1 UVOD

MISRA C je skup smernica za razvoj programske podrške u C jeziku, za upotrebu u automobilskoj inustriji. Razvijen je od strane MISRA (Motor Industry Software Reliability Association) organizacije. Njeni ciljevi su unapređenje pouzdanosti, portabilnosti, i sigurnosti programskog koda, u kontekstu računarskih sistema, posebno onih koji su programirani po ISO C-C90-C99 standardu. MISRA C je široko prihvaćen standard i model za najbolje prakse u radu. Koristi se u razvoju programske podrške za automotiv, avio, telekomunikacione, medicinske industrije, kao i drugih.

Suština primene Misra pravila je pisanje softvera koji će biti što je više moguć pouzdan i robustan u 100 posto situacija, jer se taj softver koristi na uređajima (automobile, avioni, medicinski uređaji…) gde čak i mala greška u kodu može dovesti do situacije gde je ugrožen ljudski život.

Pomoću Misra standard vrši se statička analiza softvera, (eng. *static program analysis*), tj analiza softvera bez izvršavanja istog. Pored toga, za potpunu proveru softvera, potrebno je izvršiti dinamičku analizu softera, tj testiranje rada softvera koja se izvršava.

MISRA C standard trenutno je trećoj iteraciji. Prva iteracija standarda je MISRA-C:1998, i imala je 127 smernica, od čega su 93 obavezne, a preostale 34 saveti. Druga edicija je nazvana "Guidelines for the use of the C language in critical systems" ili MISRA-C:2004, I sadrži 142 pravila, od kojih su 122 obavezna. **MISRA-C:2012** proširuje podršku na C99 verziju C programskog jezika, **i ova upustva vam šaljem u obliku pdf-fajla**. Sadrži 143 pravila i 16 direktiva, koja su podeljena u tri grupe: obavezna, potrebna, i saveti. U aprilu 2016, MISRA je objavila *MISRA Compliance:2016*, koji obezeđuje poboljšane smernice za postizanje usaglašenosti sa standardom.

Svaka smernica pripada jednoj klasi smernica: Obavezno, Potrebno, i Saveti

Klasifikacija:

- Obavezne smernice koje uvek moraju biti zadovoljenje (ispoštovane).
- Potrebne smernice koje uvek moraju biti zadovoljene, osim ako nije predmet odstupanja.
- Saveti se smatraju dobrom praksom, ali ne moraju biti zadovoljeni.

Smernice se mogu logički podeliti u kategorije, kao sto je urađeno u *MISRA-C:2012*, neke od osnovnih pravila koje je potrebno pratiti :

- Izbegavanje mogućnosti razlika u kompajleru, npr veličina celobrojne promenjive u C može da varira u zavisnosti od korišćenog kompajlera, ali celobrojna promenjiva od 16 bita je uvek 16 bita (C99 standardizacija *int16_t*).

- Izbegavanje korišćenja funckija koje su sklone greškama, npr funkcija *malloc()* može neuspešno da se izvrši.

- Pisati održiv i lako čitljiv kod, u kome se lako pronalaze greške.

- Koristeći pravila i smernice koja su se dobro pokazali u praksi.

- Izbegavanje pravljenja previše kompleksnih rešenja. Mnoga MISRA C pravila se mogu okarakterizovati kao smernice zato što pod određenim uslovima inženjer može odstupiti od pravila, a da se smatra da je u saglasnosti sa standardom. Odstupanja moraju biti dokumentovana, ili u samom programskom kodu, ili u posebnom dokumentu. Pored dokaza da je inženjer razmotrio da bezbednost sistema nije ugrožena odstupanjem od pravila, zahtev za odstupanje od pravila mora da sadrži:

- Pravilo od kog se odstupa,

- Razlog odstupanja,

- Objašnjenje zašto to odstupanje neće narušiti bezbednost sistema

2 PRIMERI KORIŠĆENJA MISRA SMERNICA

Prikazano je pisanje jednog veoma jednostavnog koda, od samog početka tj. pisanja main funkcije, i uporedna provera koda pomoću PC-linta , softvera kojeg koristimo za proveru Misra smernica (**upustvo za instalaciju PC-linta na MPLAB-X je dato u fajlu Integracija PC-Linta sa MPLABX.pdf).** Na ovom primeru može se videti da se čak i kod jednostavnog programa stalno dešavaju situacije gde se prekršene Misra smernice, u određenim slučajevima pisanjem samo jedne linije koda može se desiti da dođe do prekršaja dve ili više Misra smernice, i da poštovanje ovih smernica može da značajno produži pisanje koda. Ipak, u praksi se pokazalo da programeri veoma brzo počnu da automatski poštuju većina Misra smernica, posle par situacija u kodu gde su prekršili te smernice.

