Тема: «Пользовательские типы данных»

Вопросы:

1. Структуры. Алгоритмы работы со структурами.
2. Объединения. Алгоритмы работы с объединениями.
3. Перечисления. Алгоритмы работы с перечислениями.

Структуры

Структура – это набор взаимосвязанных данных, возможно и разных типов, объединенных в единое целое. Возможностью объединять данные разных типов, структура отличается от массива, а массив, как известно, объединяет данные только одного типа. Структуру целесообразно использовать в тех случаях, когда необходимо иметь одну переменную, содержащую набор взаимосвязанных данных. Например, для хранения списка сотрудников организации, удобно иметь одну переменную (Kadry), содержащую такие данные как фамилию сотрудника, его должность и отдел. При использовании структуры ее в начале необходимо объявить, а затем создать экземпляр этой структуры. Структура описывается с помощью ключевого слова struct :

```
struct Kadry { // описание структуры
int nzap; //номер записи

char fam[20]; //фамилия

char dol[10]; //должность

int otdel; // отдел
};
```

Идентификатор данной структуры Kadry определяет ее тег(имя). Элементы структуры называются членами-данными или полями. Каждый член структуры объявляется так же как обычная переменная. В приведенном примере структура содержит два массива типа char и два члена типа int. В описании структуры после закрывающейся фигурной скобки ставится точка с запятой. Когда структура описана, ее можно использовать, предварительно создав экземпляр структуры, которая выглядит следующим образом:

```
Kadry sotr ;
```

В результате для структуры выделяется необходимая память, и эта область памяти связывается с переменной, имеющей имя sotr. После чего можно присваивать значения членам-данным:

sotr. nzap = 1; strcpy(sotr.fam, “Сидоров”); strcpy(sotr.dol, “лаборант”); sotr.otdel = 12;

Для доступа к членам-данным используется оператор доступа к членам структуры, который представляет собой точку между именем переменной и именем члена структуры. Оператор доступа позволяет обращаться к конкретному члену структуры, как для чтения, так и для изменения его значения. При желании можно сразу инициализировать все члены вновь созданного экземпляра структуры:

Kadry engineer = { 2, “Петров”, “инженер” , 15

};

Этот способ короче предыдущего, но он не всегда применим в реальных ситуациях. Обычно структура заполняется в результате ввода данных пользователем или чтения их из файла. В этих случаях присвоение значений подобным образом невозможно.

Ниже показан пример ввода структуры Vkladchik пользователем. В этом примере форматированный вывод элементов структуры на дисплей осуществляется с помощью функции, прототип которой имеет вид void output (int, char*, float);.

//Пример ввода - вывода простейшей структуры на дисплей

#include <iomanip.h> #include <conio.h> #include <stdlib.h> #include <iostream.h> #include <fstream.h>

void output(int, char*, float); //прототип функции ввода struct Vkladchik

{
// определение структуры "Вкладчик"
int account; // номер счета

char name[10]; // имя

```
float suma; // сумма вклада
};
int main()
{
clrscr();
Vkladchik k ; // создание экземпляра объекта
cout<<"Введите счет, имя , сумму \n"; cin>>k.account>>k.name>>k.suma;
cout<<”Счет”<<setw(9 )<<”Имя” <<setw(16) <<”Сумма”<<endl;
output(k.account,k.name,k.suma);
cout<<"\n\n";
cout<<"\nНажмите любую клавишу ..."; getch();
return 0;
}
void output(int a, char* n, float s)
{
cout<<setiosflags(ios::left)<<setw(10)<< a<<setw(13)<<n
<<setw(7)<<setprecision(2)<<setiosflags( ios::showpoint|ios::right) << s<<endl;
}
```

Результаты работы программы:

Введите счет, имя, сумму

1
Victor
125.45

Счет	**Имя**	**Сумма**
1	**Victor**	**125.45**

Структуры, как и элементы других типов, могут объединяться в массивы структур. Чтобы объявить массив структур, надо сначала создать экземпляр структуры (например, struct Vkladchik), а затем объявить массив структур:

Vkladchik k[10]; .

Этот оператор создает в памяти 10 переменных типа структуры с шаблоном

Vkladchik и именами k[0], k[1] и т.д.

