

1 Задача 1.

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

1. Введение в задачу.

Во время объявления переменных можно сообщить им значение путем помещения знака равенства и константы после имени переменной. Этот процесс называется инициализацией и в общем случае имеет вид:

тип имя_переменной = константа;

Ниже приведено несколько примеров:

```
1 char c = 'a';
2 int number = 0;
3 float another_number = 123.23;
```

2. Постановка задачи.

Требуется узнать, допускается ли инициализация переменных различных типов при их объявлении, в этом нетрудно убедиться непосредственной проверкой. Происходит ли инициализация по умолчанию выясним с помощью экспериментов.

3. Исследование задачи.

Собственно, инициализация переменных действительно допускается при описании:

Листинг 1. p01_1.c

```
1 #include <stdio.h>
2
3 int main(void) {
4     int i = 1;
5     long j = 2345678;
6     char c = 'a';
7     double a = 123.456;
8
9
10    printf("%d\n", i);
11    printf("%ld\n", j);
12    printf("%c\n", c);
13    printf("%f\n", a);
14    return 0;
15 }
```

Вот что выводится:

```
P01 — -bash — 80x10
((base) MacBook-Pro-Dmitry:P01 dmitry$ gcc -Wall -o p01_1 p01_1.c
((base) MacBook-Pro-Dmitry:P01 dmitry$ ./p01_1
1
2345678
a
123.456000
(base) MacBook-Pro-Dmitry:P01 dmitry$
```

Теперь проверим, происходит ли инициализация по умолчанию:

Листинг 2. p01_2.c

```
1 #include <stdio.h>
2
3 int main(void) {
4     int i;
5     long j;
6     char c;
7     double a;
8
9     printf("%d\n", i);
10    printf("%ld\n", j);
11    printf("%c\n", c);
12    printf("%f\n", a);
13    return 0;
14 }
```

Вывод программы (с каждым выполнением разный результат):

```
P01 — -bash — 80x18
((base) MacBook-Pro-Dmitry:P01 dmitry$ ./p01_2
246272054
140732736223560
0.000000
((base) MacBook-Pro-Dmitry:P01 dmitry$ ./p01_2
301695030
140732817938760
0.000000
((base) MacBook-Pro-Dmitry:P01 dmitry$ ./p01_2
328212534
140732824066376
0.000000
(base) MacBook-Pro-Dmitry:P01 dmitry$
```

Общий вывод: неинициализированные переменные типа `int` и `long` по умолчанию принимают случайные значения, переменные типа `char` принимают значение `null`, а переменные типа `float` (или `double`) зануляются (компиляция проводилась под `gcc`).

2 Задача 2.

18. Определите, каким образом при выполнении операции присваивания и явном приведении происходит преобразование беззнаковых целых (М-битовое представление) к беззнаковым целым (N-битовое представление) при $M > N$, $M = N$, $M < N$.

1. Введение в задачу.

Присваивание — механизм связывания в программировании, позволяющий динамически изменять связи имён объектов данных (как правило, переменных) с их значениями. Переменной отводится место в памяти - какое-то количество бит (ячеек), зависит от типа переменной. Типы соответственно бывают целые, символьные, вещественные и так далее, все они размещаются в памяти линейно, занимая определенное число бит.

2. Постановка задачи.

Требуется узнать, что происходит с переменными в памяти после явного приведения типов, в данном случае при "расширении" или "урезании" отводимой для беззнаковой целочисленной переменной памяти. Это будет выяснено при помощи непосредственного побитового вывода.

3. Исследование задачи.

Выведем содержимое ячеек памяти до и после непосредственного преобразования типов (используя написанные функции побитового вывода):

Листинг 3. `p013.c`

```

1 #include <stdio.h>
2
3 void print_bit_short(short input) {
4     unsigned short mask = 1 << 15;
5     while(mask != 0) {
6         printf("%d", (input & mask) == 0 ? 0 : 1);
7         mask = mask >> 1;
8     }
9     printf("\n");
10 }
11
12 void print_bit_int(int input) {
13     unsigned int mask = 1 << 31;
14     while (mask != 0) {
15         printf("%d", (input & mask) == 0 ? 0 : 1);
16         mask = mask >> 1;
17     }
18     printf("\n");
19 }
20
21 int main(void) {
22     printf("short = %ld bytes\n", sizeof(short));

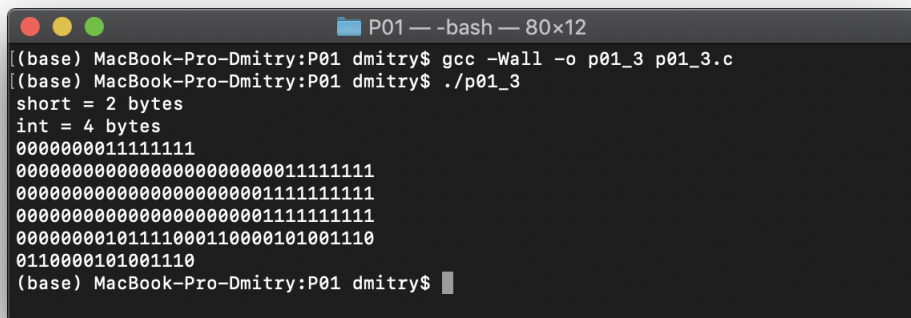
```

```

23     printf("int = %ld bytes\n", sizeof(int));
24
25     // M < N
26     unsigned short i = 255;
27
28     print_bit_short(i);
29     print_bit_int((int)i);
30
31     // M = N (where is the point?)
32     unsigned int j = 1023;
33
34     print_bit_int(j);
35     print_bit_int((int)j);
36
37     // M > N
38     unsigned int k = 12345678;
39
40     print_bit_int(k);
41     print_bit_short((short)k);
42
43     return 0;
44 }

```

Получаем следующий вывод:



```

P01 — -bash — 80x12
((base) MacBook-Pro-Dmitry:P01 dmitry$ gcc -Wall -o p01_3 p01_3.c
((base) MacBook-Pro-Dmitry:P01 dmitry$ ./p01_3
short = 2 bytes
int = 4 bytes
0000000011111111
00000000000000000000000011111111
00000000000000000000000011111111
00000000000000000000000011111111
00000000000000000000000011111111
000000001011100011000101001110
011000101001110
(base) MacBook-Pro-Dmitry:P01 dmitry$

```

Нетрудно видеть, что при преобразовании из short в int новые биты заполняются нулями, при преобразовании из int в int ничего не меняется (действительно), при преобразовании из int в short первые 2 байта просто "обрубаются".