U suštini velika većina Misra smernica opisuje situacije u programiranju koje se obično karakterišu kao *dobra programerska praksa* koje bi se trebale poštovati i što pre usvojiti prilikom programiranja jer značajno smanjuju broj bagova na duže staze i omogućavaju pisanje čitljivijeg koda, nezavisno od toga da li se piše program za automotive industriju ili za neku drugo svrhu.

2.1 DEKLARACIJA FUNKCIJE

Čak i kada se napiše najjednostavniji mogući program, kao što je prazna main funkcija, jedna Misra smernica nije ispoštovana . Kada se pokrene provera Misra smernica, tj *Lint this file* , dobijaju se komentari koji opisuju koja lint pravila nisu ispoštovana, i u kojoj liniji koda su se desila.

KOD:

```
void main(void)
{
}
```

LINT KOMENTARI

PC-lint for C/C++ (NT) Vers. 9.00L, Copyright Gimpel Software 1985-2014

***1.** C:\lint\lnt\configSL\co-xc32.lnt:42: Warning 686: Option '-wlib(0)' is suspicious because of 'the likelihood of causing meaningless output'; receiving a syntax error in a library file most likely means something is wrong with your Lint configuration*

--- Module: D:\GOOGLE_DRIVE\Materijal za vezbe\autoelektronika\lekcije_za_sovu\vezba5-misra\misra-primer\misra-primer.X\newmain.c (C)

***2.** D:\GOOGLE_DRIVE\Materijal za vezbe\autoelektronika\lekcije_za_sovu\vezba5-misra\misra-primer\misra-primer.X\newmain.c:5: Note 9075: external symbol 'main(void)' defined without a prior declaration [MISRA 2012 Rule 8.4, required]*

Objašnjenje LINT komentara:

1. Ovaj komentar je sistemska informacija o podešavanjima PCLinta, i uvek je prisutna, ali nije od značaja za našu analizu
2. Ovde je prekršeno pravilo 8.4, koje se u *Misra guidlines* nalazi u potpoglavlju 8.8, na strani 75 u pdf-u tj 67 strani *Misra guidlines* , i koje glasi

Rule 8.4	A compatible declaration shall be visible when an object or function with external linkage is defined

C90 [Undefined 24], C99 [Undefined 39]

Category	Required
Analysis	Decidable, Single Translation Unit
Applies to	C90, C99

Amplification

A compatible declaration is one which declares a compatible type for the object or function being defined.

Ovo pravilo propisuje da je neophodno napraviti deklaraciju objekata i funkcija, detalje pročitajte u *Misra guidlines*, i naročito proučite primere. Za naš slučaj govori nam da nemamo deklaraciju main funkcije (što praktično nikad nije ni neophodno, ali PC-lint ne razlikuje main funkciju od bilo bilo koje druge funkcije).

Ispravka koda:

```
void main(void); // deklaracija main funkcije

void main(void)
{

}
```

2.2 DEFINICIJA PROMENJIVE

Ako u kod ubacimo definiciju globalne promenjive temp, kao u kodu dolje, dobijaju se 2 prekršaja Misra smernica.

KOD:

```
void main(void); // deklaracija main funkcije
int temp;
void main(void)
{
}
```

LINT KOMENTARI

--- Module: D:\GOOGLE_DRIVE\Materijal za vezbe\autoelektronika\lekcije_za_sovu\vezba5-misra\misra-primer\misra-primer.X\newmain.c (C)

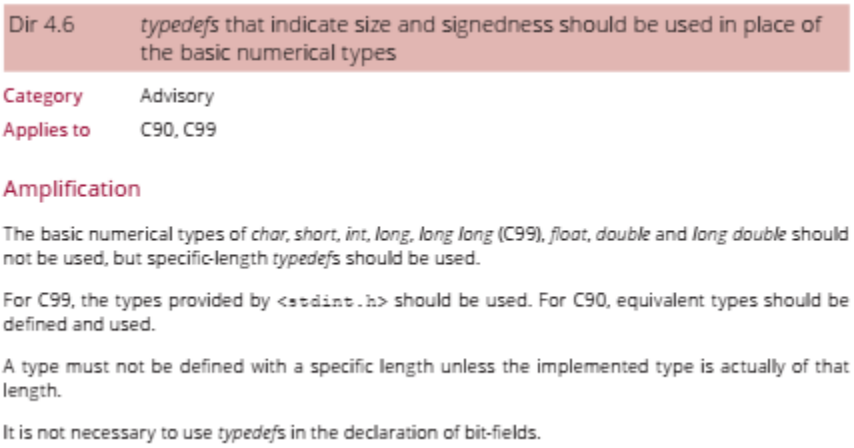
int x;

