ЛАБОРАТОРНАЯ РАБОТА № 29

ОБЪЕДИНЕНИЯ. БИТОВЫЕ ПОЛЯ. ПЕРЕЧИСЛЕНИЯ

Цель работы: получение навыков в организации программ с использованием объединений, перечислений и битовых полей.

Краткие теоретические сведения

Объединение – поименованная совокупность данных разных типов, размещаемых с учетом выравнивания в одной и той же области памяти, размер которой достаточен для хранения наибольшего элемента.

Объединенный тип данных декларируется подобно структурному типу:

```
union ID_объединения {
            описание полей    };
```

Пример описания объединенного типа:

```
union word {
        int nom;
        char str[20];   };
```

Пример объявления объектов объединенного типа:

```
union word *p_w,  mas_w[100];
```

Объединения применяют для экономии памяти в случае, когда объединяемые элементы логически существуют в разные моменты времени либо требуется разнотипная интерпретация поля данных.

Практически все вышесказанное для структур имеет место и для объединений. Декларация данных типа *union*, создание переменных этого типа и обращение к полям объединений производится аналогично структурам.

Пример использования переменных типа *union*:

```
        . . .
typedef union q {
        int a;
        double b;
        char s[5];
} W;
void main(void)
{
W  s, *p = &s;
s.a = 4;
printf("\n Integer a = %d, Sizeof(s.a) = %d", s.a, sizeof(s.a));
p –> b = 1.5;
printf("\n Double b = %lf, Sizeof(s.b) = %d", s.b, sizeof(s.b));
strcpy(p–>s, "Minsk");
printf("\n String a = %s, Sizeof(s.s) = %d", s.s, sizeof(s.s));
printf("\n Sizeof(s) = %d", sizeof(s));
}
```

Результат работы программы:
Integer a = 4, Sizeof(s.a) = 2
Double b = 1.500000, Sizeof(s.b) = 4
String a = Minsk, Sizeof(s.s) = 5
Sizeof(s) = 5

Перечисления – средство создания типа данных посредством задания ограниченного множества значений.

Определение перечисляемого типа данных имеет вид

enum *ID_перечисляемого_типа* {
список_значений };

Значения данных перечисляемого типа указываются идентификаторами, например:

```
enum marks { zero, one, two, three, four, five  };
```

Компилятор последовательно присваивает идентификаторам списка значений целочисленные величины 0, 1, 2,... . При необходимости можно явно задать значение идентификатора, то-

гда очередные элементы списка будут получать последующие возрастающие значения. Например:

```
enum level {
    low=100, medium=500, high=1000, limit   };
```

Константа *limit* по умолчанию получит значение, равное 1001.

Примеры объявления переменных перечисляемого типа:

```
enum marks Est;
enum level state;
```

Переменная типа *marks* может принимать только значения из множества {zero, one, two, three, four, five}.

Основные операции с данными перечисляемого типа:

– присваивание переменных и констант одного типа;

– сравнение для выявления равенства либо неравенства.

Практическое назначение перечисления – определение множества различающихся символических констант целого типа.

Пример использования переменных перечисляемого типа:

```
                . . .
typedef enum {
        mo=1, tu, we, th, fr, sa, su
} days;
void main(void)
{
days w_day;             // Переменная перечисляемого типа
int t_day, end, start;
// Текущий день недели, начало и конец недели соответственно
puts(" Введите день недели (от 1 до 7) : ");
scanf("%d", &t_day);
w_day = su;
start = mo;
end = w_day – t_day;
printf("\n Понедельник – %d день недели, \
сейчас %d – й день и \n\
до конца недели %d дн. ", start, t_day, end );
}
```

Результат работы программы:
Введите день недели (от 1 до 7) : 5
Понедельник – 1 день недели, сейчас 5-й день и
до конца недели 2 дн.

Битовые поля – это особый вид полей структуры. Они используются для плотной упаковки данных, например, флажков типа «*да*/*нет*». Минимальная адресуемая ячейка памяти – 1 байт, а для хранения флажка достаточно одного бита. При описании битового поля после имени через двоеточие указывается длина поля в битах (целая положительная константа), не превышающая разрядности поля типа *int*:

```
struct fields {
      unsigned int flag: 1;
      unsigned int mask: 10;
      unsigned int code: 5;     };
```

Битовые поля могут быть любого целого типа. Имя поля может отсутствовать, такие поля служат для выравнивания на аппаратную границу. Доступ к полю осуществляется обычным способом – по имени. Адрес поля получить нельзя, однако в остальном битовые поля можно использовать точно так же, как обычные поля структуры. Следует учитывать, что операции с отдельными битами реализуются гораздо менее эффективно, чем с байтами и словами, так как компилятор должен генерировать специальные коды и экономия памяти под переменные оборачивается увеличением объема кода программы. Размещение битовых полей в памяти зависит от компилятора и аппаратуры. В основном битовые поля размещаются последовательно в поле типа *int*, а при нехватке места для очередного битового поля происходит переход на следующее поле типа *int*. Возможно объявление безымянных битовых полей, а длина поля 0 означает необходимость перехода на очередное поле *int*:

```
struct areas {
    unsigned f1: 1;
                 : 2;   – безымянное поле длиной 2 бита;
    unsigned f2: 5;
                : 0        – признак перехода на следующее поле int;
```

```
    unsigned f3:5;
        double data;
char buffs[100];   };   // структура может содержать элементы
                       // любых  типов данных
```

Битовые поля могут использоваться в выражениях как целые числа соответствующей длины поля разрядности в двоичной системе исчисления. Единственное отличие этих полей от обычных объектов – запрет операции определения адреса (&). Следует учитывать, что использование битовых полей снижает быстродействие программы по сравнению с представлением данных в полных полях из-за необходимости выделения битового поля.

