

**Методические указания к лабораторной работе № 6 по курсу
ОСНОВЫ ПРОГРАММИРОВАНИЯ
ГУИМЦ**

**" Структуры данных в СИ "
(6 часов)**

СОДЕРЖАНИЕ

СОДЕРЖАНИЕ	2
1. 1. Цель лабораторной работы № 6 по дисциплине ОП (Основы программирования)	4
2. 2. Порядок выполнения лабораторной работы	4
3. 3. Основные понятия	4
3.1. 3.1 Проблемы хранения и обработки данных	4
3.2. 3.2 Структура - элемент хранения разнородных данных - struct	5
3.3. 3.3 Инициализация структурных переменных	6
3.4. 3.4 Работа с полями структуры через структуру и через указатель на структуру	7
3.5. 3.5 Многоуровневая квалификация полей в структурах	8
3.6. 3.6 Указатели на структуры и на динамические структуры	9
3.7. 3.7 Структуры параметры и типы возврата функций	9
3.8. 3.8 Передача структур в функцию	10
3.9. 3.9 Передача указателя на структуру в функцию и возврат структур	11
3.10. 3.10 Присваивание структур	12
3.11. 3.11 Массивы структур	12
3.12. 3.12 Массивы структур в функциях	13
3.13. 3.13 Вложенные структуры	13
3.14. 3.14 Размер и размещение структур в ОП	14
3.15. 3.15 Динамической структуры	15
3.16. 3.16 Создание и удаление динамической структуры со строками	16
3.17. 3.17 Структуры с указателями на собственный тип	17
3.18. 3.18 Перечисления - enum	17
3.19. 3.19 Указатели на поля структуры	18
3.20. 3.20 Союзы – union - объединения	19
4. 4. Примеры программы с использованием структур	19
4.1. 4.1 Примеры, описанные в теоретической части ЛР	20
4.2. 4.2 Копирование и обмен статических структур	20
4.3. 4.3 Функция Swar для структур с динамическими строками	22
4.4. 4.4 Заполнение случайных - rand числовых полей массива структур	24
4.5. 4.5 Сортировка массива структур с помощью функции Swar	25
5. 5. Контрольные задание ЛР №6.	26
5.1. 5.1 Создание консольного проекта ЛР	26
5.2. 5.2 Описать структуру своего варианта ДЗ и работа с ней	27
5.3. 5.3 Функция Распечатки структуры	29
5.4. 5.4 Функция Распечатки структуры через указатель	29
5.5. 5.5 Массив структур, его инициализация и функция распечатки	30
5.6. 5.6 Функция копирования структур	31
5.7. 5.7 Функция обмена (Swar) для структур	32
5.8. 5.8 Использование перечислений по варианту	33
5.9. 5.9 Заполнение числовых полей массива структур случайными данными	33
6. 6. Варианты заданий для студентов СУЦ.	34
7. 7. Дополнительные требования для студентов СУЦ (д.т.).	35
7.1. 7.1 Д.Т. Сортировка массива по целому полю	36
7.2. 7.2 Д.Т. Двумерный массив структур, инициализация и распечатка	36
7.3. 7.3 Д.Т. Заполнение данных случайными числами в массиве структур	36
7.4. 7.4 Д.Т. Поиск экстремума в массиве структур, его номера и распечатка	36
7.5. 7.5 Д.Т. Сортировка массива структур своего варианта ДЗ по символьному параметру	36
7.6. 7.6 Д.Т. Сортировка массива указателей на структуры	36

ОП ГУИМЦ 2024 ЛР№6

7.7. 7.8 Д.Т. Функция для создания и заполнения динамического массива структур.....	36
7.8. Д.Т. Функция для корректного удаления динамического массива структур ДЗ	36
7.9. 7.9 Д.Т. Swap для структуры с динамическими строками	36
7.10. 7.10 Д.Т. Функция ввода структуры	37
7.11. 7.11 Д.Т. Функция изменения структуры	37
7.12. 7.12 Д.Т. Сортировка массива структур	37
8. 8. Демонстрация, защита ЛР и отчет по ЛР.	37
9. 9. Контрольные вопросы по ЛР.	37
10. 10. Литература.	38
Новые темы для структур.....	39

ОП ГУИМЦ 2024 ЛРН№6

1. 1. Цель лабораторной работы № 6 по дисциплине ОП (Основы программирования)

Целью данной ЛР по дисциплине ОП является получение навыков работы с переменными типа структура на языке программирования СИ. Студенты используют консольные проекты и отлаживают программы в среде программирования MS VS 2005/2008/2010. Студенты знакомятся с основными операциями при работе со структурами, способами их инициализации, созданием массивов структур их заполнением, их сортировкой, проверяют работу отлаженных примеров и делают контрольные задания. Они выполняют отладку программы по своему варианту и получают исполнимую программу, готовую к выполнению, оформляют отчет по ЛР и защищают его.

В данной ЛР студенты проектируют структуру, которая в дальнейшем будет использоваться в последующих ЛР и Домашнем задании по курсу "Основы программирования".

2. 2. Порядок выполнения лабораторной работы

1. Познакомиться с содержанием методических указаний и основными понятиями ЛР (разделы 3 и 4)
2. Проработать порядок выполнения работы (раздел 5).
3. Создать консольные проекты для проверки примеров и выполнения задания ЛР(разделы 3,4).
4. Проверить в данном проекте примеры из методических указаний, выполнив их в отладчике в пошаговом режиме (раздел 4).
5. Написать программу заданий ЛР по варианту, выданному преподавателем и отладить ее (раздел 5).
6. **Продемонстрировать работу программы преподавателю в режиме отладчика по шагам и изменяемыми переменными.**
7. Подготовить отчет по ЛР по представленному шаблону (раздел 8).
8. Защитить ЛР с предоставлением отчета и ответами на контрольные вопросы (раздел 8,9).
9. Для продвинутых студентов выполнить задания для дополнительных (необязательных) требований и также отобразить их в отчете по ЛР (раздел 7).

3. 3. Основные понятия

В теоретической части описания лабораторной работы вводятся основные понятия и рассматриваются принципы для работы со структурами на языке программирования СИ.

3.1. 3.1 Проблемы хранения и обработки данных

Самыми важными функциями современных компьютеров являются функции хранения и обработки информации. Количество хранимой в компьютерах информации возрастает многократно, поэтому то, как эти данные хранятся, существенно влияет на эффективность процессов использования этой информации.

Одним из решений наглядного и обозримого способа хранения информации, является ее структуризация и абстрагирование. Структуризация подразумевает, что данные, связанные между собой хранятся вместе, а абстрагирование позволяет отвлечься от детализации информации на определенных этапах ее обработки.

Массивы информации могли бы быть таким хранилищем связанной информации, если бы не требования ее однородности (Напомним, массив – множество однотипных переменных). В реальных задачах требования однотипности не соблюдаются. Так для описания человека, как

минимум, нужны и символьные данные (ФИО) и числовые данные (даты и время) и вещественные данные (зарплата, стипендия). Для реализации технологии совместного хранения разнотипных данных в языки программирования добавлено понятие структура данных. Технология структур данных позволяет в программах реализовать более высокий уровень абстракции данных и сделать программы более наглядными.

3.2. 3.2 Структура - элемент хранения разнородных данных - struct

Массивы объединяют группы переменных одного типа. В программах часто требуется группировать вместе разнотипные переменные. Для этого предусмотрены специальные описания – структуры данных (struct). Структура данных – это совокупность разнородных данных для описания отдельного информационного множества данных. Фактически структуры описывают новые типы переменных (подобно классам, но об этом в других ЛР). Нужно различать понятия:

- Описание шаблона структуры.
- Описание переменных структурного типа.
- Использование переменных структурного типа.

Описание шаблона структуры включает описание отдельных переменных (часто их называют полями структуры) с указанием их типов и заданием уникальных имен в пределах одной структуры. Формализовано это выглядит так:

```
struct <имя структуры> {
    <описание поля 1 структуры>;
    ...
    <описание поля N структуры>;
};
```

Пример описания структуры:

```
// Описание структуры Student
struct Student {
    char Name[14]; // Фамилия студента
    int kurs; // Курс обучения
    Student * pStud; // Указатель на другого студента
    bool pol; // Пол студента: true - women, false - men
    float Stipen; // Размер стипендии
}; //
```

Допустимо описание шаблона структуры с одновременным описанием структурных переменных этого типа:

```
struct <имя структуры> {
    <описание поля 1 структуры>;
    ...
    <описание поля N структуры>;
} [<список описаний конкретных структурных переменных>;];
```

Пример одновременного описания шаблона структуры и структурных переменных:

```
// Описание структурных переменных типа Student с одновременным описанием структурной
// переменной этого типа
// (Student1) и массива этого типа (Group31[30])
// Описание структуры Student
struct Student {
    char Name[14]; // Фамилия студента
    int kurs; // Курс обучения
    Student * pStud; // Указатель на другого студента
```

ОП ГУИМЦ 2024 ЛР№6

```
bool pol; // Пол студента: true - women, false - men
float Stipen; // Размер стипендии
}; //
} Student1, Group31[30]; // Необязательное описание переменных и массивов этой
структуры
```

Допустимо отдельное описание структурных переменных, после того как описан шаблон структуры. Это выглядит так:

```
struct Student Student1, Group31[30];
```

Или так, таким образом, без ключевого слова **struct**, но только с названием новой структуры (кстати, ранее не допускалось):

```
Student Student1, Group31[30];
```

Ставить точку с запятой после описания структуры нужно обязательно. Другие примеры описания структур даны ниже. Их назначение понятно из комментариев, которые даны в тексте программы:

```
struct Complex { // Структура - комплексная переменная
double re; // действительная часть
double im; // мнимая часть
};
//
struct Date { // Структура - дата
int day; // День
int month; // месяц
int year; // Год
};
//
struct Person { // Структура - Персона
char name[50];
Date birthdate; // структурная переменная дата рождения
double salary; // Оклад
};
```

Правила описания отдельных полей структуры совпадают с правилами описания обычных переменных. Поля могут быть любого типа, в том числе и также структурными переменными (за исключением типа самой описываемой структуры). Однако допустимо объявлять указатели на данный структурный тип в виде отдельного поля структурной переменной. Описание структурных переменных производится с указанием имени структуры как нового типа данных и может быть выполнено в программе многократно. Шаблон структуры должен быть доступен перед первым его использованием для описания переменных. Пример отдельного описания структурных переменных и их массивов:

```
struct Student S333; // Описание без инициализации полей
Student S1 = {"Петров", 1, false, 1500.0f}; // Описание с инициализацией полей
Student Group[30]; // Описание массива структур
Complex z;
Date d;
Person p;
```

3.3. 3.3 Инициализация структурных переменных

Инициализация структурных переменных (начальные значения полей структуры), как и для массивов, может быть выполнена при описании конкретной структурной переменной.