1. D:\GOOGLE_DRIVE\Materijal za vezbe\autoelektronika\lekcije_za_sovu\vezba5-misra\misra-primer\misra-primer.X\newmain.c:3: Note 970: Use of modifier or type 'int' outside of a typedef [MISRA 2012 Directive 4.6, advisory]

2. D:\GOOGLE_DRIVE\Materijal za vezbe\autoelektronika\lekcije_za_sovu\vezba5-misra\misra-primer\misra-primer.X\newmain.c:3: Note 9075: external symbol 'x' defined without a prior declaration [MISRA 2012 Rule 8.4, required]

Objašnjenje LINT komentara:

1. Ovde se radi o *Directivi 4.6 , koja se nalazi u poglavlju 7, potpoglavlje 7.4 .Kategorija ove smernice je advisory, znači nije obavezna da se izvrši, ali primenićemo je pošto je veoma korisna*



Dir 4.6	*typedefs* that indicate size and signedness should be used in place of the basic numerical types
Category	Advisory
Applies to	C90, C99

Amplification

The basic numerical types of *char*, *short*, *int*, *long*, *long long* (C99), *float*, *double* and *long double* should not be used, but specific-length *typedefs* should be used.

For C99, the types provided by `<stdint.h>` should be used. For C90, equivalent types should be defined and used.

A type must not be defined with a specific length unless the implemented type is actually of that length.

It is not necessary to use *typedefs* in the declaration of bit-fields.

Smernica je prilično jasna, potrebno je koristiti typedef za definiciju tipa promenjive umesto osnovnih tipova promenjivih tipa char, short,int i slično. Razlog za ovo je smanjenje problema prilikom portovanja koda, da bi se znala tačna količina memorije koju zauzima određen tip promenjive, jer npr. int promenjiva na 16 bitnim procesorima najčešće zauzima 16 bita, a na 32-bitnim najčešće zauzima 32 bita. Za detalje procitajte Rationale.

Ispravka koda:

```
void main(void); // deklaracija main funkcije
typedef signed int int16_s; // definisan tip promenjive int16_s
int16_s temp;
void main(void)
{
}
```

2. Ovde je opet prekršeno pravilo 8.4, ali sad prilikom definicije promenjive. Mora se prvo izvršiti deklaracija funkcije, što se čini ključnom reči *extern*, kao u sledećom kodu

```
void main(void); // deklaracija main funkcije
typedef signed int int16_s; // definisan tip promenjive int16_s
extern int16_s temp; // deklarisana globalna promenjiva temp
int16_s temp; // definisana globalna promenjiva temp
void main(void)
{
}
```

Extern se koristi ako je izvorni program u više fajlova tj modula, jer onda se globalna promenljiva definiše samo u jednom fajlu, a definiše sa extern u ostalim. Ključna reč extern ukazuje kompajleru

da su promenljive koje slede definisane negde drugde i da za njih ne treba zauzeti memorijski prostor. U trenutku kada kompajler naiđe na promenljivu koja nema unutrašnju deklaraciju on je traži među globalnim (spoljašnjim) promenljivama i ako je nađe obezbeđuje povezivanje.

2.3 MANJE ISPRAVKE

Obično se deklaracije ubacuju u header fajlove, tj. to je pravilo " dobrog" programiranja, što ćemo ovde ispoštovati. U projektu ubacite prazan header file (ako se i generiše neki kod u heder fajlu,obrišite ga), u ubacite deklaracije i typedef –ove u ovaj fajl.

