Структура на програма в C

типове, декларации, изрази, оператори и клаузи

Алекс Осенов, Александър Иванов, Александър Чернаев, Веселин Русинов, Пламен Тенев, Христо Стефанов

Технологично училище "Електронни системи",
Технически университет, София

17 февруари 2020 г.

Съдържание

Съдържание

Инструкции и формат на данни

- Компютърните процесори обработват данни, които се намират в RAM паметта на компютъра
- Обработването се извършва като процесорът изпълнява *последователност от инструкции* (*програма*), които манипулират данните
- Почти всеки съвременен процесор поддържа инструкции за работа с целочислени и реални числа
- За да може една инструкция да обработва данни те трябва предварително да са записани в определен вид (*формат*), с който инструкцията може да работи

Инструкции и формат на данни

```
# store 1 byte integer number 3 to address 0x1000
line1: storeb   $0b00000011, [$0x1000] # 3 in decimal
# store 1 byte integer number 1 to address 0x1001
line2: storeb   $0b00000001, [$0x1001] # 1 in decimal

# load 1 byte at address 0x1000 to register r1
line3: loadb    [$0x1000], %r1
# load 1 byte at address 0x1001 to register r2
line4: loadb    [$0x1001], %r2

# add the two 1 byte integer numbers found in registers r1 and r2
# and save the result in register r3
line5: addb     %r1, %r2, %r3
```

Фрагмент 1: Програма на асемблер за събиране на целочислени числа с големина 1 байт за измисления процесор PROG1

Инструкции и формат на данни

```
# store 4 byte real number 3.0 starting at address 0x1000
line1: storel   $0b01000000010000000000000000000000, [$0x1000]
# store 4 byte real number 1.0 starting at address 0x1004
line2: storel   $0b00111111100000000000000000000000, [$0x1004]

# load the 4 bytes starting at address 0x1000 to register fl1
line3: loadl    [$0x1000], %fl1
# load the 4 bytes starting at address 0x1004 to register fl2
line4: loadl    [$0x1004], %fl2

# add the two 4 byte real numbers found in registers fl1 and fl2
# and save the result in register fl3
line5: addf     %fl1, %fl2, %fl3
```

Фрагмент 2: Програма на асемблер за събиране на реални числа с големина 4 байта (IEE 754) за измисления процесор PROG1

Съдържание

Типове данни

- В езика C всички данни имат *тип*
- Типът определя какъв вид стойности и самите стойности, които могат да се съдържат в данните от типа. Също определя и какви операции могат да се извършват с тях.
- Знаейки какви данни, от какъв тип са и какви операции се извършват върху тях от програмата, компилаторът може да подбере правилните инструкции при генерирането на машинен код за определен процесор
- След като инструкциите се подберат, компилаторът записва данните в нужния вид за инструкциите

Типове данни

- Езикът C се опитва да поддържа възможно най-много видове процесори, което налага някои аспекти на езика да са специфицирани по-общо
- Например в повечето случаи броят байтове, които заема единица данни от определен тип, не е посочен
- От една страна това позволява една и съща програма да работи непроменена върху различни процесори
- От друга страна прави писането на коректни програми по-трудно

Основни типове в C

Тип данни	Описание
`signed char`	Цяло число със знак, което се събира в 1 байт памет. Може да съдържа поне числата [-127, +127]
`unsigned char`	Цяло число без знак, което се събира в 1 байт памет. Може да съдържа поне числата [0, 255]
`signed short int`	Цяло число със знак. Може да съдържа поне числата [-32767, +32767]
`unsigned short int`	Цяло число без знак. Може да съдържа поне числата [0, 65535]
`signed int`	Цяло число със знак. Може да съдържа поне числата [-32767, +32767]
`unsigned int`	Цяло число без знак. Може да съдържа поне числата [0, 65535]

Таблица 1: Основни целочислени типове данни в C (1/2)

Основни типове в C

Тип данни	Описание
`signed long int`	Цяло число със знак. Може да съдържа поне числата: [-2147483647, +2147483647]
`unsigned long int`	Цяло число без знак. Може да съдържа поне числата: [0, 4294967295]
`signed long long int`	Цяло число със знак. Може да съдържа поне числата: [-9223372036854775807, +9223372036854775807]
`unsigned long long int`	Цяло число без знак. Може да съдържа поне числата: [0, 18446744073709551615]

Таблица 2: Основни целочислени типове данни в C (2/2)

Основни типове в C

Тип данни	Описание
`float`	Реално число. Съдържа числа с малка точност.
`double`	Реално число. Съдържа числа с по-голяма точност.
`long double`	Реално число. Съдържа числа с възможно най-голяма точност.