Строковые значения указываются при этом в кавычках, а инициализация массивов в фигурных скобках. Описание структурных переменных с их инициализацией:

```
Complex z1 = { 1.6, 0.5 };
Date d1 = { 1, 4, 2001 };
Person p1 = { "Сидоров", {10, 3, 1978}, 1500.48 };
// Описание структуры лицо
struct Face {
int MasF [5]; // М массив целых
Date birthdate; // Структура даты
char name[50]; // Строка
};
//Описание структурной переменной с инициализацией массива, строки и другой
структуры
Face f = { {1,2,3,4,5} , { 1, 9, 2014 } , "Представительное лицо"};
//
```

В последнем случае мы имеем пример инициализации вложенной структурной переменной: в структуре **Face** объявлена структура **Date**. Для ее инициализации добавляются фигурные скобки.

Допустимо для инициализации использовать переменные, но их значения к моменту описания новой структуры должны быть вычислены:

```
int test = 10; // Простая переменная
Person STest= {"Большаков" , test , 60000.0f} ; // Переменная задана при
инициализации
```

Допустимо для инициализации использовать и структурные переменные если они предварительно проинициализированы:

```
Person S31;
Person STest3 =STest; // Правильно
Person STest4 =S31; // Ошибка - S31 не инициализирована
```

3.4. 3.4 Работа с полями структуры через структуру и через указатель на структуру

При описании структурных переменных ключевое слово **struct** в новой нотации перед описанием C++ необязательно, поэтому мы не будем его указывать. Для работы с переменными такого типа недостаточно указывать только имя структурной переменной, нужно указать также и имя конкретного поля. Такой элемент программы называется квалифицированной ссылкой (имя структуры и поля разделяет точка – “.”). Такая квалифицированная ссылка рассматривается системой программирования как обычная переменная. Ее можно использовать в любых операторах и выражениях программы, без каких либо ограничений. Единственным требованием в этом случае – необходимость предварительного описания или обеспечения доступа к этой структурной переменной на момент ее использования.

Формализовано ссылка на поле структурной переменной выглядит так:

<имя структурной переменной>.<имя поля структуры>

Например, для структурных переменных:

```
struct Student S333; // Описание без инициализации полей
Student S1 = {"Петров" , 1 , false , 1500.0f } ; // Описание с инициализацией полей
Student Group5[30]; // Описание массива структур
```

Примеры использования конкретных полей (**kurs**) конкретных структурных переменных (**S1, Group5[]**) в операторах присваивания:

```
S1.kurs = 2; // Для поля kurs структуры S1
Group5 [4].kurs = 3; // Для поля kurs 5-го элемента массива структур Group5
```

ОП ГУИМЦ 2024 ЛРН№6

Можно использовать указатели на структурную переменную:

```
Student * pStud = &S1; // Описание указателя и его инициализация
```

В этом случае доступ к полю структуры задается не точкой, а специальной операцией из двух символов (“->” - похоже на сам указатель):

```
pStud->kurs = 5 // обращение к полю структурной переменной
```

Для работы в функциях со структурами в качестве параметра передается указатель на нее, что позволяет существенно сократить список передаваемых параметров.

3.5. 3.5 Многоуровневая квалификация полей в структурах

Для вложенных структур (в одной структуре используется другая и т.д.) применяется много уровневая квалификация. Это необходимо при использовании других структурных переменных в качестве полей основной структуры (в **Face** описано поле типа **Date** – см. ниже). Доступ к полю в этом случае выполняется через двойную квалификацию:

```
struct Date { // Структура даты
int day;      // День
int month;    // Месяц
int year;     // Год
};
// Описание структуры лицо
struct Face {
int MasF [5]; // М массив целых
Date birthdate; // Структура типа даты
char name[50]; // Строка
} ;
...
// Описание структурной переменной
Face Face2;
...
// Использование поля и вложенного поля day
Face2.birthdate.day = 30; // Двойная квалификация
...
```

В программах не исключается возможность вложений структурных переменных с большим числом уровней. Например (**Person - Student - Date**):

```
struct Date { // дата
int day; // День
...
};
//
struct Student{
...
Date birthdate; // Подструктура типа даты
...
};
//
struct Person{
...
Student Stud; // Подструктура типа даты
...
};
// Описание структурной переменной
Person Person2;
...
Person2.Stud.birthdate.day = 30; // Тройная квалификация
```


3.6. 3.6 Указатели на структуры и на динамические структуры

Мы уже говорили, что на структуру можно задать указатель. Пусть есть такое описание указателя:

```

struct Date { // Структура даты
int day;      // День
int month;    // Месяц
int year;     // Год
};

struct Person { // Персона
char name[50];
Date birthdate; // структурная переменная дата рожденич
double salary; // Оклад
};

...
// Описание структурной переменной и ее инициализация
Person p2 = { "Сидоров", {10, 3, 1978}, 1500.48 };

...
// Описание указателя и его инициализация
Person * pPerson = &p2;

...
// Изменить оклад
p2.salary = 100.50; // Без указателя
pPerson -> salary = 10.50; // С указателем
(*pPerson).salary = 12.50; // Можно и так (разименование указателя), скобки
обязательны

```

Если память под структуру выделена динамически (функцией **malloc** или операцией **new**), то также должен использоваться указатель на структуру (**pPerson**):

```

...
// Выделить динамическую память
pPerson = (Person *) malloc (sizeof(Person));
pPerson -> salary = 5.5;
strcpy(pPerson -> name, "Петров"); // Копирование строк
pPerson -> birthdate.year = 1995; // Комбинированная квалификация (и стрелка и
точка)
// Освободить динамическую память
free(pPerson);
...

```

Если в структурах описаны указатели на структуры (Например, **pStudent**), то квалификация указателями выполняется двойными стрелками (здесь внутри одной структуры задан указатель на вложенную структуру **Student**):

```
ptrPerson -> pStudent -> salary = 15.00;
```

или такого вида (где имеет место двойная квалифицированная ссылка):

```
p2.birthdate. year = 2014;
```

Напомню, что в структуре допускается использовать указатель на “себя” – структуру такого же типа, это часто применяется для создания списковых структур данных.

3.7. 3.7 Структуры параметры и типы возврата функций

При работе с функциями возможны случаи (прототипы):

- Структура передается в функцию

ОП ГУИМЦ 2024 ЛР№6

```
void PrintKurs (Student s);
```

- Функция возвращает структуру

```
Student Kurs10 (Student s);
```

- Указатель на структуру передается в функцию

```
void ChangeStudKurs(Student * pS, int NewKurs);
```

- Функция возвращает указатель на структуру (статика). В функцию передается указатель на существующую структуру.

```
Student * StudPerevod( Student * pS );
```

- Функция возвращает указатель на структуру (динамика). В этом случае память выделяется в функции, а затем должна быть возвращена (free).

```
Student * StudNew(void);
```

Примеры применения функций со структурами:

```
// структуры параметры и типы возврата
// Структура в виде параметра (без указателя)
S1.kurs = 2;
PrintKurs(S1);
printf("Печать поля вне функции S1.kurs = %d\n", S1.kurs);
Kurs10(S1);
printf("Печать поля вне функции S1.kurs = %d\n", S1.kurs);
Student S0;
S0 = Kurs10(S1); // Присваивание структур!!! Kurs10 - возвращает структуру
printf("Печать поля вне функции S0.kurs = %d\n", S0.kurs);
Student S5 = { "Иванов", 2, &S0, 2000.0f };
// Параметр указатель на структуру
printf("Печать до функции ChangeStudKurs S.kurs = %d\n", S5.kurs);
ChangeStudKurs(&S5, 1);
printf("Печать после функции ChangeStudKurs S.kurs = %d\n", S5.kurs);
// Возврат указатель на структуру
//
printf("Печать до функции StudPerevod S.kurs = %d\n", S5.kurs);
// Вызов функции возвращающей структуру
printf("Печать после функции StudPerevod S.kurs = %d\n", (StudPerevod (&S5))->kurs);
//
Student * pStud;
printf("Печать после функции StudNew S.kurs = %d\n", (pStud= StudNew())->kurs);
free (pStud);
```

Получим результат:

```
Печать поля в функции s.kurs = 2
Печать поля вне функции S1.kurs = 2
Печать поля вне функции S1.kurs = 2
Печать поля вне функции S0.kurs = 10
Печать до функции ChangeStudKurs S.kurs = 2
Печать после функции ChangeStudKurs S.kurs = 1
Печать до функции StudPerevod S.kurs = 1
Печать после функции StudPerevod S.kurs = 2
Печать после функции StudNew S.kurs = 6
Для продолжения нажмите любую клавишу . . .
```

3.8. 3.8 Передача структур в функцию

Передать структуру в функцию, в качестве фактического параметра можно двумя способами: по значению, тогда изменить структуру в функции нельзя, или передать указатель на структуру. Рассмотрим здесь первый случай. Пусть разработана функция для печати отдельного поля структуры (**name**) и изменения другого поля (**salary**):

```

// Описание структуры
struct Person { // Структура - Персона
char name[50];
Date birthdate; // структурная переменная дата рожденич
double salary; // Оклад
};
// Описание функции печати поля структуры (name)
void PrintPersonName( Person per)
{
printf ("Печать в функции per.name = %s\n" , per.name);
per.salary = 5.0; // зменение поля структуры
// Для проверки изменения поля структуры в функции
printf( "Печать внутри функции per.salary = %f \n" ,per .salary );
};
...