KOD main.c:

```
#include "main.h"

int16_s temp; // definisana globalna promenjiva temp
void main(void)
{
}
```

Kod main.h

```
void main(void); // deklaracija main funkcije
typedef signed int int16_s; // definisan tip promenjive int16_s
extern int16_s temp; // deklarisana globalna promenjiva temp
```

Napomena: u slučaju da se `#include` ubacuju fajlovi iz drugog foldera ili da se koriste <> (npr `#include <main.h>` , PCLint će ove fajlove smatrati sistemskim bibiliotekama, i neće ih proveravati, osim ako se u podešavanjima ne podesi drugačije preko komande *–wlib.*

Kada se lintuje ovaj program, dobija se sledeća poruka

```
Info 714: Symbol 'temp' (line 3, file D:\GOOGLE_DRIVE\Materijal za
vezbe\autoelektronika\lekcije_za_sovu\vezba5-misra\misra-primer\misra-primer.X\main.c) not
referenced
D:\GOOGLE_DRIVE\Materijal za vezbe\autoelektronika\lekcije_za_sovu\vezba5-misra\misra-
primer\misra-primer.X\main.c:3: Info 830: Location cited in prior message
```

Ovde se ne radi o određenoj Misra smernici, već Pclint napominje da se promenjiva *temp* nigde ne koristi.

Ako se u mainu dodeli vrednost ovoj promenjivi,npr sa

```
void main(void)
{
    temp=5;
}
```

nestaje ova poruka, ali se javljaju LINT KOMENTARI

```
1. --- Module:   D:\GOOGLE_DRIVE\Materijal za vezbe\autoelektronika\lekcije_za_sovu\vezba5-misra\misra-primer\misra-primer.X\main.c (C)
Note 9003: could define variable 'temp' (line 3, file D:\GOOGLE_DRIVE\Materijal za vezbe\autoelektronika\lekcije_za_sovu\vezba5-misra\misra-primer\misra-primer.X\main.c) at block scope [MISRA 2012 Rule 8.9, advisory]
D:\GOOGLE_DRIVE\Materijal za vezbe\autoelektronika\lekcije_za_sovu\vezba5-misra\misra-primer\misra-primer.X\main.c:3: Info 830: Location cited in prior message

--- Global Wrap-up

2. Warning 552: Symbol 'temp' (line 3, file D:\GOOGLE_DRIVE\Materijal za vezbe\autoelektronika\lekcije_za_sovu\vezba5-misra\misra-primer\misra-primer.X\main.c) not accessed
D:\GOOGLE_DRIVE\Materijal za vezbe\autoelektronika\lekcije_za_sovu\vezba5-misra\misra-primer\misra-primer.X\main.c:3: Info 830: Location cited in prior message
```

Objašnjenje LINT komentara:

1. Pošto se promenjiva *temp* koristi samo u jednoj funkciji, tj u main funkciji, nema potrebe da se definise kao globalna promenjiva, već bi trebala kao lokalna. Pošto planiramo koristiti ovu promenjivu i negde drugde, ovaj komentar ćemo ignorisati

2. ovo je upozorenje da se promenjiva *temp* u suštini ne koristi nigde (jer smo joj samo dodelili vrednost i ništa drugo).

2.4 NAZIVI PROMENJIVIH I DELARACIJA FUNKCIJE

Ako napravimo novu funkciju, npr. Sabiranje, gde ćemo definisati lokalnu promenjivu *temp*, kao u sledećem kodu (i naravno deklarišemo tu funkciju u main.h), dobijaju se sledeći komentari

```c
#include "main.h"

int16_s temp; // definisana globalna promenjiva temp
void main(void)
{
    temp=5;

}
int16_s sabiranje(void)
{int16_s temp=5;
     return temp;
}
```

LINT KOMENTARI

```
--- Module:   D:\GOOGLE_DRIVE\Materijal za vezbe\autoelektronika\lekcije_za_sovu\vezba5-misra\misra-primer\misra-primer.X\main.c (C)


      _
```

{int16_s temp=5;

D:\GOOGLE_DRIVE\Materijal za vezbe\autoelektronika\lekcije_za_sovu\vezba5-misra\misra-primer\misra-primer.X\main.c:10: Warning 578: Declaration of symbol 'temp' hides symbol 'temp' (line 3) [MISRA 2012 Rule 5.3, required], [MISRA 2012 Rule 5.6, required], [MISRA 2012 Rule 5.7, required], [MISRA 2012 Rule 5.8, required], [MISRA 2012 Rule 5.9, advisory]

D:\GOOGLE_DRIVE\Materijal za vezbe\autoelektronika\lekcije_za_sovu\vezba5-misra\misra-primer\misra-primer.X\main.c:3: Info 830: Location cited in prior message

Note 9003: could define variable 'temp' (line 3, file D:\GOOGLE_DRIVE\Materijal za vezbe\autoelektronika\lekcije_za_sovu\vezba5-misra\misra-primer\misra-primer.X\main.c) at block scope [MISRA 2012 Rule 8.9, advisory]

D:\GOOGLE_DRIVE\Materijal za vezbe\autoelektronika\lekcije_za_sovu\vezba5-misra\misra-primer\misra-primer.X\main.c:3: Info 830: Location cited in prior message

