Лабораторная работа №10

Тема: «Структуры, перечисления, объединения»

Цель работы: Изучить синтаксис и правила работы со структурами. Реализовать программу с применением структур, перечислений и объединений.

Теоретический сведения

При разработке программ важным является выбор эффективного способа представления данных. Во многих случаях недостаточно объявить простую переменную или массив, а нужна более гибкая форма представления данных. Таким элементом может быть структура, которая позволяет включать в себя разные типы данных, а также другие структуры.

Структура – это составной тип данных, в котором под одним именем объединены данные различных типов. Отдельные данные структуры называются полями. Объявление структуры осуществляется с помощью ключевого слова struct, за которым указывается ее имя и список элементов, заключенных в фигурные скобки:

```
struct имя {
тип_элемента_1 имя_элемента_1;
тип_элемента_2 имя_элемента_2;
...
тип_элемента_nимя_элемента_n;
} ;
```

Правила работы с полями структуры идентичны работе с переменными соответствующих типов. К полям структуры можно обращаться через составное имя. Формат обращения:

```
имя_структуры.имя_поля
```

или

```
указатель_на_структуру->имя_поля
```

Приведем пример, в котором использование структуры позволяет эффективно представить данные. Таким примером будет инвентарный перечень книг, в котором для каждой книги необходимо указывать ее наименование, автора и год издания. Причем количество книг может быть разным, но будем полгать, что не более 100. Для хранения информации об одной книге целесообразно использовать структуру, которая задается в языке С с помощью ключевого слова struct, за которым следует ее имя. Само определение структуры, т.е. то, что она будет содержать, записывается в фигурных скобках {}. В данном случае структура будет иметь следующий вид:

```
struct book {
```

```
char title[100]; //наименование книги
char author[100]; //автор
int year; //год издания
};
```

Такая конструкция задает своего рода шаблон представления данных, но не сам объект, которым можно было бы оперировать подобно переменной или массиву. Для того чтобы объявить переменную для структуры с именем book используется такая запись:

```
struct booklib; //объявляется переменная типа book
```

После объявления переменной lib имеется возможность работать со структурой как с единым объектом данных, который имеет три поля: title, author и year. Обращение к тому или иному полю структуры осуществляется через точку: lib.title, lib.author и lib.year. Таким образом, для записи в структуру информации можно использовать следующий фрагмент программы:

```
printf("Введите наименование книги: ");
scanf("%s",lib.title);
printf("Введите автора книги: ");
scanf("%s",lib.author);
printf("Введите год издания книги: ");
scanf("%d",&lib.year);
```

После этого в соответствующие поля будет записана введенная с клавиатуры информация и хранится в единой переменной lib. Однако по условиям задачи необходимо осуществлять запись не по одной, а по 100 книгам. В этом случае целесообразно использовать массив структур типа book, который можно задать следующим образом:

```
struct booklib[100];
```

В этом случае программу ввода и хранения информации по книгам можно записать в виде:

Листинг 1. Инвентарный перечень книг.

```
    #include<stdio.h>
struct book {
char title[100]; //наименование книги

char author[100]; //автор

int year; //год издания

};

int main() {

int cnt_book = 0, ch;
```

```
struct book lib[100];
do
     {
printf("Введите наименование книги: ");
scanf("%s",lib[cnt_book].title);
printf("Введите автора книги: ");
scanf("%s",lib[cnt_book].author);
printf("Введите год издания книги: ");
scanf("%d",&lib.year);
printf("Нажмите q для завершения ввода: ");
cnt_book++;
     }
while(scanf("%d",ch) == 1 &&cnt_book< 100);
return 0;
}
```

Данный пример показывает удобство хранения информации по книгам. Тот же алгоритм в общем случае можно реализовать и без структуры, но тогда пришлось бы использовать два двумерных массива символов и один одномерный массив для хранения года издания. Несмотря на то, что формально такая запись была бы корректной с точки зрения языка С, но менее удобна в обращении. Графически массив структур можно представить в виде таблицы, в которой роль столбцов играют поля, а роль строк элементы массива структур.