Таблица 3: Реални типове данни в C

Съдържание

Декларации

- В C, за да се използват данни или подпрограми, те трябва първо да бъдат *декларирани*
- *Декларацията* служи за описване на различни атрибути, нужни на компилатора, свързани с определени данни
- Например, един от основните атрибути за данни е *типът данни*

Декларации на променливи

- Данните в C се наричат *променливи*
- Променливите имат следните основни атрибути - идентификатор (име), тип, начална стойност.
- Ако началната стойност не е посочена, то тя е произволна

Фигура 1: Декларация на променлива. Непълна синтактична диаграма

- Съществуват и *временни променливи*, които нямат име, но имат тип и зададена начална стойност. Те не могат да бъдат декларирани или променяни и се създават от компилатора за запазване на междинни резултати

Декларации на променливи

```
int x;
int a = 42;
int b = 'g';

char c = 97;

float f = 3.5;
double d = 5.43;
```

Фрагмент 3: Декларации на променливи

Съдържание

Изрази

- *Израз* (*expression*) е последователност от операции върху променливи, резултата от които е запазен във временна променлива
- За *стойност на израза* се счита стойността запазена във временната променлива
- За *тип на израза* се счита типа на временната променлива

Първични изрази

- *Първичен израз* се нарича израз, който съдържа само една константа, един литерал или един идентификатор.
  - Ако съдържа идентификатор, стойността и типът на временната променлива са тези на променливата посочена от идентификатора
  - Ако съдържа константа/литерал, стойността и типът на временната променлива са тези на константата/литерала

```
a /* type of 'a' */
1 /* int */
2.5 /* double */
'z' /* int */
"hello" /* char* (will be explained in a later lecture) */
```

Фрагмент 4: Примери за първични изрази и техните типове в C

Оператори

- Начинът, по който се обработват данни в C, е чрез *оператори*
- Операторите служат за обозначаване на операция върху една или повече *операнди*
- Операндите най-често са изрази[1]
- Операторите най-често се отбелязват с пунктуатор(и)
- Начинът на прилагане на оператор върху операнди варира

```
op1 + op2, -op1, op1 << op2, op1 >= op2,
op1 * op2, op1 / op2, sizeof(op1)
```

Фрагмент 5: Примери за изрази с един оператор в C

[1] Например могат да бъдат и имена на типове, които се използват от операцията по определен начин

Оператори

- Броят на операндите зависи от операцията, която се обозначава от оператора
- Например:
  - операция събиране - 2 операнди
  - операция смяна на знак - 1 операнда
- Спрямо броя операнди операциите се разделят на *унарни* (1 операнда), *бинарни* (2 операнди) и *тернарни* (3 операнди)
- Някои операции съдържат *странични ефекти* - след изпълнението им освен резултат, може да се наблюдават промени в някои от операндите

Особености на оператори

- Съществуват оператори, които се отбелязват с един и същ пунктуатор и се различават само по броя операнди
- Например операторът за смяна на знак и операторът за изваждане се отбелязват с '-'

```
-op1, op1 - op2
```

Фрагмент 6: Оператор за смяна на знак, оператор за изваждане

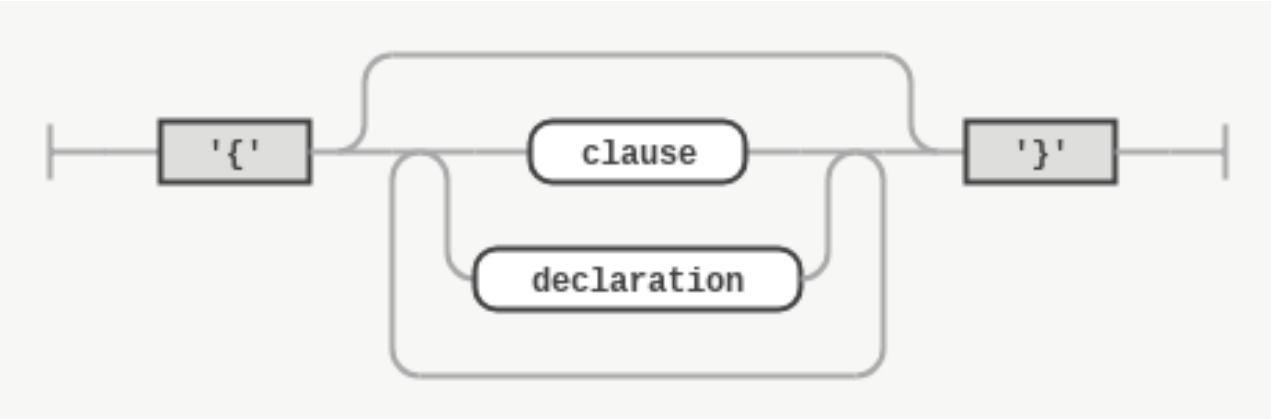
Съдържание

Клаузи

- *Клауза* (*твърдение*, *clause*/*statement*) е фрагмент от код, който се изпълнява последователно
- Съществуват пет вида клаузи
  - сложни клаузи
  - клаузи за изрази
  - клаузи за избор
  - клаузи за цикъл
  - клаузи за преход