```

Зададим прототип функции печати поля структуры в главной программе

```
void PrintPersonName( Person per);
```

В главной программе зададим описание структуры и выполним вызов функции:

```

// Описание структуры
Person p2 = { "Сидоров", {10, 3, 1978}, 15.00 };
//Вызов функции параметром структура
printf( "До функции p2.salary = %f \n" ,p2 .salary );
PrintPersonName( p2 ); // p2 – имя структуры в функцию передача по значению
printf( "После функции p2.salary = %f \n" ,p2 .salary ); //Поле в функции не изменилось

```

Получим результат:

```

До функции p2.salary = 15.000000
Печать в функции per.name = Сидоров
Печать внутри функции per.salary = 10.000000
После функции p2.salary = 15.000000

```

Значение полей структуры (в частности **p2.salary**) не изменяются при передаче всей структуры. Подумайте почему.

3.9. 3.9 Передача указателя на структуру в функцию и возврат структур

При передаче в функцию всей структуры в качестве параметра, значения полей изменить нельзя, так как передача в СИ выполняется по значению (через стек передается копия структуры). Если мы хотим изменять значения полей в структурной переменной внутри функции, то в нее необходимо передать указатель на эту структуру. Функция передачи указателя на структуру, имеет вид (опишем ее для разнообразия в другом файле проекта - **second.cpp**):

```

//
void ChangePersonSalary( Person * p , double newSalary)
{
p -> salary = newSalary; // Изменение поля по указателю на структуру p поля salary
};
...

```

Прототип этой функции нужно задать в начале главной программы:

```

void ChangePersonSalary ( Person per);
...
// Описание и инициализация структурной переменной Person
Person p2 = { "Сидоров", {10, 3, 1978}, 15.00 };
...
// Печать поля до вызова функции изменения поля
printf( "До функции p2.salary = %f \n" ,p2 .salary );

```

ОП ГУИМЦ 2024 ЛРН№6

```
// Передача указателя в функцию
ChangePersonSalary( &p2 , 30.0); // Передача указателя = &p2
// Печать поля после вызова функции изменения поля
printf( "После функции ChangePersonSalary p2.salary = %f \n" ,p2 .salary );
```

Получим результат:

До функции p2.salary = 15.000000

После функции ChangePersonSalary p2 .salary = 30.000000

Функция может иметь тип возврата структуры или указателя на структуру. Тип возврата структура:

3.10. 3.10 Присваивание структур

Допустимо для структур без динамических строк и без указателей такое выражение присваивания:

```
printf("До COPY S1=S3:\n");
StudentPrint(S1);
StudentPrint(S3);
S1 = S3;
printf("ПОСЛЕ COPY S1=S3:\n");
StudentPrint(S1);
StudentPrint(S3);
```

Результат работы фрагмента:

До COPY S1=S3:

Структура Student в функции:

Фамилия - Петров

Имя - Иван

Курс - 1

Стипендия - 2000.00 р.

Структура Student в функции:

Фамилия -

Имя -

Курс - 4

Стипендия - 20.00 р.

ПОСЛЕ COPY S1=S3:

Структура Student в функции:

Фамилия -

Имя -

Курс - 4

Стипендия - 20.00 р.

Структура Student в функции:

Фамилия -

Имя -

Курс - 4

Стипендия - 20.00 р.

3.11. 3.11 Массивы структур

Так как структура определяет новый тип переменной в СИ, то разрешается описывать массивы структур этих переменных. Например, ниже дано описание шаблона структуры и массива таких структур:

```
// Описание шаблона структуры типа Prepod
struct Prepod {
...
float Oklad; // Оклад
...
}
```

```

};
// Описание массива структур типа Prepod
Prepod KafIU[30];
// Работа с элементами массива структур (индексными переменными)
KafIU[0].Oklad = 10.0;
KafIU[10].Oklad = 10.0;
// Пример цикла занесения оклада для всех структурных переменных массива
for (int i = 0 ; i < 30 ; i++)
{
    KafIU[i].Oklad = 10.0;
};

```

С массивом, естественно, можно работать и через указатель (**ptrMas**), при этом инициализация указателя должна быть проведена с начальным адресом массива структур (**&KafIU[0]**) или просто именем этого массива (**KafIU**), так как само имя массива задает адрес начала массива (указатель на массив структур). Примеры:

```

// или с указателем
Prepod *ptrMas = &KafIU[0];
// или
Prepod *ptrMas = KafIU;

```

Не трудно проверить, что значения вычисленных указателей **ptrMas** в первом и втором случае одинаково.

Допустимы разные способы доступа через указатели с индексацией в массиве структур (Например, так **ptrMas[2].Oklad**). Или с вычислением нового адресного выражения для указателя (**ptrMas + 2**), причем плюс 2 на самом деле эквивалентно прибавлению двойного размера одиночной структуры, то есть величины равной - `sizeof(Prepod)`. Можно также выполнить разименование указателя структуры (*) и работать с ней как с обычной структурной переменной. Посмотрите и проверьте примеры, приведенные ниже:

```

ptrMas->Oklad = 45.0;           // Использование указателя для структуры
ptrMas[2].Oklad = 15.0;        // ИНДЕКСАЦИЯ: для второго элемента массива структур
(*ptrMas).Oklad = 55.0;        // РАЗИМЕНОВАНИЕ указателя
(ptrMas + 2)->Oklad = 25.0;    // СЛОЖЕНИЕ УКАЗАТЕЛЕЙ И КОНСТАНТ
(*(ptrMas + 2)).Oklad = 35.0;  // можно и так – разименование адресного выражения с указателем
ptrMas = ptrMas + 2;          // Явное изменение указателя (фактически плюс sizeof(Prepod))
ptrMas->Oklad = 45.0;          // Использование вычисленного указателя для массива структур

```

Во всех рассмотренных случаях будет изменена одно и тоже поле элемента структуры с номером **2** в массиве структур типа **Prepod**.

3.12. 3.12 Массивы структур в функциях

В функцию передается указатель на массив и его размер (**pP** и **Razm**).

```

// Указатель на массив структур
float SumSalaryMas (Person * pP, int Razm) // сумма полей salary массива
{
    float SumSal = 0.0f;
    for (int i = 0 ; i < Razm ; i++)
        SumSal += (pP + i)->salary ;
    return SumSal;
};

```

3.13. 3.13 Вложенные структуры

Вложенными структурами называются такие структуры, в которых в качестве поля описана другая структурная переменная или указатель на структурную переменную. Причем поле структуры того же типа описывать запрещено, но описание указателя на саму себя допускается.

Примеры вложенных структур, где внутри одной структуры описана другая структура (тип **Prepod** - **DecPrep**) и указатель на другую структуру (тип **Person** - **pFace**) приведены ниже.

```

struct Date { // Структура Date
int day;      // День
int month;    // Месяц
int year;     // Год
};
// Структура Person с вложенной структурой Date
struct Person {
char name[50];
Date birthdate; // структурная переменная дата рождения типа Date
double salary; // Оклад
};
// Структура Prepod со вложенными указателями на структуру Person - pFace
struct Prepod {
char fam[50]; // Фамилия
Person * pFace; // Указатель на структурную переменную типа Person
double Oklad; // Оклад
};
// Структура со вложенными структурами Prepod
struct Decan {
char fam[50]; // Фамилия
Prepod DecPrep; // Структурная переменная Prepod вложенная (не указатель)
double Oklad; // Оклад
};

```

В программе для указателей поместим операторы вычисления (**salary**) стоимости:

```

Person p2 = { "Сидоров", {10, 3, 1978}, 20.00 };
// Динамическая структура – доступ через указатель
Prepod *ptrPerson = (Prepod *) malloc ( sizeof (Prepod));
ptrPerson -> pFace = &p2;
ptrPerson -> pFace -> salary = 15.00;
// Доступ к полю динамической структуры возможен разными способами
printf( "p2 .salary = %f \n", p2 .salary );
printf( "ptrPerson ->pFace ->salary = %f \n", ptrPerson -> pFace ->salary ); // Два указателя

```

В результате работы фрагмента программы получим на консоли:

```

p2 .salary = 15.000000
ptrPerson ->pStudent ->salary = 15.000000

```

Для вложенных структур (в структуру **Decan** вложена структура **Prepod**, см. Описания выше) можем записать и комбинированную ссылку, включая доступ и посредством указателя (**pFace ->salary**):

```

// Вложенные структуры
Decan DecIU;
DecIU.DecPrep.Oklad = 100.00;
// Доступ с указателем
DecIU.DecPrep. pFace = &p2;
DecIU.DecPrep. pFace ->salary = 10.0;

```

3.14. 3.14 Размер и размещение структур в ОП

Поля структуры располагаются в оперативной памяти последовательно, в связи с описанием в программе. При размещении в памяти разные типы должны быть выровнены на границу адреса своего размера (**int** – 2 байта, **long** – 4 байта, **double** – 8 байт и т.д.). Из-за этого, даже при равных по количеству (общему объему) и типу полей размер структуры может отличаться. Это показано на примере двух структур одинаковых при первоначальном взгляде: **First** и **Second**.

```

struct First {
int i;

```

ОП ГУИМЦ 2024 ЛРН№6

```

long j;
double k;
};
struct Second {
int i;
double k;
long j;
};

```

В программе определим актуальный размер структур и их полей:

```

// Размеры полей и величина размещения структур в ОП
printf( "Размер int = %d \n" ,sizeof (int) );
printf( "Размер long = %d \n" ,sizeof (long) );
printf( "Размер double = %d \n" ,sizeof (double) );
printf( "Размер структуры First = %d \n" ,sizeof (First) );
printf( "Размер структуры Second = %d \n" ,sizeof (Second) );

```

...

