

Programski jezik C

Savremeni pristup i praksa

Dušan Gvozdenović

27. april 2019

Matematički fakultet

Tema predavanja

- Moderni principi programiranja u C-u
- Nestandardne osobine koje nude popularni prevodioci (*GCC, Clang*)
- Česti patterni u praksi
- Šta zaobići
- Kul trikovi o kojima se ne priča dovoljno

Zašto C?

- Brz
- Direktno upravljanje memorijom i memorijska efikasnost
- Minimalan
- Kao nekakav *high-level assembler*?
 - <https://wiki.gnome.org/Projects/Vala>
 - <https://nim-lang.org/>

Standardi

- “K&R” C
- ANSI-C (C89) / ISO-C90 (C90)
- ISO-C99 (C99)
- ISO-C11 (C11)

Definicija u *for*-petlji

```
1  for (int i = 0, j = 10; i < j; i+=2, j++) {  
2      printf("%d %d\n", i, j);  
3  }
```

Nizovi promenljive dužine

- Mogućnost da unutar funkcije kreiramo niz promenljive dužine
- Niz se čuva na steku^{*}
- Nije moguća statička inicijalizacija!

Nizovi promenljive dužine

```
1  int n;  
2  scanf("%d", &n);  
3  int a[n];  
4  for (size_t i = 0; i < n; i++) { a[i] = 0; } // Ovo je ok  
5  
6  int b[n] = { }; // Ovo ne prolazi
```

Makroi promenljive dužine

- C99 standard nam omogućava da definišemo makroe promenljive arnosti
- Koristimo ... kao kod funkcija sa promenljivim brojem parametara
- Koristimo makro `__VA_ARGS__` da ispišemo ostale argumente (odvojene zarezom)
- Takođe postoji makro `__VA_OPT__(T)` koji ispisuje T ako ima proizvoljnih parametara (primer kasnije)

Operatori # i

- # - Pretvara token u konstantan string (*stringify*)
- ## - Nadovezuje vrednost argumenta makroa na neki token (*token-pasting*)

Operatori # i

```
1  #define STR(X) #X
2
3  #define COMMAND(NAME) { #NAME, command_##NAME }
4
5  struct command commands[] = {
6      COMMAND(quit),
7      COMMAND(help),
8      ...
9  };
```

inline funkcije

- Uobičajeni način pozivanja funkcija
 - Vrednosti parametara se kopiraju
 - *Base pointer* se stavlja na stek
 - *Stack pointer* postaje *base pointer*
- Ovo može biti problem ukoliko se funkcija poziva u nekoj petlji i očekuje se brzo izvršavanje

inline funkcije

- Ako funkcija *f* poziva funkciju *g* i pritom definicije obe funkcije su prisutne u istoj jedinici prevođenja (TU), prevodioc može da zameni poziv funkcije *g* samim telom funkcije *g*
- Ključna reč *inline* daje sugestiju prevodiocu da je funkcija pogodna za inline-ovanje
- Napomena: Inline u C99 standardu ne funkcioniše kao u C++-u!

inline funkcije

- Ukoliko se funkcija inline-uje svugde, prevodioc ne mora ni da emituje njenu definiciju.
- Kada je funkcija pogodna za inline-ovanje?

Primer: *inline* funkcije

```
1  inline unsigned gcd(unsigned x, unsigned y)
2  {
3      if (x < y) { swap(&x, &y); }
4      if (x % y == 0) { return y; }
5      unsigned tmp;
6      while ((tmp = x % y) != 0) {
7          x = y;
8          y = tmp;
9      }
10     return y;
11 }
```

inline funkcije

- static inline
- extern inline

Compound statement expressions

- blok koda zapisan izmedju zagrada () se tumači kao vrednosni iskaz
- vrednost bloka je vrednost poslednjeg iskaza u njemu
- često se koriste u makroima koji trebaju da se ponašaju kao *inline* funkcije

Primer

```
1  #define max(type, fst, ...) ({ \
2      type __max__ = (fst); \
3      type __vals__[] = { __VA_ARGS__ };
4      for (size_t i = 0; i < sizeof(__vals__) / sizeof(type); i++) { \
5          if (__max__ < __vals__[i]) { \
6              __max__ = __vals__[i]; \
7          } \
8      } \
9      __max__; })
```

Compound literals

- Omogućavaju da dodeljujemo i prosledjujemo strukturne vrednosti promenljivama i funkcijama.
- Izgledaju kao kast operator na inicijalizator strukture

Designated Initializers

```
1  int num_widths[100] = { [0 ... 8] = 1, [9 ... 99] = 2 };
2
3  ...
4
5  struct person {
6      char * const name;
7      char * const surname;
8      int8_t age;
9  };
10
11  ...
12
13  struct person p = {
14      .name = "Dusan",
15      .surname = "Gvozdenovic",
16      .age = 20
17  };
```

_Generic

- Omogućava biranje izraza na osnovu pruženog tipa u vremenu prevođenja
- Sintaksa: `_Generic(<kontrolni-izraz>, <tip_1>: <izraz>, ..., <tip_k>: <izraz>, default: <izraz>)`
- Kontrolni izraz kao i izrazi svih grana koje nisu odabrane se nikada ne izvršavaju
- Napomena: default se ne mora navesti
- Napomena: Pre izbora se skidaju `const`, `volatile`, `restrict`, i `_Atomic`

Primer: _Generic

```
1  struct bool_option {
2      struct option option;
3      bool * const value;
4  };
5
6  struct int_option {
7      struct option option;
8      int * const value;
9  };
10
11 #define option_get(option) _Generic((option), \
12     struct bool_option: *(option).value, \
13     struct int_option: *(option).value, \
14     struct enum_option: (option).value, \
15     struct string_option: (option).value \
16 )
```

