Отчётная работа
по практикуму на ЭВМ

Выполнила студентка 208 группы ВМК МГУ
Мазур Анастасия Вадимовна

ОГЛАВЛЕНИЕ

Задание 6

Проконтролировать, допускается ли инициализация переменных при описании; происходит ли инициализация по умолчанию.

Введение

Инициализация переменной - это процесс присваивания этой переменной некоторого начального значения, которое может быть изменено в ходе программы.

Несмотря на то, что во многих языках программирования описанным, но явно не инициалазированным переменным по умолчанию присваивается нулевое начальное значение, среди программистов всё же считается хорошим тоном в явном виде инициализировать переменную.

Такой стиль написания кода помогает избежать ошибок по невнимательности или при работе в команде.

Постановка задачи

В процессе выполнения данной задачи необходимо опытным путём выяснить, как работает инициализация переменных в языке Си:
- Можно ли инициализировать переменные сразу при их описании?
- Можно ли вовсе не инициализировать переменные? Если да, то определить, какое начальное значение компилятор присвоит переменным по умолчанию.

Примечание: для проверки программного кода, приведенного в листингах использовалась следующая версия компилятора - **gcc version 9.3.0**

Исследование задачи

Для того, чтобы ответить на вопросы задачи, отдельно рассмотрим переменные различных классов и типов. Так мы сможем сделать необходимые выводы об особенностях их инициализации.

Скалярные переменные можно инициализировать в их определениях, помещая после имени знак =

```
#include <stdio.h>
int
main(void) {
    int a = 208;
    int a1 = 0xAB1; // число в шестнадцатиричной системе счисления
    float b = 2.5;
    double b1 = 4E-2; // экспоненциальная запись числа
    char c = '#';
    long g = 20*4; // вычисление значения, а затем преобразование типов
    long f = g+5L; // выражение использует ранее определённое значение
    printf("integer a = %i\n", a);
    printf("integer a1 = %i\n", a1);
    printf("float b = %f\n", b);
    printf("double b1 = %lf\n", b1);
    printf("character c = %c\n", c);
    printf("long integer g = %li\n", g);
    printf("long integer f = %li\n", f);
    return 0;
}
```

Вывод:

```
integer a = 208
integer a1 = 2737
float b = 2.500000
double b1 = 0.040000
character c = #
long integer e = 80
long integer f = 85
```

Для внешних и статических переменных инициализирующие выражения должны быть константными, при этом инициализация осуществляется только один раз до начала выполнения программы.

Например, если приведённую выше программу дополнить следующим описанием переменной:

```
static int m = a1 - a;
```

то компилятор выдаст ошибку:

```
error: initializer element is not a compile-time constant
```

Инициализация автоматических и регистровых переменных выполняется каждый раз при входе в функцию или блок. Для таких переменных инициализирующее выражение - не обязательно константное. Это может быть любое выражение, использующее ранее определенные значения, включая даже и вызовы функций.

Рассмотрим пример:

```
#include <stdio.h>
void
func(void) {
    int a = 1;
    printf("a = %d\n", a++); // a == 2
    // но при следующем вызове функции a снова будет инициализирована 1;
}
int
main(void) {
    int i;
    for (i = 0; i < 3; i++) {
        func();
    }
    return 0;
}
```

Вывод:

```
a = 1
a = 1
a = 1
```

Что касается массивов, то их тоже можно инициализировать при описании. Массив инициализируется в своём определении с помощью заключённого в фигурные скобки списка инициализаторов, разделённых запятыми.

```
int arr[5] = {1,2,3,4,5};
```

Если размер массива не указан, то длину массива компилятор вычисляет по числу заданных инициализаторов.

Символьные массивы можно инициализировать не используя конструкцию с фигурными скобками, а задать значение с помощью строки.