В результате работы программы получим:

```

Размер int = 4
Размер long = 4
Размер double = 8
Размер структуры First = 16
Размер структуры Second = 24

```

При выравнивании на границу для другого порядка полей размер структуры увеличивается с 16 до 24 байт! Проверьте и другие порядки расположения полей в таких структурах.

3.15. 3.15 Динамической структуры

Работа с динамическими структурами и их массивами выполняется посредством указателей соответствующих типов (указателей на структуры такого типа - **Decan * pDec**;). Выделение памяти будем производить производиться библиотечными функциями из **malloc.h**. В этой библиотеке доступны функции: **malloc**, **calloc**, **free**, **realloc** и др. На примере, размещенном ниже, показано применение этих функций для структур.

```

Decan * pDec = (Decan * ) malloc ( sizeof (Decan)); // Выделение динамической памяти
pDec ->DecPrep.Oklad = 100.00; // Работа с полями структур
strcpy( pDec ->fam , "Фамилия декана");
pDec ->Oklad = 50.00;
pDec = (Decan *) realloc( pDec , sizeof (Decan) * 2 ); // Изменение размера выделяемой
памяти
//
pDec = pDec + 1;
strcpy( pDec ->fam , "Новая Фамилия декана");
pDec = pDec-1; //Восстановление указателя для освобождения памяти(можно и сначала
запомнить)
free ( pDec );

```

Обратите внимание на использование функции **realloc**, позволяющей изменить размер выделенной памяти в два раза, а также добавить к этому указателю единицы (**pDec = pDec + 1**). При операциях целого типа с указателями определенного вида одна единица соответствует размеру типа, для которого объявлен данный указатель (у нас - **sizeof(Decan)**).

3.16. 3.16 Создание и удаление динамической структуры со строками

В динамических структурах часто возникает задача работы со строками (символьными массивами: имена, фамилии и другие тексты). Если заранее в структуре выделять максимальное число требуемых знаков в символьных массивах для строк (так мы поступали ранее, см. – **fam** , **name**), то, очевидно, будет значительный перерасход оперативной памяти, особенно в тех случаях когда выполняется работа с большими массивами структурных переменных. Поэтому при инициализации таких структур намного экономнее выделить столько байт памяти, сколько необходимо в конкретном случае, то есть динамической памяти (ДП). Кроме того, подобная процедура необходима и при изменении значений строковых полей структурных переменных, для универсальности: первоначально выполняется освобождение памяти (**free**), а затем новый захват ДП (**malloc**) “по потребности”. Покажем это на примере. Пусть есть структура, в которой два указателя на строки (**pName** и **pAvtor**):

```
struct Book{
char * pName; // Указатель на строку для Названия книги
char * pAvtor; // // Указатель на строку для Автора книги
int StrCount; // Число страниц в книге
};
```

При инициализации структурной переменной нужно захватить требуемую память, указатель запомнить в структуре и скопировать строку в полученную область ДП:

```
Book Book1; // Описание структурной переменной
char * pStr = (char *) malloc ( strlen ("Три мушкетера") + 1); // Число символов в строке
+1 для \0
Book1.pName = pStr; // Запомним указатель
strcpy(pStr , "Три мушкетера" ); // Копируем строку в поле ДП
pStr = (char *) malloc ( strlen ("Александр Дюма") + 1);
Book1.pAvtor = pStr;
strcpy(pStr , "Александр Дюма" );
Book1.StrCount =670;
```

При завершении программы динамическая память должна быть освобождена (функция освобождения - **free**):

```
// При завершении программы освободим память
free (Book1.pName);
free (Book1.pAvtor);
//
```

Если вся структура динамически порождается, этот текст программы будет выглядеть следующим образом:

```
// Динамическая структура
Book * pBook = (Book *) malloc ( sizeof(Book));
pStr = (char *) malloc ( strlen ("Две Дианы") + 1);
pBook->pName = pStr;
strcpy(pStr , "Две Дианы" );
pStr = (char *) malloc ( strlen ("Александр Дюма") + 1);
pBook->pAvtor = pStr;
strcpy(pStr , "Александр Дюма" );
pBook->StrCount =470;
// При завершении программы освободим память под строки и саму структуру
free (pBook->pName);
free (pBook->pAvtor);
free (pBook );
////////
```


Если нужно изменить значение содержимого в отдельной строке, то динамическую память предварительно освобождаем, а затем снова выделяем, так как размеры ДП строк могут не совпадать:

```
free (pBook->pName); // Заметим что было - "Две Дианы"
pStr = (char *) malloc ( strlen ("Дама с камелиями") + 1);
pBook->pName = pStr;
strcpy(pStr , " Дама с камелиями" );
```

Если память не освобождать, то очень скоро она переполнится (не хватит ресурса ОП).

3.17. 3.17 Структуры с указателями на собственный тип

Выше было сказано, что структурах нельзя использовать поля типа структурных переменных такого же типа (получается бесконечное рекурсивное описание). Однако поля - указатели на эти структурные переменные допускаются. Часто такие указатели используются для описания данных типа список. Например, элемент двухсвязного списка может выглядеть так:

```
// структуры со ссылками на самих себя – Элемент двухсвязного списка
struct Node {
    Node * pNext;
    Node * pPrev;
    int ValList;
};
```

В данном случае **pNext** и **pPrev** являются указателями на структуру **Node** и используются для организации связей в списке (следующий – предыдущий).

3.18. 3.18 Перечисления - enum

Перечисления **enum** (перенумерация) – это специальный вид структурных переменных, использующихся для универсализации программ и их большей наглядности. Например, можно перенумеровать дни недели, месяцы и т.д.

Перечисление, другими словами, - это набор именованных целых констант, используемый для большей наглядности программы и ее переносимости. Перечисление может быть использовано и для объявления переменных, которые принимают значения на заданном множестве значений. Перечисления - это по сути множество целых констант, используемых в программе на этапе компиляции. Формально перечисления (**enum**) могут быть описаны так:

```
enum [ <имя> ] {список перечисления} [ список переменных];
```

Значение **<имя перечисления>** может быть опущено. Например, для дней недели можно задать список констант:

```
// Дни недели
enum { mon , tue , wed , thu , fri , sat , sun };
int d = mon; // При этом значения mon = 0 , tue = 1 и т.д.
```

Можно задать также имя перечисления для отдельного описания переменных этого перечисления:

```
// Дни недели с именем day
enum day { mon , tue , wed , thu , fri , sat , sun };
day dw = sun; // Получим dw = 6
```

Можно задать перечисления и с произвольными значениями констант, например:

```
enum { Con1 = 5 , Con2 = 8, Con3 = 15 };
printf("Перечисления с произвольными значением: Con1=%d Con2=%d Con3=%d\n" ,
Con1 , Con2, Con3 );
```

Получим после выполнения:

Перечисления с произвольными значениями: Con1=5 Con2=8 Con3=15

Константы можем проверять в условиях:

// Проверка значений перечислений в операторе ветвления

```
int m = Con1;
```

```
if( m ==Con1 )
```

```
printf("значение: Con1 , m =%d\n" , m ); // Совпадение переменной и константы
```

```
else
```

```
printf("значение: не Con1 \n" );
```

Получим после выполнения фрагмента:

значение: Con1 , m = 5

Можно задать перечисления с одним базовым значением:

```
enum { Const1=5 , Const2, Const3 };
```

```
printf("Перечисления с базовым значением: Const1=%d Const2=%d Const3=%d\n" ,  
Const1 , Const2, Const3 );
```

Результат будет таким:

Перечисления с базовым значением: Const1=5 Const2=6 Const3=7

Более детально по технологиям работы с перечислениями вы можете познакомиться в литературе по языку СИ [1,2].

3.19. 3.19 Указатели на поля структуры

В отдельных случаях можно задать указатель на отдельное поле структуры, а для доступа к полям использовать специальные операции (.* при наличии самой структуры) и (->.* при наличии указателя на структуру). Это позволяет сделать программу более динамичной, так как во время выполнения программы можно его изменить (задать адрес другого поля того же типа но той же структуры).