Ниже приведен пример программы, позволяющей вводить и выводить массив структур на дисплей.

```
//Пример ввода – вывода массива структур на дисплей

#include <iomanip.h>
#include <conio.h>
#include <stdlib.h>
#include <iostream.h>
#include <fstream.h>
void output(int , char* ,float ) ;
struct Vkladchik {

// определение структуры "Вкладчик"
int account;
// номер счета
char name[10]; // имя
float suma; // сумма вклада
};
int main() { int i, n; clrscr();
cout<<"Введите число записей n= "; cin>>n;
Vkladchik k[10]; // создание массива экземпляров объектов
cout<<"Введите счет, имя, сумму \n";
for(i=0;i<n; i++){ cin>>k[i].account>>k[i].name>>k[i].suma;
cout<<"?"<<endl; }
cout<<”Счет”<<setw(9)<<”Имя” <<setw(16) <<”Сумма”<<endl;
for(i=0;i<n; i++)

output(k[i].account, k[i].name, k[i].suma); cout<<"\n\n";

cout<<"\nНажмите любую клавишу ...";
getch(); return 0;

}

void output(int a, char* n, float s)

{

cout<<setiosflags( ios::left)<<setw(10)<< a<<setw(13)<<n
<<setw(7)<<setprecision(2)<<setiosflags( ios::showpoint|ios::right) << s<<endl;

}
```

Запустите программу самостоятельно и проанализируйте полученный результат.

//Пример ввода - вывода структуры в файл

#include <iomanip.h> #include <conio.h> #include <stdlib.h> #include <iostream.h> #include <fstream.h>

void output(int, char*, char*, int); // прототип функции вывода struct book

{// определение структуры

int nzap; //номер записи

char fam[20]; //фамилия

char dol[10];//должность

int otdel;// отдел

};

int main()

{

clrscr();

book b; // создание экземпляра объекта

ofstream outfile("kniga"); //открытие файла для записи if(!outfile)

{cerr<<"файл не открыть"; exit(1);

}

cout<<"Введите номер записи, фамилию, должность, номер отдела" <<" и символ eof по окончании ввода\n";

while(cin>>b.nzap>>b.fam>>b.dol>>b.otdel)

{

outfile<<b.nzap<<" "<<b.fam<<" "<<b.dol<<" "<<b.otdel<<endl; cout<<" ? ";

}

outfile.close();

ifstream infile("kniga"); // открывает файл для чтения if(!infile)

{cerr<<"файл не открыть"; exit(1);

}

cout<<"Это содержание файла:\n"; cout<<setw(10)<<"номер записи” <<setw(10)<<"фамилия"<<setw(15)<<”должность" <<setw(13)<<"отдел"<<endl; while(infile>>b.nzap>>b.fam>>b.dol>>b.otdel) output(b.nzap, b.fam, b.dol, b.otdel); // вызов функции вывода infile.close();

cout<<"\n\n";

```
cout<<"\nНажмите любую клавишу ..."; getch();
return 0;
}
void output(int z, char* a, char* n, int s) // описание функции вывода
{
cout<<setiosflags( ios::left)<<setw(15)<< z<<setw(13)<< a<<setw(13)
<<n<<setw(8)<<setiosflags( ios::right)
<< s<<endl; }
```