Эти записи эквивалентны:

```
char arr[] = "praktikum";
```

```
char arr[] = {'p','r','a','k','t','i','k','u','m','\0'};
// Си строки оканчиваются символом '\0'
```

Таким образом, мы рассмотрели примеры инициализации разнообразных типов переменных и с уверенностью можем сделать вывод о том, что в языке Си можно инициализировать переменные сразу при их описании, это упрощает код. Однако необходимо учитывать особенности типа и производить только корректные присваивания.

Что же происходит в тех случаях, когда переменная не инициализирована? Произойдёт ли инициализация по умолчанию? Какое значение будет присвоено? Для проверки модифицируем написанную ранее программу:

```
#include <stdio.h>
int
main(void) {
    int a;
    float b;
    double b1;
    char c;
    long g;
    printf("integer a = %i\n", a);
    printf("float b = %f\n", b);
    printf("double b1 = %lf\n", b1);
    printf("character c = %c\n", c);
    printf("long integer g = %li\n", g);
    return 0;
}
```

Компилируя с ключом -Wall получаем большое количество предупреждений:

```
prog.c:11:32: warning: variable 'a' is uninitialized when used here [-Wuninitialized]
    printf("integer a = %i\n", a);
                               ^
prog.c:5:10: note: initialize the variable 'a' to silence this warning
    int a;
         ^
          = 0
prog.c:12:30: warning: variable 'b' is uninitialized when used here [-Wuninitialized]
    printf("float b = %f\n", b);
                             ^
prog.c:6:12: note: initialize the variable 'b' to silence this warning
    float b;
           ^
            = 0.0
prog.c:13:33: warning: variable 'b1' is uninitialized when used here [-Wuninitialized]
    printf("double b1 = %lf\n", b1);
                                ^~
prog.c:7:14: note: initialize the variable 'b1' to silence this warning
    double b1;
             ^
              = 0.0
prog.c:14:34: warning: variable 'c' is uninitialized when used here [-Wuninitialized]
    printf("character c = %c\n", c);
                                 ^
prog.c:8:11: note: initialize the variable 'c' to silence this warning
    char c;
          ^
           = '\0'
prog.c:15:38: warning: variable 'g' is uninitialized when used here [-Wuninitialized]
    printf("long integer g = %li\n", g);
                                    ^
prog.c:9:11: note: initialize the variable 'g' to silence this warning
    long g;
          ^
           = 0
```

Компилятор предлагает нам проинициализировать переменные нулём, но мы не слушаем его и запускаем программу. Получаем следующий вывод в консоль:

```
integer a = 325111845
float b = 0.000000
double b1 = 0.000000
```

```
character c =
long integer g = 0
```

Но не спешим радоваться полученному результату. Запустим программу ещё раз:

```
integer a = 226791461
float b = 0.000000
double b1 = 0.000000
character c =
long integer g = 0
```

Видим, что значение переменной типа integer изменилось. Оно будет разным при каждом запуске программы. Таким образом, мы готовы ответить на поставленный вопрос.

На этапе трансляции программы компилятор замечает, что переменные не инициализированы и предупреждает программиста об этом. Однако это не является фатальной ошибкой и трансляция не прерывается. Переменным типов float, double, char, long и т.п. (в том числе и массивы этих типов) присваивается нулевое значение. В переменной типа int же хранится, так скажем, мусор - случайное значение, которое осталось в той области памяти, которая была выделена под переменную. Это связано, в первую очередь, с оптимизацией работы программы: если нет необходимости в инициализации, то незачем тратить ресурсы для записи нулей.

Следовательно, делаем следующий вывод: во многих случаях компилятор поможет программисту и присвоит нули (хотя это может не всегда быть уместно, например, в теле программы переменная используется для умножения => результат будет занулён, это бесполезно).

Поэтому необходимо всегда инициализировать переменные исходя из поставленной задачи, чтобы не допускать ошибок.

Литература и веб-ресурсы

– Б. Керниган, Д. Ритчи
Язык программирования Си

– Н.В.Вдовикина, И.В.Машечкин, А.Н.Терехин, В.В.Тюляева
Программирование в ОС UNIX на языке Си

– **Онлайн справочник программиста на C и C++**

Задание 13

Определите, каким образом расширяются unsigned- варианты "малых"целых (до unsigned int как в "классическом"Си или сначала делается попытка расширить до int, и только в случае неудачи - до unsigned int).