Для демонстрации изменим нашу структуру, добавив в нее новое поле целого типа

(Kurs2):

```
struct Student { // Структура со статическими строками
```

```
char Name[20]; //
```

```
char Fam[20]; // Динамические строки
```

```
int Kurs;
```

```
float Stipen;
```

```
int Kurs2; }; // для проверки указателя на поле
```

Покажем работу указателя на поле на примере (указатель на поле выделен красным цветом **pPole**):

```
// Указатель на поле структуры (объявление)
```

```
int Student::*pPole;
```

```
// Указатель на поле структуры (задание значения)
```

```
pPole=&Student::Kurs; // здесь может быть и другое целое поле структуры
```

```
Student S8={ "Сидоров" , "Василий" , 1 , 2000.0f , 10 }; // Kurs = 1 , Kurs2 = 10
```

```
// Структура и указатель на поле
```

```
S8.*pPole =2; // Kurs = 2
```

```
// На Указатель pPole на поле задаем на новое поле (Kurs2 = 10 предварительно)
```

```
pPole=&Student::Kurs2;
```

```
S8.*pPole =3; // Kurs2 = 3
```

```
// Указатель на структуру и указатель и на поле вместе
```

```
Student * pStud;
```

```
pStud=&S8;
```

```
pStud->*pPole=6; // Kurs = 6
```

Проверить работу фрагмента программы можно в отладчике или выполнить распечатку значений полей до и после изменения.

3.20. 3.20 Союзы – union - объединения

В некоторых случаях структуру нужно использовать для данных, которые перекрываются в оперативной памяти. Для этого могут быть использованы специальные структуры -объединения – (**union**). Объединения являются полноправными структурами данных: имеют шаблон, имеют поля, могут использоваться для описания структурных переменных. Поля в структура типа union перекрывают друг друга. Объединение (наложения) — это фактически выделенное место в памяти (они могут быть и динамическими), которое используется для хранения переменных, даже разных типов. Получается, что можно задавать разные имена, одним и тем же полям оперативной памяти. Покажем на примере. Пусть есть объединение содержащее три поля разной длины (Пусть это три разных типа: **int** , **float** и **char ***).

```
union u_tag {
    int ival; // целое поле
    float fval; // вещественное поле
    char *sval; // указатель на строку
};
```

После описания новой переменной (**U1**) и выполнения операций по занесению данных получим следующий результат (желательно посмотреть в отладчике в пошаговом режиме переменную U1!):

```
u_tag U1; // u_tag - это структурная переменная нового типа данных!!!!
U1.ival = 5;
printf("Поле объединения в целом формате: u.ival=%d\n" , U1.ival );
U1.fval = 0.5f; // Изменим fval
printf("Поле объединения в действительном формате: u.ival=%5.2f\n" , U1.fval );
printf("Поле объединения в целом формате: u.ival=%d\n" , U1.ival ); // Попытка печати (F)
как целое!!
char * pStrU = (char *)malloc(100) ;
U1.sval = pStrU;
strncpy(U1.sval, "Объединения",100);
// Заполнение указателя союза (sval)
printf("Поле объединения в адреса (указатель): u.sval =%p\n" , U1.sval );
printf("Поле объединения в символьном формате (адрес): u.sval =%s\n" , U1.sval );
free (pStrU); // Освобождение памяти
```

Результат будет таким:

```
Поле объединения в целом формате: u.ival=5
Поле объединения в действительном формате: u.ival= 0.50
Поле объединения в целом формате: u.ival=1056964608
Поле объединения в адреса (указатель): u.sval =004C79A8
Поле объединения в символьном формате (адрес): u.sval =Объединения
```

```
Поле объединения в действительном формате: u.ival= 0.50
Поле объединения в целом формате: u.ival=1056964608
Поле объединения в символьном формате: u.sval =Объединения
```

В третьей строке вывода для примера показана попытка печати данных как целого числа, хотя на это место было предварительно занесено вещественное число (**float**). Объединения могут использоваться при компактной передаче данных, при хранении или при плотной битовой упаковке данных (см. в литературе использование СИ для работы с битами и данными).

4. 4. Примеры программы с использованием структур

Вторая часть задания в данной лабораторной работе, помимо первой, связанной с изучением теоретического раздела заключается в том, чтобы проверить и испытать в проекте СИ уже отлаженные программы и фрагменты программ. Возможно, что, осваивая теоретическую часть работы, вы уже на компьютере проверили выполнение фрагментов текста и применения различных операторов ветвления (из раздела 3), тогда вам будет проще продемонстрировать их работу преподавателю. В дополнение к примерам, расположенным выше нужно испытать и изучить примеры расположенные ниже. Эти действия нужно проделать в отладчике.

Для этого нужно создать пустой проект в MS VS (Test_LR2), как описано выше, скопировать через буфер обмена в него текст данных примеров, отладить проект и выполнить фрагменты программ, которые уже работают.

4.1. 4.1 Примеры, описанные в теоретической части ЛР

Нужно внимательно изучить и проверить работу всех примеров из теоретической части ЛР. Эти примеры расположены выше. Все примеры можно скопировать в свой проект. Все эти задания выполняются обязательно, они не требуют дополнительной отладки и легко (через буфер обмена -**Clipboard**) переносятся в программу. Все фрагменты должны демонстрироваться преподавателю. В частности, в первой части (раздел 3), представлены следующие примеры:

Описание простой структуры: типа **Student, Complex, Date, Person** (Разделы: 3.2, 3.3).

Демонстрация примеров инициализации структур (Разделы: 3.2, 3.3).

Работа со структурами через указатели и квалифицированные ссылки (Раздел: 3.4).

Многоуровневая квалификация полей во вложенных структурах (Раздел: 3.5).

Использование указателей для доступа к полям структур (Раздел: 3.5).

Демонстрация передачи структур в функцию и указателей на структуры (Разделы: 3.7, 3.8).

Использование массивов структур (Раздел: 3.9).

Показать использование вложенных структур и доступа к полям (Раздел: 3.10).

Показать определение размера структуры (Раздел: 3.11).

Пример использования динамических структур и динамических строк в них (Раздел: 3.5, 3.12, 3.13, 3.14).

Показать использование перечислений (Раздел: 3.15).

Показать использование объединений (Раздел: 3.16).

Кроме этого ниже представлены примеры, которые могут быть полезными, в том числе и при выполнении контрольных заданий. Их тоже целесообразно изучить и проверить их работу в реальном пошаговом режиме на компьютере.

4.2. 4.2 Копирование и обмен статических структур

Если в структурных переменных все поля статические (фиксированный размер), то можно копировать структуру целиком. Отметим, что при копировании и обмене в функции параметры для структур должны задаваться указателями. Пусть есть структура следующего вида:

// Структура **Person** для примеров

```
struct Person {
    char Name[20]; // Фамилия
    int Kurs; // Курс
    float Stipen; // Стипендия
```

```

};
...

/// Функции для структуры Person печать содержимого структурной переменной
void PersonPrint( Person P) //Для печати в функции указателя не нужно
{
    printf( "Имя - %s  Курс - %d  Стипендия - %6.2f р. \n" ,P.Name, P.Kurs , P.Stipen );
};

// Функции копирования и обмена структур по указателю
// Копирование структур Person
void CopyPerson( Person * P1 , Person * P2) //Для изменения в функции нужны указатели
{
    P1->Kurs = P2->Kurs;
    P1->Stipen = P2->Stipen;
    strcpy( P1->Name , P2->Name);
};

// Обмен статических структур
void SwapPerson( Person * P1 , Person * P2) //Для изменения в функции нужны указатели
{ // Структура статическая копируем память
    Person Temp;
    CopyPerson( &Temp , P1);
    CopyPerson( P1 , P2);
    CopyPerson( P2 , &Temp);
};

...

// Прототипы функций, представленных выше, в главной программе, если нужно
void PersonPrint( Person P);
void CopyPerson( Person * P1 , Person * P2);
void SwapPerson( Person * P1 , Person * P2);

...

// Вызов функций из главной программы
// Печать структуры
Person Stud ;//
// Заполнение структуры вручную
strcpy (Stud.Name , "Петров");
Stud.Kurs = 2;
Stud.Stipen = 2000.00f ;
// печать
PersonPrint ( Stud );
// Инициализация структуры
Person StudNew= {"Аксенова" , 10 , 50000.0f} ;//
PersonPrint ( StudNew );

Результат вызова функций имеет вид:
Имя - Петров  Курс - 2  Стипендия - 2000.00 р.
Имя - Аксенова  Курс - 10  Стипендия - 50000.00 р.
////////////////////////////////

// Копирование и обмен начальная инициализация
Person S11= {"Аксенова" , 10 , 50000.0f} ;//
Person S21= {"Большаков" , 11 , 60000.0f} ;
Person S31;
//
    printf("Сору функцией: \n");
// Копирование
CopyPerson( &S31 , &S21);
    PersonPrint ( S31 );
    PersonPrint ( S21);//

Результат вызова функций имеет вид:

```

Сору функцией:

Имя - Большаков Курс - 11 Стипендия - 60000.00 р.

Имя - Большаков Курс - 11 Стипендия - 60000.00 р.

Для копирования структур без динамики можно использовать функции RTL (**memcpy**):

```
// Копирование структур средствами RTL
printf("Непосредственно в ОП - memcpy: \n");
Person Stud1 = { "Сидоров", 1, 100.00f};
Person Stud2 = { "", 0, 0.00f};
PersonPrint ( Stud1 );
PersonPrint ( Stud2 );
// Stud2 = Stud1; // Это возможности C++
memcpy ( &Stud2, &Stud1, sizeof (Person) ); // Так можно если нет указателей
PersonPrint ( Stud2 );
```

Результат вызова функций имеет вид:

Непосредственно в ОП - memcpy:

Имя - Сидоров Курс - 1 Стипендия - 100.00 р.