--- Global Wrap-up

Warning 552: Symbol 'temp' (line 3, file D:\GOOGLE_DRIVE\Materijal za vezbe\autoelektronika\lekcije_za_sovu\vezba5-misra\misra-primer\misra-primer.X\main.c) not accessed

D:\GOOGLE_DRIVE\Materijal za vezbe\autoelektronika\lekcije_za_sovu\vezba5-misra\misra-primer\misra-primer.X\main.c:3: Info 830: Location cited in prior message

Info 714: Symbol 'sabiranje(void)' (line 9, file D:\GOOGLE_DRIVE\Materijal za vezbe\autoelektronika\lekcije_za_sovu\vezba5-misra\misra-primer\misra-primer.X\main.c) not referenced

D:\GOOGLE_DRIVE\Materijal za vezbe\autoelektronika\lekcije_za_sovu\vezba5-misra\misra-primer\misra-primer.X\main.c:9: Info 830: Location cited in prior message

U suštini jedini značajan novi tip komentara je prvi, boldovan, ovi ostali smo već imali u nekom obliku(govore da se promenjive ili funkcije ne koriste). Može se videti da se krše čak četiri misra smernice, suština je naziv lokalne promenjive ne sme biti isti kao i globalne (iako sa stanovništa C kompajlera je to sasvim prihvatljivo). Osnovni razlog zašto je to nedozvoljeno po Misra standardima je da se time smanjuje čitljivost koda, i povećava mogućnost greške jer programer može negde da pomeša lokalnu i globalnu promenjivu istog naziva.

Ispravka koda:

```
int16_s sabiranje(void)
{int16_s temp_local=5;
     return temp_local;
```

```
}
```

Nazvali smo lokalnu promenjivu *temp_local*, tako da se ne poklapa sa nazivom globalne promenjive.

Ako malo izmenimo ovu funkciju, tako da se u njoj stvarno dešava neko sabiranje, kao npr sa sledećim kodom

```
int16_s sabiranje(int16_s y)
{int16_s temp_local=5;
     return temp_local+y;
}
```

Pri čemu moramo da promenimo i deklaraciju, jer sad ova funkcija ima jedan parameter, tako da sad main.h izgleda

```
void main(void); // deklaracija main funkcije
typedef signed int int16_s; // definisan tip promenjive int16_s
extern int16_s temp; // deklarisana globalna promenjiva temp
int16_s sabiranje(int16_s);
```

dobija se sledeći LINT komentar od značaja

```
int16_s sabiranje(int16_s);
D:\GOOGLE_DRIVE\Materijal za vezbe\autoelektronika\lekcije_za_sovu\vezba5-misra\misra-primer\misra-primer.X\main.h:4: Note 955: Parameter name missing from prototype for function 'sabiranje' [MISRA 2012 Rule 8.2, required]
Note 9003: could define variable 'temp' (line 3, file D:\GOOGLE_DRIVE\Materijal za vezbe\autoelektronika\lekcije_za_sovu\vezba5-misra\misra-primer\misra-primer.X\main.c) at block scope [MISRA 2012 Rule 8.9, advisory]
D:\GOOGLE_DRIVE\Materijal za vezbe\autoelektronika\lekcije_za_sovu\vezba5-misra\misra-primer\misra-primer.X\main.c:3: Info 830: Location cited in prior message
```

Prekršeno je pravilo 8.2 (potpoglavlje 8.8), koje glasi

Rule 8.2	Function types shall be in *prototype form* with named parameters
	C90 [Undefined 22–25], C99 [Undefined 36–39, 73, 79]
Category	Required
Analysis	Decidable, Single Translation Unit
Applies to	C90, C99

Suština ovog pravila je da deklaracija mora biti potpuna, tj sa tačniim nazivima parametrima. Razlog je opet veća čitljivost koda i smanjenje pojavljivanja baga u kodu. Znači ispravka u kodu se svodi na sledeću deklaraciju, tj da se doda naziv parametra,ovde y .

```
int16_s sabiranje(int16_s y);
```

II deo

2.5 PRIMERI ZA RAZNE SMERNICE

2.5.1 Inicijalizacija niza

Prilikom inicijalizacije niza, potrebno je inicijalizovati sve članove niza, a ne samo deo članova.

```
int16_s z[ 3 ] = { 0, 1};
```

