Структура на програма в C

Част 5 - области на видимост, класове на съхранение

Иван Георгиев, Христо Иванов, Христо Стефанов

Технологично училище "Електронни системи",
Технически университет, София

16 април 2019 г.

Съдържание

Съдържание

Област на видимост

- Когато една променлива или функция се декларира, асоциираният с нея идентификатор започва да бъде разпознаван от компилатора
- С други думи идентификаторът става *видим*
- Областта от програмата, за която някой идентификатор е видим, определя *областта на видимост* на идентификатора (*scope*)

Файлова (глобална) област на видимост

- Областта на видимост на някой идентификатор зависи от мястото на неговото деклариране
- Ако декларацията е извън блок, областта на видимост започва от мястото на самата декларация, до края на файла
- Идентификатори с такава област на видимост се казва, че притежават *файлова (глобална) област на видимост* (*file/global scope*)

Файлова (глобална) област на видимост

```c
int a /* scope of 'a' starts here */ = 4;
int b /* scope of 'b' starts here */ = 3;

int countdown() /* scope of 'countdown' starts here */ {
    printf("%d\n", a);
    if (a > 0) {
        a = a - 1;
        countdown();
    }
    return a;
}

int main() /* scope of 'main' starts here */ {
    return countdown() + b;
}
/* scopes of 'a', 'b', 'countdown, 'main' end here */
```

Фрагмент 1: Файлова (глобална) област на видимост

Блокова (локална) област на видимост

- Ако декларацията е в блок, областта на видимост започва от мястото на самата декларация, до края на блока
- Идентификатори с такава област на видимост се казва, че притежават *блокова (локална) област на видимост* (*block/local scope*)

Блокова (локална) област на видимост

```c
int main() /* scope of 'main' starts here */ {
    int c /* scope of 'c' starts here */ = 10;
    int d /* scope of 'd' starts here */ = c + 15 + d;
    return c + d;
    /* scopes of 'c' and 'd' end here */
}
/* scope of 'main' ends here */
```

Фрагмент 2: Блокова (локална) област на видимост

Функционална област на видимост

- Променливите, декларирани като параметри на функция, са видими в цялата декларация на функцията
- Казваме, че техните идентификатори имат *функционална област на видимост*

```
int sum( /* scopes of 'a' and 'b' start here */ int a, int b)
{
    return a + b;
    /* scopes of 'a' and 'b' end here */
}
```

Фрагмент 3: Функционална област на видимост

Засенчване

- В C е позволено идентификатори декларирани в по-вътрешни области на видимост да съвпадат с идентификатори от по-външни области
- В такъв случай употребата на идентификатора води до използването на смисъла на най-вътрешната декларация за този идентификатор
- Това скриване на по-външния идентификатор, се нарича *засенчване*, тъй като той става недостъпен в областта на видимост на по-вътрешния идентификатор

```c
int main() {
    int main = 10; // OK, the variable 'main' shadows the
                   // function 'main' in this scope
    return main + 5;
}
```

Фрагмент 4: Засенчване на идентификатори

Засенчване

- Причината засенчването да е позволено е, че при писане на код, програмиста може да използва всеки идентификатор, без да се притеснява, че той може вече да е използван по-напред в програмата

```
int sum(int a, int b) {
    return a + b;
}
int main() {
    int sum = 0;
    int i = 0;
    for (i = 0; i < 10; i++) {
        sum = sum + i;
    }
    return 0;
}
```

Фрагмент 5: Засенчване на идентификатори

Съдържание

Време на живот на променливи

- При компилиране на една C програма е невъзможно да се определи колко точно памет е нужна на програмата за да бъде изпълнена
- Съответно не е възможно да се задели цялата нужна памет преди изпълнението ѝ
- Това води до нуждата от заделяне на памет по време на изпълнение на програмата
- В C паметта за променливите се заделя, само когато има нужда от тях, и в първия възможен момент се освобождава
- Времето между заделянето и освобождаването на паметта за една променлива се нарича *време на живот* на променливата

Класове на съхранение

- В C всяка променлива има определено време на живот
- Благодарение на това, компилатора може да генерира код, който да задели и освободи паметта за променливата без намеса от програмиста
- Времето на живот на всяка променлива се посочва в нейната декларация чрез избор на *клас на съхранение*
- Съществуват два класа на съхранение - *статичен* и *автоматичен*

Автоматичен клас на съхранение

- Паметта за променливите със автоматичен клас на съхранение (*автоматични променливи*) се заделя при първа нужда - в началото на тяхната област на видимост
- Паметта за променливите се освобождава автоматично, в края на тяхната област на видимост
- Една променлива се отбелязва, че притежава автоматичен клас на съхранение чрез ключовата дума `'auto'` в началото на нейната декларация.
- Ако не е посочен друг клас на съхранение, по подразбиране променливите с блокова област на видимост имат автоматичен клас на съхранение

```
int main() {
    auto int a = 10; // 'a' has automatic storage class
    int b = 10; // 'b' has automatic storage class as well
}
```

Фрагмент 6: Автоматичен клас на съхранение на променливи

Статичен клас на съхранение

- Паметта за променливите със статичен клас на съхранение (*статични променливи*) се заделя преди началото на програмата и се освобождава след нейното изпълнение
- Една променлива се отбелязва, че притежава статичен клас на съхранение чрез ключовата дума `'static'` в началото на нейната декларация
- Всички променливи с файлова област на видимост винаги имат статичен клас на съхранение - те не могат да имат автоматичен клас на съхранение

```
int g = 10; // 'g' has static storage class
int main() {
    static int a = 10; // 'a' has static storage class
}
```

Фрагмент 7: Статичен клас на съхранение

Статичен клас на съхранение

- Статичните променливи могат да бъдат инициализирани само с константи или изрази с операции върху константи
- След инициализация стойността на статичните променливи може да бъде променяна
- Тъй като времето на живот на статичните променливи обхваща времето за изпълнение на цялата програма, стойността им се запазва при повторно влизане в тяхната област на видимост

Статичен клас на съхранение

```c
int count_calls() {
    // initialization is done only once at the begining of the program
    static int c = 0;
    c = c + 1;
    return c;
}
int main() {
    printf("%d\n", count_calls()); // prints '1'
    printf("%d\n", count_calls()); // prints '2'
    printf("%d\n", count_calls()); // prints '3'
}
```

Фрагмент 8: Демонстриране на запазването на стойността на статична променлива