Имя - Курс - 0 Стипендия - 0.00 р.

Имя - Сидоров Курс - 1 Стипендия - 100.00 р.

```
//
printf("Swap: \n");
PersonPrint ( S11 );
PersonPrint ( S21);
// Замена
SwapPerson( &S11, &S21);
PersonPrint ( S11 );
PersonPrint ( S21);
```

//

Результат вызова функций имеет вид:

Swap:

Имя - Аксенова Курс - 10 Стипендия - 50000.00 р.

Имя - Большаков Курс - 11 Стипендия - 60000.00 р.

Имя - Большаков Курс - 11 Стипендия - 60000.00 р.

Имя - Аксенова Курс - 10 Стипендия - 50000.00 р.

Результаты выполненных фрагментов программ должны совпадать с результатами данных методических указаний.

4.3. 4.3 Функция Swap для структур с динамическими строками

Рассмотрим простую структуру с динамическими строками(см. выше). При инициализации структуры нужно выделять динамическую память под имя (указатель **pName**) и фамилию (указатель **pFam**). В данном примере нас уже предусмотрена функция печати структуры (**StudentPrint**) и обмена структур с учетом динамики (**SwapStudent**).

```
//
struct Student { // Структура с динамическими строками
char * pName; // Динамические строка
char * pFam; // Динамические строк
int Kurs;
float Stipen;
};

// Функция печати
void StudentPrint( Student S)
{
printf( "Фамилия - %s Имя - %s Курс - %d Стипендия - %6.2f р. \n"
,S.pFam,S.pName,
S.Kurs, S.Stipen );
};
// Функция обмена с учетом изменения размера ДП
```

```

void SwapStudent( Student * pA , Student * pB )
{ // 1) Temp = a , 2) a = b , 3) b = Temp  => Обычный обмен!!!
  Student Temp; // Временная для хранения
  //1)
  Temp.Kurs = pA->Kurs;
  Temp.Stipen = pA->Stipen;
  Temp.pName = (char *) malloc ( strlen(pA->pName) + 1);
  Temp.pFam = (char *) malloc ( strlen(pA->pFam) + 1);
  strcpy ( Temp.pName , pA->pName);
  strcpy ( Temp.pFam , pA->pFam);
  free ( pA->pName); // Освобождение старой ДП для pA
  free ( pA->pFam); // Освобождение старой ДП для pA
  //2)
  pA->Kurs = pB->Kurs;
  pA->Stipen = pB->Stipen;
  pA->pName = (char *) malloc ( strlen(pB->pName) + 1);
  pA->pFam = (char *) malloc ( strlen(pB->pFam) + 1);
  strcpy ( pA->pName , pB->pName);
  strcpy ( pA->pFam , pB->pFam);
  free ( pB->pName); // Освобождение старой ДП для pB
  free ( pB->pFam); // Освобождение старой ДП для pB
  //3)
  pB->Kurs = Temp.Kurs;
  pB->Stipen = Temp.Stipen;
  pB->pName = (char *) malloc ( strlen(Temp.pName) + 1);
  pB->pFam = (char *) malloc ( strlen(Temp.pFam) + 1);
  strcpy ( pB->pName , Temp.pName);
  strcpy ( pB->pFam , Temp.pFam);
  free ( Temp.pName); // Освобождение старой ДП для Temp
  free ( Temp.pFam); // Освобождение старой ДП для Temp
};

```

Прототипы наших функций в главной программе:

```
void StudentPrint( Student S);
```

```
void SwapStudent( Student * pA , Student * pB );
```

Текст в главной программе: инициализация, распечатка, обмен и новая распечатка:

```

// Новая структура Student с динамическими строками (фамилия и имя)
Student Ivanov;
// Явное заполнение структуры Ivanov
Ivanov.Kurs = 5;
Ivanov.Stipen = 5000.00f ;
Ivanov.pFam = (char *) malloc (strlen ("Иванов") + 1) ; // Инициализация
Ivanov.pName = (char *) malloc ( strlen ("Иван") + 1 ) ;
strcpy ( Ivanov.pFam , "Иванов" );
strcpy ( Ivanov.pName , "Иван" );
StudentPrint( Ivanov);
// Неявное выделение динамической памяти при инициализации для структуры Sidorov
Student Sidorov = { "Сидор" , "Сидоров" , 3 , 2000.0f};
StudentPrint( Sidorov);
// Нельзя изменять строки для структуры Sidorov
//strcpy ( Sidorov.pName , "Иван" ); // и это ошибка, так как pName указывает на строку -
константу
// Для возможности изменения нужно выделить ДП для того, чтобы работали указатели!!
Sidorov.pName =(char *) malloc (strlen ("Петр") + 1);
strcpy ( Sidorov.pName , "Петр" );
Sidorov.pFam =(char *) malloc (strlen ("Петров") + 1);
strcpy ( Sidorov.pFam , "Петров" );
StudentPrint( Sidorov);

```



```

//SWAP для структуры Student
printf("ДО SWAP функцией: \n");
StudentPrint( Sidorov);
StudentPrint( Ivanov);
printf("SWAP функцией: \n");
SwapStudent( &Sidorov , &Ivanov ); // Обмен динамических структур – передаем
указатели
StudentPrint( Sidorov);
StudentPrint( Ivanov);
// Освободить динамическую память под строки
free (Ivanov.pFam);
free (Ivanov.pName);
free (Sidorov.pFam);
free (Sidorov.pName);

//

```

Результат вызова функций имеет вид:

```

Фамилия - Иванов  Имя - Иван  Курс - 5  Стипендия - 5000.00 р.
Фамилия - Сидоров  Имя - Сидор  Курс - 3  Стипендия - 2000.00 р.
Фамилия - Петров   Имя - Петр  Курс - 3  Стипендия - 2000.00 р.
ДО SWAP функцией:
Фамилия - Петров   Имя - Петр  Курс - 3  Стипендия - 2000.00 р.
Фамилия - Иванов  Имя - Иван  Курс - 5  Стипендия - 5000.00 р.
SWAP функцией:
Фамилия - Иванов  Имя - Иван  Курс - 5  Стипендия - 5000.00 р.
Фамилия - Петров   Имя - Петр  Курс - 3  Стипендия - 2000.00 р.

```

Синим цветом - работа функции после обмена структур.

4.4. 4.4 Заполнение случайных - rand числовых полей массива структур

В следующем примере используются функции заполнения массива структурных переменных случайными значениями. Для этого в библиотеке СИ (**stdlib.h**) имеются специальные функции: **srand** (задание начального значения ряда случайных чисел) и **rand** (генерации вещественного случайного числа в диапазоне от 0 до **RAND_MAX**).

```

#include <stdlib.h>
#include <time.h>

...
//
struct Student { // Структура с динамическими строками
char * pName; // Динамические строка
char * pFam; // Динамические строк
int Kurs;
float Stipen;
};

...
// Заполнение статического массива структур случайными числами
const int Rzm = 6;
Student Potok[ Rzm ]; // Статический массив
// Для запуска новой последовательности случайных чисел
srand( (unsigned) time( NULL ) );
//srand( (unsigned) NULL ); // Если NULL то при новом запуске программы
последовательность //повторяется
//Цикл заполнения массива структур
for (int i = 0 ; i < Rzm ; i++ )
{
// Локальные временные массивы для генерации имен и номеров
char Buf[20];
char Num[5];
char Buf2[20];

```


ОП ГУИМЦ 2024 ЛРН№6

```

char Num2[5];
//
strcpy(Buf, "Stud № - "); // Условная фамилия
strcpy(Buf2, "Num - "); // условное имя
int k = ( i < Rzm/2 ) ? 1 : 2 ; // Курс для целого – условное выражение
// (1- Rzm/2) = 1, а (Rzm/2 - Rzm) = 2 (первая половина массива 1, а вторая 2 – без случайностей)
// Стипендия задается 0 - 10000 Для вещественных (случайный диапазон)
float St = 1000.0f * 10.0f * rand() / RAND_MAX ;
// 0 - 99 – номер добавка для фамилии: "Stud № - " + этот случайный номер
int n = (rand()*99)/ RAND_MAX ;
strcat(Buf, itoa (n + 1 ,Num, 10 ));
// 0 - 30 - номер добавка для имени: "Num - " + + этот случайный номер
n = (rand()*30)/ RAND_MAX ;
strcat(Buf2, itoa (n + 1 ,Num, 10 ));
// Добавим пробелы
strcat(Buf, " ");
strcat(Buf2, " ");
// Заполним отдельную структуру массива по индексу i (статика)
// Каждая структура статическая а строки динамические из буферов Buf и Buf2
Potok[i].Kurs = k;
Potok[i].Stipen = St;
// Выделяем динамическую память и копируем по указателю
Potok[i].pFam = (char *) malloc( strlen(Buf) + 1 );
strcpy(Potok[i].pFam, Buf );
Potok[i].pName = (char *) malloc( strlen(Buf2) + 1 );
strcpy(Potok[i].pName, Buf2 );
};
// Цикл печати массива структур через функцию
for (int i = 0 ; i < Rzm ; i++)
    StudentPrint( Potok[i]);

```

Результат распечатки массива структур имеет вид, все поля кроме курса случайные:

Фамилия - Stud № - 11	Имя - Num - 3	Курс - 1	Стипендия - 1115.45 р.
Фамилия - Stud № - 82	Имя - Num - 15	Курс - 1	Стипендия - 6272.16 р.
Фамилия - Stud № - 9	Имя - Num - 6	Курс - 1	Стипендия - 4890.59 р.
Фамилия - Stud № - 86	Имя - Num - 10	Курс - 2	Стипендия - 1998.05 р.
Фамилия - Stud № - 52	Имя - Num - 19	Курс - 2	Стипендия - 4951.32 р.
Фамилия - Stud № - 80	Имя - Num - 8	Курс - 2	Стипендия - 2121.65 р.

