Объединения

Объединение – это место в памяти, которое используется для хранения переменных, разных типов. Объединение дает возможность интерпретировать один и тот же набор битов не менее, чем двумя разными способами. Объявление объединения (начинается с ключевого слова union) похоже на объявление структуры и в общем виде выглядит так:

```
union тег {
  тип имя-члена;
  тип имя-члена;
  тип имя-члена;
  .
  .
  .
} переменные-этого-объединения;
```

Например:

```
union u_type {
  int i;
  char ch;
};
```

Это объявление не создает никаких переменных. Чтобы объявить переменную, ее имя можно поместить в конце объявления или написать отдельный оператор объявления. Чтобы с помощью только что написанного кода объявить переменную-объединение, которая называется cnvt и имеет тип u_type, можно написать следующий оператор:

```
union u_type cnvt;
```

В cnvt одну и ту же область памяти занимают целая переменная i и символьная переменная ch. Конечно, i занимает 2 байта (при условии, что целые значения занимают по 2 байта), а ch – только 1. На рис. 7.2 показано, каким образом i и ch пользуются одним и тем же адресом. В любом месте программы хранящиеся в cnvt данные можно обрабатывать как целые или символьные.

```
|<------ i ------>|
|                 |
+--------+--------+
| Байт 0 | Байт 1 |
+--------+--------+
|        |
|<- ch ->|
```

Рис. 7.2. Как i, так и ch, хранятся в объединении cnvt (подразумевается, что целые значения занимают по 2 байта)

Когда переменная объявляется с ключевым словом union, компилятор автоматически выделяет столько памяти, чтобы в ней поместился самый большой член нового объединения. Например, при условии, что целые значения занимают по 2 байта, для размещения i в cnvt необходимо, чтобы длина этого объединения составляла 2 байта, даже если для ch требуется только 1 байт.

Для получения доступа к члену объединения используйте тот же синтаксис, что и для структур: операторы точки и стрелки. При работе непосредственно с объединением следует пользоваться точкой. А при получении доступа к объединению с помощью указателя нужен оператор стрелка. Например, чтобы присвоить целое значение 10 элементу i из cnvt, напишите

```
cnvt.i = 10;
```

В следующем примере функции func1 передается указатель на cnvt:

```
void func1(union u_type *un)
{
  un->i = 10; /* присвоение cnvt значение 10 с помощью указателя */
}
```

Объединения часто используются тогда, когда нужно выполнить специфическое преобразование типов, потому что хранящиеся в объединениях данные можно обозначать совершенно разными способами. Например, используя объединения, можно манипулировать байтами, составляющими значение типа double, и делать так, чтобы менять его точность или выполнять какое-либо необычное округление.

Чтобы получить представление о полезности объединений в случаях, когда нужны нестандартные преобразования типа, подумайте над проблемой записи целых значений типа short в файл, который находится на диске.

В стандартной библиотеке языка С не определено никакой функции, специально предназначенной для выполнения этой записи.

Хотя данные любого типа можно записывать в файл, пользуясь функцией fwrite(), но было бы нерационально применять этот способ для такой простой операции, как запись на диск целых значений типа short, так как получится чрезмерный перерасход ресурсов. А вот, используя объединение, можно легко создать функцию putw(), которая по одному байту будет записывать в файл двоичное представление целого значения типа short. (В этом примере предполагается, что такие значения имеют длину 2 байта каждое.) Чтобы увидеть, как это делается, вначале создадим объединение, состоящее из целой переменной типа short и из массива 2-байтовых символов:

```
union pw {
  short int i;
  char ch[2];
};
```

Теперь с помощью pw можно написать вариант putw(), приведенный в следующей программе.

```
#include <stdio.h>
#include <stdlib.h>

union pw {
  short int i;
  char ch[2];
};

int putw(short int num, FILE *fp);

int main(void)
{
  FILE *fp;

  fp = fopen("test.tmp", "wb+");
  if(fp == NULL) {
    printf("Файл не открыт.\n");
    exit(1);
  }

  putw(1025, fp);  /* запись значения 1025 */
  fclose(fp);

  return 0;
}

int putw(short int num, FILE *fp)
{
  union pw word;

  word.i = num;

  putc(word.ch[0], fp); /* записать первую половину */
  return putc(word.ch[1], fp); /* записать вторую половину */
}
```

Хотя функция putw() и вызывается с целым аргументом типа short, ей для выполнения побайтовой записи в файл на диске все равно приходится использовать стандартную функцию putc().

Содержание | <<< | >>>
[Главная] [Гостевая]