Порядок выполнения работы

1. Изучить теоретические сведения.
2. Ответить на контрольные вопросы.
3. Выполнить задание.

Контрольные вопросы

1. Дайте определение объединения.
2. Какое основное применение объединения?
3. Как осуществляется доступ к отдельным элементам объединения?
4. Дайте определение перечисления.
5. Какое основное применение переменных типа перечисления?

Задания для выполнения

При выполнении задания следует проанализировать приведенные фрагменты программы.

1. Приведенный ниже фрагмент программы предназначен для заполнения массива ***TABL***, т.е. для задания названий АСУ, которые разработаны на каждом заводе. При работе этого фрагмента следует ввести пять целочисленных значений переменной ***M*** (допустимыми являются целые числа от ***1*** до ***5***). Можно задать

и одинаковые значения ***M*** в том случае, если на заводах разработаны одинаковые АСУ.

```
for ( i=BOREC; i<=ZIL; i++)
{ printf(“ \nВведите число в диапазоне от 1 до 5 означающее: \n
1 – ACYP;\n
2 – ACYTP;\n
3 – ACYTO;\n
4 – ACYMIV;\n
5 - ACOI;\n “);
scanf(“ %d “, &M);
switch (M)
{  case 1: TABL[i]= ACYP; break;
   case 2: TABL[i]= ACYTP; break;
   case 3: TABL[i]= ACYTO; break;
   case 4: TABL[i]= ACYMIV; break;
   case 5: TABL[i]= ACOI; break;
   default: printf(“ \nНедопустимое значение”); break;
}}
```

При этом переменная ***i*** должна быть описана как ***enum zavod i;*** , а переменная ***M*** - как ***enum acy M; .***

2. Приведенный ниже фрагмент программы позволяет распечатать для каждой АСУ названия заводов, на которых она разработана:

```
for ( M = ACYP ; M <= ACOI ; M++ )
  { switch ( M )
case 1:  printf( " \n АСУП " ) ; break ;
case 2:  printf( " \n АСУТП " ) ; break ;
case 3:  printf( " \n АСУТО " ) ; break ;
case 4:  printf( " \n АСУМИВ " ) ; break ;
case 5:  printf( " \n АСОИ " ) ; break ;
}
printf( " разработана на заводах " ) ;
for( i = BOREC ; i <= ZIL ; i++ )
{ if ( mas [ i ] = = M )
switch ( i )
case 0:  printf ( " \n БОРЕЦ " ) ; break ;
case 1:  printf ( " \n ДЗЕРЖИНЕЦ " ) ; break ;
```

```
case 2:  printf ( " \n ДИНАМО " ) ; break ;
case 3:  printf ( " \n МАНОМЕТР " ) ; break ;
case 4:  printf ( " \n КОМПРЕССОР " ) ; break ;
case 5:  printf ( " \n ЗИЛ " ) ; break ;
 }
```

При этом переменная ***i*** должна быть описана как ***enum zavod i;*** , а переменная ***M*** - как ***enum acy M; .***

Варианты 1 – 7 задания по АСУ

Вывести на экран значение ***TRUE***, если среди заводов, информация о которых обрабатывается в программе, есть заводы, разработавшие АСУ одного типа, и значение ***FALSE*** - в противном случае.

1. Вывести на экран список заводов, разработавших АСУ вводимого типа.

2. Вывести на экран список заводов, разработавших АСУ одинакового типа.

3. Вывести на экран список заводов, разработавших АСУ разных типов.

4. Вывести на экран список АСУ, не разработанных ни на одном заводе.

5. Вывести на экран список АСУ, разработанных одновременно более чем на двух заводах.

6. Вывести на экран список АСУ, разработанных на всех заводах.

7. Вывести на экран список АСУ, разработанных одновременно менее чем на трех заводах.

Варианты 8 – 15 задания по странам

8. Вывести на экран список стран в порядке не убывания среднегодовой температуры.

9. Вывести на экран список стран в порядке не возрастания среднегодовой температуры.

10. Вывести на экран список стран, со среднегодовой температурой ниже заданной.

11. Вывести на экран список стран, со среднегодовой температурой выше заданной.

12. Вывести на экран название страны, где самая высокая среднегодовая температура.

13. Вывести на экран название страны, где самая низкая среднегодовая температура.

14. Для указанной страны вывести на экран названия месяцев, в которых среднемесячная температура ниже заданной.

15. Для указанной страны вывести на экран названия месяцев, в которых среднемесячная температура выше заданной.