	название	автор	год издания
lib[0]	lib[0].title	lib[0].author	lib[0].year
lib[1]	lib[1].title	lib[1].author	lib[1].year
lib[2]	lib[2].title	lib[2].author	lib[2].year
⋮			
lib[99]	lib[99].title	lib[99].author	lib[99].year

Структуры можно автоматически инициализировать при их объявлении подобно массивам, используя следующий синтаксис:

```
struct book lib  = {
     "Евгений Онегин",
     "Пушкин А.С.",
     1995
};
```

При выполнении данного фрагмента программы в переменные структуры title, author и year будет записана соответственно информация: “Евгений Онегин”, “Пушкин А.С.”, 1995. Здесь следует обратить внимание, что последовательность данных при инициализации должна соответствовать последовательности полей в структуре. Это накладывает определенные ограничения, т.к. при инициализации

необходимо помнить последовательность полей в структуре. Стандарт C99 допускает более гибкий механизм инициализации полей структуры:

```
   struct book lib  = {.year = 1995,
                       .author = "Пушкин А.С.",
                      .title = "Евгений Онегин" };
```

или

```
struct book lib  = { .year = 1995,
                    .title = "Евгений Онегин" };
```

или

```
struct book lib  = {.author = "Пушкин А.С.",
                    .title = "Евгений Онегин",
                   1995 };
```

В первом и во втором примерах при инициализации указываются наименования полей через точку. При этом их порядок и число не имеет значения. В третьем примере первые два поля указаны через имена, а последнее инициализируется по порядковому номеру – третьему, который соответствует полю year.

В некоторых случаях имеет смысл создавать структуры, которые содержат в себе другие (вложенные) структуры. Например, при создании простого банка данных о сотрудниках предприятия целесообразно ввести, по крайней мере, две структуры. Одна из них будет содержать информацию о фамилии, имени и отчестве сотрудника, а вторая будет включать в себя первую с добавлением полей о профессии и возрасте:

```
struct tag_fio {
char last[100];
char first[100];
char otch[100];
    };
struct tag_people {
         struct tag_fio fio; //вложенная структура
         char job[100];
         int old;
};
```

Рассмотрим способ инициализации и доступ к полям структуры people на следующем примере.

Листинг 2. Работа с вложенными структурами.

```
int main() {

struct tag_people man = {
          {"Иванов", "Иван", "Иванович"},
```

```
            “Электрик”,
            50 };
printf(“Ф.И.О.:%s %s %s\n”,man.fio.last,man.fio.first,
man.fio.otch);
printf(“Профессия : %s \n”,man.job);
printf(“Возраст : %d\n”,man.old);
return 0;
}
```

В данном примере показано, что для инициализации структуры внутри другой структуры следует использовать дополнительные фигурные скобки, в которых содержится информация для инициализации полей фамилии, имени и отчества сотрудника. Для того чтобы получить доступ к полям вложенной структуры выполняется сначала обращение к ней по имени man.fio, а затем к ее полям: man.fio.last, man.fio.first и man.fio.otch. Используя данное правило, можно создавать многоуровневые вложения для эффективного хранения и извлечения данных.

Пример

Создать массив структур, содержащий информацию о студентах: ФИО, номер группы, оценки за последнюю сессию. Вывести информацию о студен- тах группы 610205 в порядке убывания среднего балла.

```
#include<iostream.h>
#include<string.h>
int main ()
{
struct strc { // Объявление структуры strc
char fio[40];
char ngr[7];
int otc[4];
double sb;
} mstud[100]; // Объявление массива структур mstud

int nst, i, j; cout<< "Vveditekol-vostudentov" <<endl;
cin>>nst;
for (i=0; i <nst; i++) // Ввод информации о студентах
{
cout<< "Vvedite FIO: ";
cin>>mstud[i].fio;
cout<< "Vveditenomergr: ";
cin>>mstud[i].ngr;
cout<< "Vvedite 4 otcenki" <<endl;
mstud[i].sb = 0;
for (j=0; j<4; j++)
// Ввод четырех оценок // за последнюю сессию
{ cin>>mstud[i].otc[j];
mstud[i].sb += mstud[i].otc[j] / 4.; }
// Вычисление
```

```
// среднего балла студента
cout<<endl;
}
strc stemp;
for (i=0; i < nst-1; i++) // Сортировка по среднему баллу

for (j=i+1; j<nst; j++)
if (mstud[i].sb<mstud[j].sb
     && !strcmp (mstud[i].ngr, "610205")
     && !strcmp (mstud[j].ngr, "610205"))
{
     mstud[i] = mstud[j];
}
mstud[j] =stemp;
for (i=0; i <nst; i++)
     if (!strcmp (mstud[i].ngr, "610205"))
     // Вывод информации
          cout<<mstud[i].fio<< " " <<mstud[i].ngr<< " "
     <<mstud[i].sb<<endl;
return 0;
}
```

Объединения

Объединение - это поименованная совокупность данных разных ти- пов, размещаемых с учетом выравнивания в одной и той же области памя- ти, размер которой достаточен для хранения наибольшего элемента.

