za određeni broj pozicija udesno. Na kraju, novi bitovi sa leve strane se popunjavaju sa 0 (za pozitivne brojeve), ili sa 1 (za negativne brojeve, zavisi od implementacije). Svako pomeranje udesno za jedan korak je ekvivalentno deljenju broja sa 2, pri čemu se decimalni deo zanemaruje.

**Sintaksa**:

rezultat = broj >> broj\_pozicija;

Gde:

* broj je broj čije bitove pomeramo,
* broj\_pozicija je broj pozicija za koje želimo pomeriti bitove udesno.

### Primer korišćenja shift operatora u C:

#include <stdio.h>

int main() {

 int a = 5; // a = 0000 0101 (u binarnom formatu)

 // Left Shift (<<)

 int left\_shift = a << 1; // Pomeranje za 1 poziciju ulevo (a \* 2)

 printf("a << 1 = %d\n", left\_shift); // Rezultat: 10 (u decimalnom formatu)

 // Right Shift (>>)

 int right\_shift = a >> 1; // Pomeranje za 1 poziciju udesno (a / 2)

 printf("a >> 1 = %d\n", right\_shift); // Rezultat: 2 (u decimalnom formatu)

 // Pomeranje ulevo za 2 pozicije

 int left\_shift\_2 = a << 2; // Pomeranje za 2 pozicije ulevo (a \* 4)

 printf("a << 2 = %d\n", left\_shift\_2); // Rezultat: 20 (u decimalnom formatu)

 return 0;

}

### Objašnjenje:

1. **Left Shift (<<)**:
	* a = 5, što je 0000 0101 u binarnom formatu.
	* Pomeranje a << 1 daje 0000 1010, što je 10 u decimalnom formatu (pomereno ulevo za jedan bit, što je ekvivalentno množenju sa 2).
	* Pomeranje a << 2 daje 0001 0100, što je 20 u decimalnom formatu (pomereno ulevo za dva bita, što je ekvivalentno množenju sa 4).
2. **Right Shift (>>)**:
	* a = 5, što je 0000 0101 u binarnom formatu.
	* Pomeranje a >> 1 daje 0000 0010, što je 2 u decimalnom formatu (pomereno udesno za jedan bit, što je ekvivalentno deljenju sa 2).

### Rezultat izvršenja programa:

a << 1 = 10

a >> 1 = 2

a << 2 = 20

### Dodatne napomene:

* **Pozitivni brojevi**: Kada pomerate pozitivne brojeve, operatori će jednostavno pomeriti bitove u levo ili desno i popuniti preostale pozicije sa 0.
* **Negativni brojevi**: Kada pomerate negativne brojeve, ponašanje operatora >> može zavisiti od implementacije i specifičnog načina reprezentacije broja (npr. **2's complement**). Obično, **right shift za negativne brojeve** popunjava levo mesto sa 1, ali ovo zavisi od implementacije.
* **Prelivanje (Overflow)**: Ako pomerate bitove broja koji je previše veliki ili mali za promenu u novi broj tipa, može doći do **prelivanja** (overflow). Na primer, pomeranje broja sa velikim brojem bitova ulevo može uzrokovati da bitovi "ispadnu" van opsega.
* **Optimalizacija**: Bit shift operatori su često korišćeni za optimizaciju, jer su brži od aritmetičkih operatora za množenje i deljenje. Na primer, množenje broja sa 2 može biti efikasnije pomoću a << 1 nego korišćenjem a \* 2.