//Ввод и вывод структуры в файл и на дисплей в режиме диалога

```
#include <fstream.h>
#include <string.h>
#include <conio.h>
#include <iomanip.h>
 #define FAM 25
#define DOL 15
struct SOTR
// объявление структуры :”Сотрудники”
{
char fam [FAM]; // фамилия
char dol[DOL]; // должность
int otdel;// отдел
} ;
void sozdanie(); //прототип функции :” Создание”
void prosmotr(); // прототип функции :” Просмотр”
// операция-функция ввода в структуру с клавиатуры istream &operator
>> (istream &in, SOTR &x)

{

cout<<"\nФамилия:"; in.seekg(0,ios::end); in.get(x.fam,FAM-1,'\n');
cout<<"\nДолжность:"; in.seekg(0,ios::end); in.get(x.dol,DOL-1,'\n');
cout<<"\nОтдел:";
in.seekg(0,ios::end); in >> x.otdel; return in;
}
// операция-функция вывода структуры на дисплей ostream &operator <<
(ostream &out, SOTR x)
// печать объекта
{
```

```
out << "\n|" << x.fam << "|" << x.dol << "|" << x.otdel <<"|"; return out;
}
// операция-функция ввода структуры с МД ifstream &operator >> (ifstream &in, SOTR &x)
{
in.setf(ios::left);
in.width(FAM);
in.get(x.fam,FAM,'\n');
in.width(DOL);
in.get(x.dol,DOL,'\n'); in >> x.otdel;
return in;
}
// операция-функция вывода структуры на МД ofstream &operator << (ofstream &out, SOTR &x)
{
out.width(FAM-1); out.setf(ios::left); out << x.fam; out.width(DOL-1); out.setf(ios::left); out << x.dol;
out << x.otdel; return out;
}
void main(void)
{
clrscr(); char c; while (1)
{
cout << endl << "1. Создание файла";
cout << endl << "2. Просмотр содержимого"; cout << endl << "3. Выход";
cout << endl << "Ваш выбор -> "; cin.seekg(0,ios::end);
c = cin.get();
switch(c)
{
case '1': sozdanie(); break; case '2': prosmotr(); break; case '3': return; default:
cout << "Вводите только цифры от 1 до 3" << endl;
}
}
}
void sozdanie()
{
char c;
//поток ff для вывода файла kniga.txt ofstream ff;
//создание экземпляра объекта
SOTR s;
```

```
ff.open("kadry.txt", ios::binary );
//цикл записи элементов в файл
do
{
cin >> s; // ввод с клавиатуры ff << s; // вывод в файл
cout<<"\nПродолжить ввод?(Y/N или Д/Н)";
}
while ((c = getch())=='y'||c=='Y'||c=='д'||c=='Д'); ff.close(); // закрытие файла
}
void prosmotr()
{
ifstream finp; SOTR s;
// поток finp для ввода из файла kniga.txt finp.open("kadry.txt", ios::binary); finp.seekg(0,ios::beg);
cout<<"\nСписок элементов из файла\n"; while ( finp ) // пока не конец файла
{
finp >> s ; // вывод из файла cout << s; // вывод на дисплей
}
finp.close(); // закрытие файла
}
```

Объединения

Объединение – это группирование переменных, которые разделяют одну и ту же область памяти. В зависимости от интерпретации осуществляется обращение к той или другой переменной объединения. Все переменные, что включены в объединение начинаются с одной границы.

Объединение позволяет представить в компактном виде данные, которые могут изменяться. Одни и те же данные могут быть представлены разными способами с помощью объединений.

Объявление объединения похоже на объявление структуры:

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

Как и для структур, можно объявить переменную, поместив ее имя в конце определения или используя отдельный оператор объявления. Для объявления переменной cnvt объединения union_type следует написать:

union union_type cnvt;

Когда объявлено объединение, компилятор автоматически создает переменную достаточного размера для хранения наибольшей переменной, присутствующей в объединении.

Для доступа к членам объединения используется синтаксис, применяемый для доступа к структурам - с помощью операторов «точка» и «стрелка». Чтобы работать с объединением напрямую, надо использовать оператор «точка». Если к переменной объединения обращение происходит с помощью указателя, надо использовать оператор «стрелка». Например, для присваивания целого числа 10 элементу i объединения cnvt следует написать:

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

Использование объединений помогает создавать машинно-независимый (переносимый) код. Поскольку компилятор отслеживает настоящие размеры переменных, образующих объединение, уменьшается зависимость от компьютера. Не нужно беспокоиться о размере целых или вещественных чисел, символов или чего-либо еще.

Объединения часто используются при необходимости преобразования типов, поскольку можно обращаться к данным, хранящимся в объединении, совершенно различными способами. Рассмотрим проблему записи целого числа в файл. В то время как можно писать любой тип данных (включая целый) в файл с помощью fwrite(), для данной операции использование fwrite() слишком «жирно». Используя объединения, можно легко создать функцию, побайтно записывающую двоичное представление целого в файл. Хотя существует несколько способов создания данной функции, имеется один способ выполнения этого с помощью объединения. В данном примере предполагается использование 16-битных целых. Объединение состоит из одного целого и двухбайтного массива символов:

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

Объединение позволяет осуществить доступ к двум байтам, образующим целое, как к отдельным символам. Теперь можно использовать pw для создания функции write_int(), показанной в следующей программе:

```
#include <stdio.h>
#include <stdlib.h>
union pw {
int i;
char ch[2];
};

int write_int(int num, FILE *fp);

int main()
{
FILE *fp;
fp = fopen("test.tmp", "w+");

if(fp==NULL) {
printf("Cannot open file. \n");
exit(1);
}

write_int(1000, fp);
fclose(fp);
return 0;
}

/* вывод целого с помощью объединения */

int write_int (int num, FILE *fp) {
union pw wrd;
wrd.i = num;
putc(wrd.ch[0], fp); /* вывод первой половины */
return putc(wrd.ch[1], fp); /* вывод второй половины */
}
```

Хотя write_int() вызывается с целым, она использует объединение для записи обеих половинок целого в дисковый файл побайтно.