Введение

Сложно не согласиться с утверждением, что для того, чтобы быть хорошим программистом, нужно понимать, каким образом выполняются процессы в компьютере: какие арифметические операции и преобразования стоят между написанным кодом и полученным результатом.

Именно поэтому в процессе изучения языка программирования Си особенно важно уделить должное внимание преобразованию типов, которое, как оказалось, может зависеть от реализации.

Постановка задачи

В процессе выполнения данного задания необходимо провести сравнительную характеристику реализаций компиляторов языка Си в прошлом, выяснить, каким образом работют компиляторы сейчас. А конкретнее, каким образом происходит расширение малых беззнаковых целых чисел.

Исследование задачи

Заглянем в историю: раньше не было единой спецификации, и поэтому разные компиляторы поступали разными способами, то есть они отличались в своей реализации. Существует два основных типа реализаций:
unsigned preserving (сохранение без знака) и **value preserving (сохранение значения).**

ANSI C использует для арифметических преобразований правила обработки с сохранением значения, в то время как реализации KR C склоняются к использованию правил операций без знака. Например, следующий код по стандарту ANSI C делает знаковое деление, где реализация unsigned-preserving делала бы деление без знака:

```
int
f(int x, unsigned char y) {
   return (x+y)/2;
}
```

Итак, опишем подходы чуть более подробно.

Подход с сохранением без знака требует приведения двух меньших типов без знака к unsigned int. Это простое правило, и оно дает тип, который не зависит от среды выполнения.

Подход с сохранением значений требует приведения малых типов к signed int, если этот тип может исправно вмещать все значения исходного типа, иначе приводится к unsigned int.

Если среда выполнения представляет short собой нечто меньшее, чем int, тогда unsigned short становится int; в противном случае становится unsigned int. Обе схемы дают один и тот же ответ в подавляющем большинстве случаев, и обе дают один и тот же эффективный результат в большинстве текущих реализаций. В таких реализациях различия между ними проявляются только тогда, когда выполняются следующие условия:

- Выражение, включающее unsigned char или unsigned short, дает int общий результат, в котором установлен знаковый бит: т.е. либо унарная операция для такого типа, либо двоичная операция, в которой другой операнд является int.
- Результат предыдущего выражения используется в контексте, в котором его знаковость имеет значение:

1. sizeof(int) < sizeof(long) и это ситуация, когда должно произойти расширение до длинного типа
2. это левый операнд оператора сдвига вправо (в реализации, где этот сдвиг определяется как арифметический)
3. это либо операнды /, %, <,<=,>, или >=.

В таких обстоятельствах возникает двусмысленность толкования. Одним из важных результатов изучения этой проблемы является понимание того, что высококачественным компиляторам может быть полезно искать такой неоднозначно интерпретируемый код и предлагать (необязательно) диагностику.

Правила сохранения без знака значительно увеличивают количество ситуаций, которые приводят к неоднозначному результату, в то время как правила сохранения значений сводят к минимуму такие ситуации. Таким образом, правила сохранения значения считались более безопасными для начинающих или неосторожных программистов. После долгих обсуждений Комитет принял решение в пользу правил сохранения значения, несмотря на то, что компиляторы UNIX развивались в направлении сохранения без знака.

Итак, ответ на поставленный вопрос задачи: сейчас все компиляторы расширяют малые целые следующим образом: делается попытка расширить до int, и только в случае неудачи - до unsigned int.

Литература и веб-ресурсы

– Н.В.Вдовикина, И.В.Машечкин, А.Н.Терехин, В.В.Тюляева
Программирование в ОС UNIX на языке Си

– **Rationale for American National Standard for Information Systems - Programming Language - C**

– microsin.net
Отличия кода ANSI C и кода KR C