Объединенный тип данных декларируется подобно структурному:

```
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 объе- диняемые элементы логически существуют в разные моменты времени либо требуется разнотипная интерпретация поля данных.

Например, пусть поток сообщений по каналу связи содержит сообще ния трех видов:

```
struct m1 {
char code;
float data[100]; };

struct m2 {
char code;
int mode; };

struct m3 {
char code,
note[80]; };
```

Элемент code - признак вида сообщения. Удобно описать буфер для

хранения сообщений в виде

```
struct m123 {
  char code;
  union {
     float data[100];
     int mode;
     char note[80];
  };
};
```

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;
     float 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 Float b = %f, 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
Float b = 1.500000,
Sizeof(s.b) = 4
String a = Minsk,
Sizeof(s.s) = 5
Sizeof(s) = 5
```

Перечисление

Перечисления - это средство создания типа данных посредством задания ограниченного множества значений. Определение перечислимого типа данных имеет вид

```
enum ID_перечислимого типа {
список значений };
```

Значения данных перечислимого типа указываются идентификаторами, например:

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

Транслятор последовательно присваивает идентификаторам списка зна- чений целочисленные величины 0,1,..., . При необходимости можно явно задать значение идентификатора, тогда очередные элементы списка бу- дут получать последующие возрастающие значения. Например:

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

Примеры объявления переменных перечислимого типа:

```
enum marks Est;
```

`enum level state;` Переменная типа marks может принимать только значения из множества {zero, two, three, four, five}.

Основные операции с данными перечислимого 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 перечислимого типа:

...

```
typedef enum { mo=1, tu, we, th, fr, sa, su } days;
```

```
void main(void)
{
     days w_day; // Переменная перечислимого типа
     // Текущий день, начало и конец недели
     int t_day, end, start;
     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-й день недели, сейчас %d-й
день. \n\ До конца недели %d дней (дня). ", start, t_day,
end );
}
```

Результат работы программы:

```
Введите день недели (от 1 до 7) :
2
Понедельник - 1-й день недели, сейчас 2-й день.
До конца недели 5 дней (дня).
```

Битовые поля

Битовые поля применяются для максимально полной упаковки информации, если не важна скорость доступа к этой информации.

В противоположность другим компьютерным языкам С имеет возможность, называемую битовыми полями, позволяющую работать с отдельными битами. Битовые поля полезны по нескольким причинам. Ниже приведены три из них:

1. Если ограничено место для хранения информации, можно сохранить несколько логических (истина/ложь) переменных в одном байте.
2. Некоторые интерфейсы устройств передают информацию, закодировав биты в один байт.
3. Некоторым процедурам кодирования необходимо получить доступ к отдельным битам в байте.

Хотя все эти функции могут выполняться с помощью битовых операторов, битовые поля могут внести большую ясность в программу.

Метод использования битовых полей для доступа к битам основан на структурах. Битовое поле, на самом деле, - это просто особый тип структуры, определяющей, какую длину имеет каждый член. В языках С и С++ при объявлении битового поля используется символ двоеточия (:). После двоеточия

указывается константное выражение, определяющее количество битов в битовом поле. Стандартный вид объявления битовых полей следующий:

```
struct имя структуры {
тип имя1: длина;
тип имя2: длина;
...
тип имяN: длина;
}
```

Битовые поля должны объявляться как int, unsigned или signed. Битовые поля длиной 1 должны объявляться как unsigned, поскольку 1 бит не может иметь знака. Битовые поля могут иметь длину от 1 до16 бит для 16-битных сред и от 1 до 32 бит для 32-битных сред.

Пример:

```
struct rgb
{
unsigned r:3;
unsigned g:10;
unsigned b:1;
};
```

Битовые поля имеют некоторые ограничения. Нельзя получить адрес переменной битового поля. Переменные битового поля не могут помещаться в массив. Переходя с компьютера на компьютер нельзя быть уверенным в порядке изменения битов (слева направо или справа налево). Любой код, использующий битовые поля, зависит от компьютера.

Наконец, можно смешивать различные структурные переменные в битовых полях. Например:

```
struct emp {
struct addraddress;
float pay;
unsigned lay_off:1;
unsigned hourly:1;
unsigned deductions:3;
};
```

определяет запись служащего, использующую только один байт для хранения трех частей информации - статуса служащего, получил ли он зарплату и размер удержаний. Без использования битовых полей данная информация заняла бы три байта.

Задание:

Создать тип структуры согласно варинту, организовать поля этой структуры так, чтобы они содержали объединение, перечисление (можно добавить дополнительные поля) и битовое поле.

Создать массив структур, содержащий информацию согласно варианту индивидуального задания.

Реализовать работу с массивом структур через меню: ввод данных в массив, вывод собержимого массива на экран, сортировка по одному полю, удаления записи по заданному значению поля, выборка записей согласно индивидуального задания.

Варианты задания:

1. В магазине сформирован список постоянных клиентов, который включает ФИО, домашний адрес покупателя и размер предоставляемой скидки. Вывести всех покупателей, имеющих 5 % - ную скидку.