Перечисления

Перечисления - это набор именованных целочисленных констант, определяющий все допустимые значения, которые может принимать переменная. Перечисления можно встретить в повседневной жизни.

Например, в качестве перечислений монет в Соединенных Штатах используются:

один цент, пять центов, десять центов, двадцать пять центов, полдоллара, доллар

Перечисления определяются с помощью ключевого слова enum, которое указывает на начало перечисляемого типа. Стандартный вид перечислений следующий:

enum ярлык { список перечислений} список переменных;

Как имя перечисления - ярлык, так и список переменных необязательны, но один из них должен присутствовать. Список перечислений - это разделенный запятыми список идентификаторов. Как и в структурах, ярлык используется для объявления переменных данного типа. Следующий фрагмент определяет перечисление coin и объявляет переменную money этого типа:

```
enum coin { penny, nickel, dime, quarter, half_dollar, dollar };
enum coin money;
```

Имея данное определение и объявление, следующий тип присваивания совершенно корректен:

```
money = dime;
if (money==quarter) printf("is a quarter\n");
```

Важно понять, что в перечислениях каждому символу ставится в соответствие целочисленное значение и поэтому перечисления могут использоваться в любых целочисленных выражениях. Например:

```
printf("The value of quarter is %d ", quarter);
```

совершенно корректно.

Если явно не проводить инициализацию, значение первого символа перечисления будет 0, второго - 1 и так далее. Следовательно:

```
printf("%d %d", penny, dime);
```

выводит 0 2 на экран.

Можно определить значения одного или нескольких символов, используя инициализатор. Это делается путем помещения за символом знака равенства и целочисленного значения. При использовании инициализатора, символы, следующие за инициализационным значением, получают значение больше чем указанное перед этим. Например, в следующем объявлении quarter получает значение 100.

```
enum coin ( penny, nickel, dime, quarter=100, half_dollar, dollar);
```

Теперь символы получат следующие значения:

```
penny    0
nickel    1
dime    2
quarter    100
Half dollar    101
dollar    102
```

Используя инициализацию, более одного элемента перечисления могут иметь одно и тоже значение.

Заблуждением считается возможность прямого ввода или вывода символов перечислений. Следующий фрагмент кода не работает так, как нужно:

```
/* Не работает. */
money = dollar;
printf("%s", money);
```

Надо помнить, что символ dollar - это просто имя для целого числа, а не строка Следовательно, невозможно с помощью printf() вывести строку «dollar», используя значение в money. Аналогично нельзя сообщить значение переменной перечисления, используя строковый эквивалент. Таким образом, следующий код не работает:

```
/* этот код не будет работать */
money = "penny";
```

На самом деле, создание кода для ввода и вывода символов перечислений - это довольно скучное занятие. Например, следующий код необходим для вывода состояния money с помощью слов:

```
switch(money) {
case penny: printf("penny");
break;
case nickel: printf("nickel");
break;
case dime: printf("dime");
break;
case quarter: printf("quarter");
break;
case half_dollar: printf("half_dollar");
break;
case dollar: printf("dollar");
}
```

Иногда возможно объявить массив строк и использовать значения перечислений как индекс для их перевода в соответствующие строки. Например, следующий код также выводит соответствующую строку:

```
char name[][12]={
"penny",
"nickel",
"dime",
"quarter",
"half_dollar",
"dollar"
};
...
printf("%s", name[money] );
```

Конечно, это работает только в том случае, если не используется инициализатор, поскольку массив строк должен индексироваться начиная с 0.

Поскольку значения перечислений должны преобразовываться вручную к строкам, которые могут читать люди, они наиболее полезны в подпрограммах, не выполняющих такие преобразования. Например, перечисления, как правило, используются для определения символьном таблицы компилятора.