Сложни клаузи. Блокове

- *Сложна клауза* (*блок*, *compound clause/block*) е клауза, която може да съдържа в себе си поредица от клаузи и/или декларации
- Сложна клауза, която не съдържа нито една клауза или декларация се нарича *празна сложна клауза/празен блок*



Фигура 2: Синтактична диаграма на сложна клауза (блок)

Сложни клаузи. Блокове

```
/* empty block */
{}

/* non-empty block */
{
    int a;
    int b = 5;
}
```

Фрагмент 7: Примери за сложни клаузи/блокове

Клаузи за изрази

- *Клауза за израз* (*expression clause*) е клауза, която може да съдържа в себе си израз, завършващ с ';'
- Изразът може да бъде изпуснат, като в такъв случай клаузата се нарича *празна клауза*

Фигура 3: Синтактична диаграма на клауза за израз

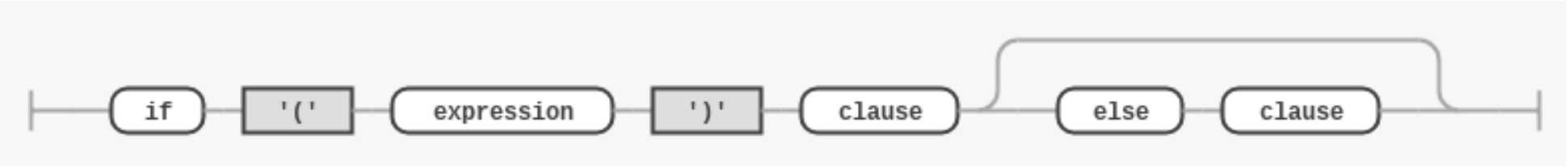
Клаузи за изрази

```
1 + 2 / 5;

a + b * c - 9123;

; /* empty clause */

(b + 6) / a - 300 % c;
```

Фрагмент 8: Примери за клауза за израз

Клаузи за избор. if клауза

- *Клаузите за избор* променят реда на изпълнение на програмата като избират между две клаузи, с които да се продължи изпълнението
- Например *if клаузата* избира на базата на стойността на някакъв израз (наречен *условие*) дали да изпълни една или друга клауза
- За да се избере първата клауза, стойността на условието трябва да бъде различна от 0. Ако стойността на условието е равна на 0 се изпълнява втората клауза, ако има такава



Фигура 4: Синтактична диаграма на `if` клауза

Клаузи за избор. if клауза

```
if (hour > 16) {
    printf("Good evening\n");
} else {
    printf("Good day\n");
}
```

Фрагмент 9: Пример за if клауза с else част

```
if (a > 1) {
    printf("Goodbye\n");
}
```

Фрагмент 10: Пример за if клауза без else част

Клаузи за цикъл. while клауза

- *Клаузите за цикъл* променят реда на изпълнение на програмата като повтарят дадена клауза. Може дадената клауза да не бъде изпълнена нито веднъж
- Например *`while` клаузата* изпълнява дадена клауза (наречена *тяло*) докато стойността на някакъв израз (наречен *условие*) е различна от 0
- Ако стойността на условието първоначално е равна на 0, тялото не се изпълнява нито веднъж

Фигура 5: Синтактична диаграма на while клауза

Клаузи за избор. while клауза

```
int i = 10;

while(i > 0) {
    printf("%d\n", i);
}
```

Фрагмент 11: Пример за while клауза

for клауза

- *for клаузата* е клауза за цикъл

Фигура 6: Синтактична диаграма на for клауза

- Първият израз във for клаузата се нарича *инициализация*, вторият - *условие*, а третият - *стъпка*. Дадената клауза след затварящата кръгла скоба се нарича *тяло*.
- Подобно на while клаузата, for клаузата изпълнява *тялото*, докато стойността на *условието* е различна от 0.
- Специфичното за for клаузата е, че *преди първата проверка* на *условието* се изпълнява израза за *инициализация*, и че след всяко изпълнение на тялото се изпълнява израза за *стъпка*.

Клаузи за избор. for клауза

```
for (i = 1; i <= 10; ++i) {
    printf("%d ", i);
}
```

Фрагмент 12: Пример за `for` клауза, която отпечатва числата от 1 до 10

for клауза

- Ако стойността на условието първоначално е равна на 0, тялото не се изпълнява нито веднъж
- Ако изразите за стъпка или инициализация са изпуснати, просто не се изпълняват
- Ако условието е изпуснато, винаги се влиза в тялото на `for` цикъла

for клауза

```
for (;;) {
    printf("hello!\n");
}
```

Фрагмент 13: Пример за `for` клауза, която предизвиква безкраен цикъл