LINT KOMENTARI

```
{ int16_s z[ 3 ] = { 0, 1};
D:\GOOGLE_DRIVE\Materijal za vezbe\autoelektronika\lekcije_za_sovu\vezba5-misra\misra-primer\misra-primer.X\main.c:6: Info 785: Too few initializers for aggregate 'z' of type 'int16_s [3]'
D:\GOOGLE_DRIVE\Materijal za vezbe\autoelektronika\lekcije_za_sovu\vezba5-misra\misra-primer\misra-primer.X\main.c:6: Note 9068: partial array initialization [MISRA 2012 Rule 9.3, required]
```

Prekršeno je pravilo 9.3 (*Arrays shall not be partially initialized*). Cilj pravila je da ako se već vrši inicijalizacija članova niza, da se ona izvrši na svim članovima, jer inače programer može da previdi da inicijalizacija nije potpuna.

Ispravka koda:

```
int16_s z[ 3 ] = { 0, 1 ,10};
```

2.5.2 Korišćenje makroa

Ako u main.h ubacimo sledeće makroe, *sabrati* i *set_bit*

```
void main(void); // deklaracija main funkcije
typedef signed int int16_s; // definisan tip promenjive int16_s
extern int16_s temp; // deklarisana globalna promenjiva temp
int16_s sabiranje(int16_s);

#define sabrati(a, b) (a+b) //makro 1
#define set_bit(address,bit) (address |= (1<<bit)) //makro 2, sets BIT to 1
//in the register specified with ADDRESS
```

Dobijamo sledeće LINT komentare

```
#define sabrati(a, b) (a+b)

D:\GOOGLE_DRIVE\Materijal za vezbe\autoelektronika\lekcije_za_sovu\vezba5-misra\misra-primer\misra-primer.X\main.h:6: Note 9026: Function-like macro, 'sabrati', defined [MISRA 2012 Directive 4.9, advisory]

      _

#define    set_bit(address,bit) (address |= (1<<bit)) // sets BIT to 1 in the register specified with ADDRESS

D:\GOOGLE_DRIVE\Materijal za vezbe\autoelektronika\lekcije_za_sovu\vezba5-misra\misra-primer\misra-primer.X\main.h:7: Note 9026: Function-like macro, 'set_bit', defined [MISRA 2012 Directive 4.9, advisory]
```

Može se videti da se krši direktiva 4.9 (*A function should be used in preference to a function-like macro where they are interchangeable*).
U makrou sabrati sabiraju se parametri a i b, a makro tipa *set_bit* se često koristi u embedded sistemima da se setuje jedan bit u nekom registru (npr izlaznom portu kontrolera). Iako su ovi makroi u odredjenim situacijama veoma korisni, ista funkcionalnost se može dobiti i pomoću funkcija, ali se pomoću funkcija dobija veća provera koda, npr da se proveri da li su tipovi promenjive koji se sabiraju validni.

Što se tiče makroa saberi, realizacija funkcije iste funckcionalnosti je trivijalna. Kada se makro *set_bit* transformiše u funkciju, dobija se čitava grupa prekršenih pravila

```
void set_bit_funkcija (int16_s address, int16_s bit)
{
address |= ( 1<<bit);
}
```

LINT komentari

```
address |= ( 1<<bit);
D:\GOOGLE_DRIVE\Materijal za vezbe\autoelektronika\lekcije_za_sovu\vezba5-misra\misra-primer\misra-primer.X\main.h:12: Note 9027: Unpermitted operand to operator '<<' [MISRA 2012 Rule 10.1, required]
D:\GOOGLE_DRIVE\Materijal za vezbe\autoelektronika\lekcije_za_sovu\vezba5-misra\misra-primer\misra-primer.X\main.h:12: Note 9027: Unpermitted operand to operator '<<' [MISRA 2012 Rule 10.1, required]
D:\GOOGLE_DRIVE\Materijal za vezbe\autoelektronika\lekcije_za_sovu\vezba5-misra\misra-primer\misra-primer.X\main.h:12: Info 701: Shift left of signed quantity (int)
D:\GOOGLE_DRIVE\Materijal za vezbe\autoelektronika\lekcije_za_sovu\vezba5-misra\misra-primer\misra-primer.X\main.h:12: Note 9027: Unpermitted operand to operator '|' [MISRA 2012 Rule 10.1, required]
```

D:\GOOGLE_DRIVE\Materijal za vezbe\autoelektronika\lekcije_za_sovu\vezba5-misra\misra-primer\misra-primer.X\main.h:12: Note 9032: Composite expression with smaller essential type than other operand [MISRA 2012 Rule 10.7, required]

Može se videti da dolazi do višestrukog kršenja Misra pravila 10.1 u vezi operatora << (*Operands shall not be of an inappropriate* essential type) , ovo pravilo detaljno opisuje šta je zabranjeno pri korišćenju operatora.