При динамическом выделении памяти под массив (10 студентов):

```

const int Rzm = 10;
Student * pPotok = (Student *) calloc( Rzm , sizeof(Student) );

```

Работа в цикле через указатель будет выглядеть так:

```

// Заполним структуру динамического массива по индексу i
(pPotok + i) ->Kurs = k;
(pPotok + i) ->Stipen = St;
(pPotok + i) ->pFam = (char *) malloc( strlen(Buf) + 1 );
strcpy((pPotok + i) ->pFam, Buf );
(pPotok + i) ->pName = (char *) malloc( strlen(Buf2) + 1 );
strcpy((pPotok + i) ->pName, Buf2 );

```

4.5. 4.5 Сортировка массива структур с помощью функции Swap

Сортировка массива **Potok** сгенерированного в предыдущем примере выполняется пузырьковым методом. При этом воспользуемся функцией **SwapStudent**, рассмотренной выше. Эта функции и ее применении описано выше. Текст программы сортировки приведен ниже:

```

// Сортировка массив структур Potok
for( int k = 0 ; k < Rzm - 1 ; k++)

```

```

{
for ( int i=0 ; i< Rzm - 1 ; i++ )
{
    if ( strcmp(Potok[i].pFam , Potok[i+1].pFam ) < 0 ) // Убывание по фамилии
        SwapStudent( &Potok[i] , &Potok[i + 1] );
};
};

// Печать после сортировки
printf("После сортировки 1: \n");
for( int k=0 ; k< Rzm ; k++)
    StudentPrint ( Potok[k]);

```

Результат распечатки массива структур после сортировки по фамилии имеет вид:

После сортировки 1:

Фамилия - Stud № - 67	Имя - Num - 21	Курс - 2	Стипендия - 4552.14 р.
Фамилия - Stud № - 48	Имя - Num - 19	Курс - 2	Стипендия - 2043.82 р.
Фамилия - Stud № - 43	Имя - Num - 29	Курс - 1	Стипендия - 6239.81 р.
Фамилия - Stud № - 33	Имя - Num - 17	Курс - 2	Стипендия - 8512.83 р.
Фамилия - Stud № - 28	Имя - Num - 28	Курс - 1	Стипендия - 2082.28 р.
Фамилия - Stud № - 16	Имя - Num - 22	Курс - 1	Стипендия - 9506.82 р.

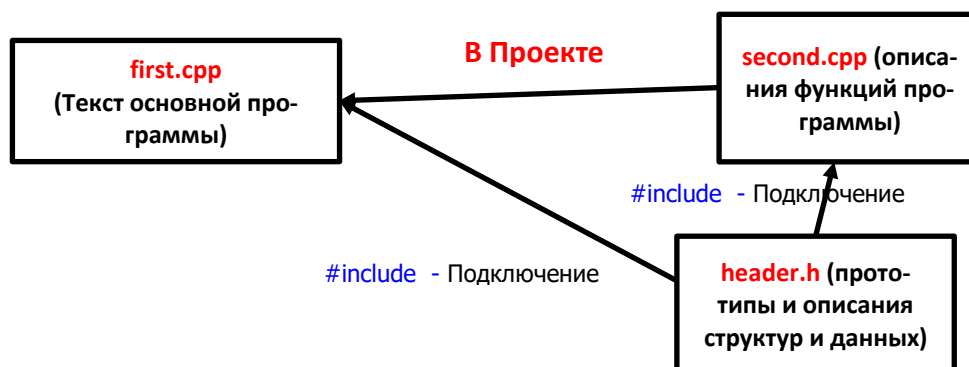
Мы выбрали массив со случайной генерацией всех полей. При сортировке по фамилии текстовый номер в ней (он находится в конце строки) определяет убывание. Сортировка основана на использовании функции сравнения строк - **strcmp**.

Нужно создать пустой проект в **MS VS**, как описано выше, скопировать через буфер обмена в него текст данного примера, отладить его и выполнить.

5. 5. Контрольные задание ЛР №6.

5.1. 5.1 Создание консольного проекта ЛР

Создать пустой консольный проект для выполнения ЛР № 6. Пример создания консольного проекта и его русификация рассмотрено в методических указаниях к ЛР №1 по дисциплине ОП (см. на сайте). В проект обязательно включить три модуля: **first.cpp** (для главной программы проекта), **second.cpp** (для размещения функций проекта) и **header.h** (для описаний структур и прототипов функций проекта). На рисунке структура проекта выглядит так:



5.2. 5.2 Описать структуру своего варианта ДЗ и работа с ней

ВНИМАНИЕ: Все функции, разрабатываемые в задании, в названии должны иметь название собственной структуры (у нас **Student**). Например, у нас в примерах: **StudentPrint** , **SwapStudent** , **StudentArrayPrint** и т.д.

Описать свою структуру (по своему варианту см. таблицу ниже – раздел б) со статическими строками и описать первоначально одну структурную переменную этого типа структур.

Поля вашей структуры должны быть иметь удобные названия для запоминания и использования. Число полей в шаблоне структуры должно быть **не менее 6-ти разных полей**. два поля должны быть числовыми(целое и вещественное). Как минимум, два поля должны быть строкой (массивом типа **char**). Все поля должны быть осмыслены (иметь определенный смысл для вашей структуры). Одно поле должно быть на выбор типа перечисления (цвета, дни,...) и/или вложенным типом структуры (например дата и т.д.). Они должны соответствовать предметной области для названия собственной структуры. Поля должны иметь разные типы: целое, вещественное и строка (статическая). Нужно очень хорошо подумать где, в каких задачах может использоваться данная структура в дальнейшем. Желательно сформировать список применений в программах данной структуры, опираясь на собственный опыт, фантазию. Можно поговорить с другими людьми, студентами, родителями и преподавателями. Качество и осмысленность выбранных полей и задач будет отдельно оцениваться.

Например, для структуры типа студент можно выбрать: фамилия, курс, пол, группа, размер стипендии и т.д. Структура может иметь статические строки и строки, формируемые в программе динамически (см. ЛР №4) (поместить в файл **header.h** проекта):

```
//
struct Student { // Структура со статическими строками
char Name[20]; // Имя студента
char Fam[20]; // Фамилия студента
int Kurs; //Курс студента
float Stipen; //Размер стипендии студента
};
Или
struct Student1 { // Структура с динамическими строками
char * pName; // Динамические строки
char * pFam; // Динамические строки
int Kurs;
float Stipen;
};
```

Нужно в программе вручную выполнить инициализацию всех полей простой структурной переменной (при описании), и сделать распечатку всех полей на консоли (функция **printf**) и продемонстрировать изменение полей вашей структуры (операторы присваивания для числовых полей и функция **strcpy** для строковых полей). После изменения полей результат также нужно распечатать. Нужно научиться просматривать поля структуры в отладчике VS и демонстрировать значение структурных переменных в пошаговом режиме отладчика (см. примеры ниже и теоретической части ЛР № 6).

Далее, нужно описать указатель на свою статическую структуру и произвести изменение каждого из полей статической структуры через этот указатель на структурную переменную. Полученный результат также распечатать через указатель (printf). Примеры приводятся на основе структуры **Student** со статическими строками (см. выше). Это описание нужно поместит в заголовочный файл **header.h** проекта.

```
#include "header.h"
```

```
#include <stdio.h>
#include <string.h>
#include <process.h>
#include <malloc.h>
#include <stdlib.h>
```

Описание и инициализация своей структуры:

```
// Описание и инициализация структуры
Student S1 = { "", "", 1, 2000.0f }; // Пустая
Student S2 = { "Иван", "Петров", 1, 2000.0f };
/// Инициализация
//Student S3 = {"Иван", "Петров", 5, 2000.0f };
struct Student S3= { "", "", 4, 20.0f }; // Пустая ytbywbfbkpbhjdffuifz
// 5.2 Доступ через структурную переменную
S1.Kurs = 2;
S1.Stipen = 5000.0f;
strcpy_s(S1.Name, "Петр");
strcpy_s(S1.Fam, "Иванов");
```

Печать структуры:

```
printf ( "Структура Student (печать через имя структуры):");
// Печать структуры Student
printf( "\nФамилия - %s \nИмя - %s \nКурс - %d \nСтипендия - %6.2f р. \n"
, S2.Fam, S2.Name, S2.Kurs, S2.Stipen );
```

Работа со статической структурой через указатель:

```
// 5.2 доступ через указатель
// Описание указателя
Student * pStudent;
// Вычисление указателя и изменение через указатель
pStudent = &S1; // УКАЗАТЕЛЬ смотрим в отладчике
pStudent->Kurs = 3;
pStudent->Stipen = 10000.0f;
strcpy_s(pStudent->Name, "Петр2");
strcpy_s(pStudent->Fam, "Иванов2");
```

Печать структуры через указатель:

```
// Печать через указатель статической структуры
printf( "\nФамилия - %s \nИмя - %s \nКурс - %d \nСтипендия - %6.2f р. \n" , pStudent-
>Fam, pStudent->Name,
pStudent->Kurs, pStudent->Stipen );
```

Далее нужно выделить динамическую память под структуру, ее заполнить вручную (операторы присваивания для числовых полей и функция **strcpy** для строковых полей) и распечатать ее поля (функция **printf**).