2. Список товаров, имеющихся на складе, включает в себя наименование товара, количество единиц товара, цену единицы и дату поступления товара на склад. Вывести список товаров, стоимость которых превышает 100 000 рублей.

3. Для получения места в общежитии формируется список студентов, ко- торый включает ФИО студента, номер группы, средний балл, доход на члена семьи. Вывести фамилии студентов, у которых доход на члена семьи меньше двух минимальных зарплат.

4. В справочной автовокзала имеется расписание движения автобусов. Для каждого рейса указаны его номер, тип автобуса, пункт назначения, время отправления и прибытия. Вывести информацию о рейсах, которыми можно воспользоваться для прибытия в пункт назначения раньше заданного времени.

5. На междугородной АТС информация о разговорах содержит дату разговора, код и название города, время разговора, тариф, номер телефона абонента. Вывести для заданного города общее время разговоров с ним и сумму.

6. Информация о сотрудниках фирмы включает ФИО, табельный номер, количество отработанных часов за месяц, почасовой тариф. Вывести размер заработной платы каждого сотрудника.

7. Информация об участниках спортивных соревнований содержит название страны, название команды, ФИО игрока, игровой номер, возраст, рост и вес. Вывести фамилии спортсменов, возраст которых больше 20 лет.

8. Для книг, хранящихся в библиотеке, задаются регистрационный номер книги, автор, название, год издания, издательство, количество страниц. Вывести список книг с фамилиями авторов, изданных после заданного года.

9. Различные цеха завода выпускают продукцию нескольких наименований. Сведения о выпущенной продукции включают наименование, количество, номер цеха. Для заданного цеха вывести количество выпущенных изделий.

10. Информация о сотрудниках содержит ФИО, номер отдела, должность, стаж работы на предприятии. Вывести список сотрудников заданного отдела, имеющих стаж работы на предприятии более 20 лет.

11. Ведомость абитуриентов содержит ФИО, адрес, оценки по трем предметам. Определить средний балл абитуриентов, проживающих в городе Минске.

12. В справочной аэропорта имеется расписание вылета самолетов. Для каждого рейса указаны его номер, тип самолета, пункт назначения, время вылета. Вывести все номера рейсов, вылетающих в заданный пункт назначения.

13. Сведения об автомобиле состоят из номера, марки, фамилии владельца, признака прохождения техосмотра. Для каждой марки подсчитать количество автомобилей этой марки.

14. У администратора железнодорожных касс имеется информация о свободных местах в поездах на текущие сутки в следующем виде: пункт назначения, время отправления, число свободных мест. Вывести информацию о числе свободных мест в поездах, следующих до заданного пункта назначения.

15. Ведомость абитуриентов, сдавших вступительные экзамены в университет, содержит ФИО абитуриента и его оценки. Определить средний балл по университету и вывести список абитуриентов, средний балл которых выше среднего балла по университету.

16. В радиоателье хранятся квитанции о сданной в ремонт радиоаппаратуре. Каждая квитанция содержит наименование изделия, дату приемки в ремонт, состояние готовности заказа (выполнен, не выполнен). Вывести информацию об изделиях, ремонт которых еще не выполнен.

17. Структура содержит следующие поля: страна, название футбольного клуба, имя футболиста, количество забитых мячей в сезоне, количество предупреждений. Написать функцию для нахождения наиболее полезного игрока сезона.

18. Структура содержит следующие поля: название процессора, внутренняя частота процессора, количество ядер, объем кэш-памяти. Написать функцию для нахождения двухъядерного процессора с самой низкой внутренней частотой.

19. Структура содержит следующие поля: название сайта, URL сайта, краткое описание, количество посещений за день. Написать функцию для нахождения наиболее посещаемого сайта.

20. Структура содержит следующие поля: пациент, пол, возраст, заболевание, прогноз (благоприятный/неблагоприятный). Написать функцию для определения наиболее опасного заболевания.

21. Структура содержит следующие поля: марка мотоцикла, фирма

изготовитель, тип, пробег, объем двигателя. Написать функцию для нахождения мотоцикла с самым меньшим пробегом.

22. Структура содержит следующие поля: страна, название города, население города, доля населения школьного возраста (в процентах). Написать функцию для нахождения страны с самым большим количеством детей.

23. Структура содержит следующие поля: название принтера, фирма-производитель, цена картриджа, ресурс картриджа (в страницах). Написать функцию для нахождения принтера с самой низкой ценой отпечатков.

24. Структура содержит следующие поля: национальная валюта, столица, общее население. Написать функцию для нахождения государства с самым большим населением.

Содержание отчета:

1. Титульный лист
2. Цель работы
3. № варианта (тематика)
4. Задание полностью + задание по варинту
5. Текст программы
6. Блок-схема алгоритма
7. Выводы