Rationale 6 iz pravila 10.1

Shift and bitwise operations should only be performed on operands of essentially unsigned type. *The numeric value resulting from their use on* essentially signed types *is implementation defined.*

Ne smemo koristiti signed tip promenjive kod shift (<<) operatora (jer je pretostavka da pomeranje bita ima smisla samo kod unsigned promenjivih). Zbog toga ćemo prepraviti kod na tri mesta, koji su boldovani u sledećem kodu

```
void set_bit_funkcija (int8_u address, int8_u bit)
{


address |= ( 1u<<bit); // dodato u da bi se konstanta 1 smatrala unsigned tip

}
```

Parametri *address* i *bit* su promenjen iz tipa signed u unsigned , tj iz `int16_s` u **int8_u** , kojeg treba dodati u header fajlu preko typedefa

```
typedef unsigned char int8_u; // definisan tip promenjive int8_u
```

Pored izmene tipe parametra, može se videti da je broju 1 dodato malo slovo **u,** kojem se konstante definišu kao unsigned tip. Po defaultu su konstante signed tip, što je detaljnije objašnjeno u pravilu 7.2.

2.5.3 Beskonačna petlja

Ako se želi ubaciti beskonačna petlja ,tipa *while (1)* kao što je često slučaj u kodovima za kontrolere gde postoji takva petlja u main-u, dobijaju se sledeći Lint komentari

```
void main(void)
{  int16_s z[ 3 ] = { 0, 1, 3};
    temp=5;
   while (1)
   {// neki kod u beskonacnoj petlji
   }
}
```

LINT komentari

```
while (1)

C:\google_drive\Materijal za vezbe\autoelektronika\lekcije_za_sovu\vezba5-misra\misra-primer\misra-primer.X\main.c:8: Info 716: while(1) ...

C:\google_drive\Materijal za vezbe\autoelektronika\lekcije_za_sovu\vezba5-misra\misra-primer\misra-primer.X\main.c:8: Note 9036: Conditional expression should have essentially Boolean type [MISRA 2012 Rule 14.4, required]
```

Vidi se da se sa *while (1)* krši pravilo 14.4 (*The controlling expression of an if statement and the controlling expression of an iteration- statement shall have essentially Boolean type*) koja govori da se pri proverama uslova traži bool tip, dok se konstanta 1 u *while (1)* pod defaultu gleda kao int tip. Zbog toga je potrebno da se koristi operator konverzije (cast operator) nad broj 1, da bi ga kompjaler tretirao kao bool tip(verzija c programa C99 podržavai bool tip promenjive). Prvo je uradjen typedef bool tipa, kao što je ranije uradjeno za int i char

```
typedef _Bool bool_t;
```

a onda se u izvršila cast operacija u *while (1)*

```
while ((bool_t)1)
```

Pored Misra smernica, PC-lint softver daje dodatne informacije u mogućim greškama ili predloge za izmene koda, koje se mogu naći ili u fajlu *C:\lint\msg.txt* ili na web strani http://gimpel-online.com/MsgRef.html. Ove informacije u većini slučajeva mogu biti korisne sa dodatnim pojašnjenima mogućih problema u kodu, a ponekad daju i predloge koda za rešavanje konkretnih probema, kao u ovom slučaju.
Sa cast opraterom u beskonačnoj petlji nijedna Misra smernica više nije prekršena, ali ostaje Info 716 koju javlja PC-lint, koja glasi

```
while(1) ... -- A construct of the form while(1) ... was found.
      Whereas this represents a constant in a context expecting a
      Boolean, it may reflect a programming policy whereby infinite
      loops are prefixed with this construct.  Hence it is given a
      separate number and has been placed in the informational
      category. The more conventional form of infinite loop prefix is
      for(;;).
```

Kao što se može videti, predlaže se korišćenje for(;;) umesto *while (1)* , jer for(;;) ne krši nijedno Misra pravilo i generalno je pravilnije rešenje za pisanje beskonačne petlje.