`_Noreturn`

- Specifikator funkcija koji navodi da funkcija nikad ne vraća
- Možemo koristiti `noreturn` makro iz `<stdnoreturn.h>` zaglavlja
- Za funkciju označenu sa `_Noreturn` se javljaju određene mogućnosti za optimizaciju:
 - Prevodioc može lakše da eliminiše i upozori o kodu koji se nikad ne izvršava

restrict ključna reč

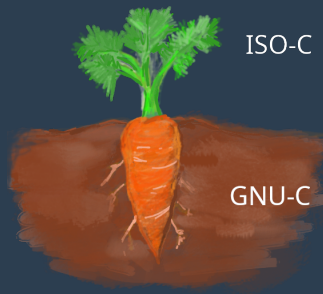
- Kvalifikator tipa koji služi da nagovesti prevodiocu da pokazivači ne pokazuju na istu memorijsku lokaciju (*aliasing*).
- Primer: Funkcije iz `string.h`

Celi brojevi određene dužine

- C standard ne određuje tačno širine celobrojnih tipova (int, short, long, long long) već samo navodi minimalne širine
- https://en.wikipedia.org/wiki/C_data_types

Nestandardna proširenja

- Pored osobina propisanih standardnom, GCC i Clang implementiraju i neke dodatne osobine
- GNU C - Dijalekt C-a koji gcc implementira
- <https://gcc.gnu.org/onlinedocs/gcc/C-Extensions.html#C-Extensions>



Dedukcija tipova

- `typeof, __auto_type`
- GNU C ima ograničenu podršku za dedukciju tipova.
- Ne funkcioniše sa nizovima.

Ugneždene funkcije

- Omogućava pisanje funkcija u funkcijama
- Unutrašnja funkcija ima pristup svim promenljivama spoljašnje (kao i globalnim statičkim, naravno)
- Oprez: Ukoliko pokušamo da pozovemo ugneždenu funkciju nakon što njena spoljašnja izađe iz opsega i pritom ona koristi njene lokalne promenljive, ponašanje je nedefinisano.

Ugneždene funkcije

```
1  walk_directory(const char *path, void (*callback)(const char *path));
2
3  ...
4
5  uint32_t new_docs = 0;
6
7  ...
8
9  walk_directory(dir, ({
10     void _wd(const char *path) {
11         struct document *doc;
12
13         if (file_get_magic_number(path) == FILE_PDF) {
14             doc = library_load_pdf_document(path);
15
16             if (!library_contains_document(doc)) {
17                 library_add(doc);
18                 new_docs++;
19             }
20         } else if (...) {
21             ...
22         }
23     } _wd; }));
```

Ugneždene funkcije

- Prednost: Enkapsulacija na nivou funkcije
- Mana: Kod funkcije se nalazi na steku i zbog toga mora biti omogućeno izvršavanje steka

EVERY TIME YOU DO THIS:



```
void foo(char *x) {  
    char buffer[500];  
    strcpy(buffer, x);  
    ...  
}
```

A KITTEN DIES.

Atributi u GCC-u

- Atributi nam omogućavaju da nagovestimo GCC-u da tretira promenljive, funkcije ili oznake na poseban način.
- Sintaksa: `__attribute__((<name>[(<params>)]))`
- <https://gcc.gnu.org/onlinedocs/gcc/Attribute-Syntax.html>

`__cleanup__`

- Atribut koji omogućava da se navedena *cleanup* funkcija pozove na kraju bloka
- Ovo možemo koristiti za implementiranje RAII uzorka i automatskog brojanja referenci!
- <https://github.com/Snaipe/libcsptr>

Primer: RAII

```
1  #define raii(type) __attribute__((__cleanup__((type##_destroy)))) struct type
2
3  struct person {
4      char *name;
5      char *surname;
6      unsigned age;
7  };
8
9  ...
10
11 int main() {
12     raii(person) *p = person_new("Petar", "Mitrovic", 26);
13     return 0;
14 }
```

Strukture podataka

```
1  struct linked_list {
2      void *data;
3      struct linked_list *prev, *next;
4  };
5
6  ...
7
8  void linked_list_add(struct linked_list *list, void *data);
9
10 void *linked_list_find(struct linked_list *list, void *value, bool (*equals)(void *a, void *b))
```

Strukture podataka

```
1  struct linked_list {  
2      struct linked_list *prev, *next; // Da li moze ovako?  
3  };
```

offsetof makro

```
1  #define offsetof(type, member) ((size_t)&((type *)0)->member)
2
3  /* GCC prepoznaje problem sa ovakvim pristupom i definise sintaksno prosirenje koje
4     radi istu stvar */
5
6  #define offsetof(type, member) __builtin_offsetof(type, member)
```

container_of makro

```
1  #define container_of(pointer, type, member) ({ \
2      const typeof(((type *)0)->member) __member = (pointer); \
3      (type *)((char *)__member - offsetof(type, member)); \
4  })
```

container_of makro

- Prednosti: Radimo sa konkretnim tipovima, obeshrabrujemo svakakvu redirekciju.
- Mane: Zahteva drugačiji način razmišljanja i pisanje više pomoćnih struktura.
- Šta se dešava kada je sadržana struktura deklarisan prva u strukturi?
- <https://github.com/torvalds/linux>
- <https://github.com/cmus/cmus>

X-makroi

Kuda dalje?

- GLib, GNOME stack
- <http://libcello.org/>
- <https://gcc.gnu.org/onlinedocs/gcc/C-Extensions.html>
- http://icube-icps.unistra.fr/img_auth.php/d/db/ModernC.pdf
- <https://www.youtube.com/user/Bisqwit>
- Secure Coding in C and C++, Robert C. Seacord

Pitanja?

Hvala na pažnji!