Ako napišemo neki kod posle for(;;), (npr temp=7;) dobija se lint komentar

```
temp=7;
C:\google_drive\Materijal za vezbe\autoelektronika\lekcije_za_sovu\vezba5-misra\misra-primer\misra-primer.X\main.c:11: Warning 527: Unreachable code at token 'temp' [MISRA 2012 Rule 2.1, required]
```

Prekršeno je pravilo 2.1 (*A project shall not contain unreachable code*), tj govori da se deo kada nikad neće izvršiti i samim time nema svrhu, što možda nije bio cilj programera koji je pisao kod,

te se ovim pravilom naglašava moguća greška u kodu. U ovom slučaju je očigledno da se kod posle beskoančne petlje neće izvršiti, ali to ne mora biti tako očito prilikom korišćenja switch oepratera ili returna iz funcije (što je jedan od razloga zašto je zabranjeno koristiti dva returna iz funkcija po Misra pravilima, *Rule 15.5 A function should have a single point of exit at the end*) .

2.6 PRIMENA MODULA U PISANJU PROGRAMA

Kao i većina ostalih programskih jezika, i u C jeziku program se najčešće piše iz više fajlova, koji se nazivaju moduli. Kompletan program se deli u module tako da svaki modul na neki način čini logičku celinu, bilo da su u pitanju funkcije vezane za određeni hardver (hardverski drajveri), ili set funkcija nekog algoritamskog bloka.

Ako dodamo novi c fajl, (u source files) nazvan math.c, moramo generisati i odgovarajući header fajl math.h.

U math.c ubacite sve funkcije koje smo dosad definisali,osim naravno main funkcije.

math.c

```
#include "math.h"
int16_s sabiranje(int16_s prvi,int16_s drugi)
{int16_s rezultat;
rezultat=prvi + drugi;
return rezultat;
}

void set_bit_funkcija (int8_u address, int8_u bit)
{
address |= ( 1u<<bit);// dodato u da bi se konstanta 1 smatrala unsigned tip
}
```

A u math.h se nalaze deklaracije tih funkcija, kao i poziv ka main.h, jer se tamo nalaze typedef tipova promenjivih.

Kada se *lintuje* ovaj kod, dobijaju se sledeći Lint komentari od značaja

```
#include "main.h"

D:\GOOGLE_DRIVE\Materijal za vezbe\autoelektronika\lekcije_za_sovu\vezba5-misra\misra-primer\misra-primer.X\math.h:1: Warning 537: Repeated include file 'D:\GOOGLE_DRIVE\Materijal za vezbe\autoelektronika\lekcije_za_sovu\vezba5-misra\misra-primer\misra-primer.X\main.h'

D:\GOOGLE_DRIVE\Materijal za vezbe\autoelektronika\lekcije_za_sovu\vezba5-misra\misra-primer\misra-primer.X\math.h:1: Warning 451: Header file 'D:\GOOGLE_DRIVE\Materijal za vezbe\autoelektronika\lekcije_za_sovu\vezba5-misra\misra-primer\misra-primer.X\main.h' repeatedly included but does not have a standard include guard [MISRA 2012 Directive 4.10, required]
```

Ovim kodom prekršena je direktiva 4.10 (*Dir 4.10 Precautions shall be taken in order to prevent the contents of a header file being included more than once)*. Potrebno je dodati zaštitu (eng. *include guard*), da se ne bi desilo da se isti header fajl više puta učitava tokom kompajliranja, što može dovesti do

problema i greški prilikom kompajiranja. Ovaj problem se rešava korišćenejm pretprocesorskih komandi kao u primeru dolje, i u kodu koji vam saljem uz ovu vežbu u oba header fajla imate *include guard.*

```
/* File foo.  */

#ifndef FILE_FOO_SEEN

#define FILE_FOO_SEEN

the entire file

#endif /* !FILE_FOO_SEEN */
```

3 KRAJ

Prikazani su najčešći primeri kršenja Misra smernica, pored njih prilikom programiranja javiće vam se razni drugi Lint komentari, uz pravilno isčitavanje **MISRA-C:2012** moguće ih je veliku većinu rešiti, a za one koji se ne mogu rešiti potrebno je objasniti zašto je tako.

U izmene koda koje ćete morati često morati da izvršite da bi se ispoštovale Mista smernice spadaju i sledeće situacije, koje će biti samo nabrojane ali ne i ispravljene:

- ako neka promenljiva dodjeljuje svoju vrednost nekoj drugoj promenljivoj, ili se vrši poređenje između dvije vrijednosti, obje promjenljive treba da budu istog tipa(potrebno je izvršiti kastovanje).
- svaka naredba i petlja(if, for, while...) i naredbe unutar njih treba da budu oivečene vitičastim zagradama.
- pored promenljivih koje koristimo samo u jednom c fajlu, potrebno je dodati static prije njihovog tipa.