Работа с динамической структурой через указатель:

```
// выделение ДП для структуры
pStudent = (Student *) malloc(sizeof(Student));
// Заполнение структуры из ДП
pStudent->Kurs = 3;
pStudent->Stipen = 20000.0f;
strcpy_s(pStudent->Name, "Антон");
strcpy_s(pStudent->Fam, "Смирнов");
```

Печать динамической структуры:

```
// Печать динамической структуры Student
printf( "\nФамилия - %s \nИмя - %s \nКурс - %d \nСтипендия - %6.2f р. \n" , pStudent-
>Fam, pStudent->Name,
pStudent->Kurs, pStudent->Stipen );
// Освобождение ДП выделенной под динамическую структуру
```

```
free (pStudent); // Освободить динамическую память
```

5.3. 5.3 Функция Распечатки структуры

Описать новую структурную переменную и заполнить все ее поля одним из способов представленных выше. Создать функцию распечатки своей структуры (**Print...**). Параметр функции структурная переменная (не указатель!). Продемонстрировать работу этой функции из главной программы. Выполнить пошаговую отладку с заходом в функцию печати.

Описание функции (описание поместим в файл **second.cpp**):

```
// Описание функции
printf( "Структура Student в функции:");
// Печать структуры Student
void StudentPrint( Student S)
{
printf( "\nФамилия - %s \nИмя - %s   \nКурс - %d   \nСтипендия - %6.2f p.   \n"
,S.Fam,S.Name,
        S.Kurs , S.Stipen , S.Kurs , S.Stipen ); };
```

Прототип функции (разметить в файле проекта **header.h**):

```
void StudentPrint( Student S);
```

Использование (текст в главном модуле **first.cpp**):

```
// 5.3 Структуры для печати, заполнение см. выше
Student S1 = { "" , "", 1 , 2000.0f }; // Пустая
Student S2 = { "Иван" , "Петров", 1 , 12000.0f };
// 5.3 Функция печати вызов функции
StudentPrint(S1);
//Вторая структура
StudentPrint(S2);
```

Результат печати функцией(поля структуры S1 заполнены/изменены в предыдущем пункте):

Структура Student в функции:

Фамилия -

Имя -

Курс - 1

Стипендия - 2000.00 p.

Структура Student в функции:

Фамилия - **Петров**

Имя - Иван

Курс - 1

Стипендия - 12000.00 p.

5.4. 3.4 Функция Распечатки структуры через указатель

Описать структуру и заполнить все поля. **Создать функцию** распечатки своей структуры (**Print...**). Параметр функции указатель на структурную переменную своего типа. Описать свою структуру и вызвать функцию печати через указатель. Продемонстрировать работу функции из главной программы.

Использование функции печати через указатель (описание поместим в файл **second.cpp**):

```
void StudentPrintP( Student * pS)
{
printf( "Структура Student печать в функции (указатель):");
printf( "\nФамилия - %s \nИмя - %s   \nКурс - %d   \nСтипендия - %6.2f p.   \n" ,pS->Fam,pS->Name,
        pS->Kurs , pS->Stipen ); };//
```

Прототип функции (разметить в файле проекта **header.h**):

```
void StudentPrintP( Student * pS)
```

Использование функции печати (текст в главном модуле **first.cpp**):

```
// 5.3 Структуры для печати, заполнение см. выше
Student S1 = { "", "", 1, 2000.0f }; // Пустая
Student S2 = { "Иван", "Петров", 1, 2000.0f };
pStudent = &S1; // смотрим в отладчике
```

```
// Печать через указатель
```

```
StudentPrintP(&S2);
```

```
pStudent = &S1;
```

```
StudentPrintP(pStudent); // указатель на S1
```

Результаты работы функции печати (структурная переменная S1 изменена выше):

Фамилия - Петров Имя - Иван Курс - 1 Стипендия - 2000.00 р.

Структура Student печать в функции (указатель):

Фамилия - Петров

Имя - Иван

Курс - 1

Стипендия - 2000.00 р.

Структура Student печать в функции (указатель):

Фамилия - Иванов1

Имя - Петр1

Курс - 1

Стипендия - 10000.00 р.

5.5. 5.5 Массив структур, его инициализация и функция распечатки

Описать массив своих структур, выполнить его инициализацию и одновременно распечатать массив структур в одном цикле, используя функцию распечатки одной структуры, разработанную ранее через указатель.

```
//5.5 Массив структур и его описание инициализация
```

```
Student Group[] = { { "Сидоров", "Василий", 2, 2000.0f }, { "Иван", "Петров", 2, 2000.0f } };
```

```
// Распечатка массива структур
```

```
printf("Группа студентов:\n");
```

```
for ( int i = 0 ; i < sizeof(Group)/sizeof(Student) ; i++ )
```

```
    StudentPrint(Group[i]);
```

```
printf("Группа студентов(Указатели):\n");
```

```
for ( int i = 0 ; i < sizeof(Group)/sizeof(Student) ; i++ )
```

```
    StudentPrintP(&Group[i]);
```

Результат:

Группа студентов:

Структура Student в функции:

Фамилия - Василий

Имя - Сидоров

Курс - 2

Стипендия - 2000.00 р.

Структура Student в функции:

Фамилия - Петров

Имя - Иван

Курс - 2

Стипендия - 2000.00 р.

Группа студентов(Указатели):

Структура Student печать в функции (указатель):

Фамилия - Василий

Имя - Сидоров

Курс - 2

Стипендия - 2000.00 р.

ОП ГУИМЦ 2024 ЛР№6

Структура Student печать в функции (указатель):

Фамилия - Петров

Имя - Иван

Курс - 2

Стипендия - 2000.00 р.5.5.1.

Создать функцию распечатки массива своих структур. Параметр передается указателем на массив, второй параметр (**int**) определяет размер массива. Продемонстрировать работу данной функции для массива из предыдущего пункта.

```
// Печать массива структур
void StudentArrayPrint( Student * pMas , int Razm )
{
    for ( int i = 0 ; i < Razm ; i++ )
        StudentPrint(pMas[i]); }
// Прототипы
void StudentPrint( Student S);
void StudentPrintP( Student * pS);
// Прототип Функции печати массива структур
void StudentArrayPrint( Student * pMas , int Razm );

// Вызов функции печати массива
// Имя массива структуры = указателю на эту структуру
StudentArrayPrint( Group , sizeof(Group)/sizeof(Student) );
Или:
pStudent = Group; // Указатель на массив pStudent = Group
StudentArrayPrint( pStudent , sizeof(Group)/sizeof(Student) );
```

Результат (без повтора):

Фамилия - Василий

Имя - Сидоров

Курс - 2

Стипендия - 2000.00 р.

Структура Student печать в функции (указатель):

Фамилия - Петров

Имя - Иван

Курс - 2

Стипендия - 2000.00 р.

Функция для печати массива:

Структура Student в функции:

Фамилия - Василий

Имя - Сидоров

Курс - 2

Стипендия - 2000.00 р.

Структура Student в функции:

Фамилия - Петров

Имя - Иван

Курс - 2

Стипендия - 2000.00 р.

...

5.6. 5.6 Функция копирования структур

Создать и продемонстрировать работу функции копирования своих структур. Исходная структура и структура для копирования передаются в функцию через указатели. При копировании структур применить функцию копирования строк. Продемонстрировать применение функции копирования структур. Распечатать исходные структуры до и после вызова функции копирования.

```
// 5.9 Функция копирования структур
void CopyStudent ( Student * Dest , Student * Source) {
    Dest->Kurs = Source->Kurs;
    Dest->Stipen = Source->Stipen;
```


ОП ГУИМЦ 2024 ЛР№6

```
strcpy_s(Dest->Name, Source->Name);
strcpy_s(Dest->Fam, Source->Fam); };

printf("ДО COPY:\n");
StudentPrint(S1);
StudentPrint(S2);

// 5.9 Вызов функции копирования структур
CopyStudent ( &S2 , &S1 );
//
printf("ПОСЛЕ COPY:\n");
StudentPrint(S1);
StudentPrint(S2);
printf("До COPY S1=S3:\n");
StudentPrint(S1);
StudentPrint(S3);
ДО COPY:
Фамилия - Петров  Имя - Иван  Курс - 1  Стипендия - 2000.00 р.
Фамилия - Иванов2  Имя - Петр2  Курс - 3  Стипендия - 10000.00 р.
ПОСЛЕ COPY:
Фамилия - Петров  Имя - Иван  Курс - 1  Стипендия - 2000.00 р.
Фамилия - Петров  Имя - Иван  Курс - 1  Стипендия - 2000.00 р.
```

5.7. 5.7 Функция обмена (Swap) для структур

Создать функцию взаимного обмена (**Swap...**) для двух структур своего варианта. В структуре все поля статические (строки – массивы фиксированного размера). Использовать функцию копирования структур из предыдущего пункта (у нас **CopyStudent**). Продемонстрировать использование этой функции, распечатав две структурные переменные до и после обмена.

```
// Взаимный обмен структур
void SwapStudent(Student * S1 ,Student * S2)
{
    Student Stemp;
    CopyStudent ( &Stemp , S1 );
    CopyStudent ( S1 , S2 );
    CopyStudent ( S2 , &Stemp );};

//Прототипы
void CopyStudent ( Student * Source ,Student * Dest );
void SwapStudent(Student * S1 ,Student * S2);
//Вызовы функции
// 5.7, 5.8/ 5.9
printf("ДО SWAP:\n");
StudentPrint(S1);
//
StudentPrint(S2);
SwapStudent( &S1 , &S2);
printf("ПОСЛЕ SWAP:\n");
StudentPrint(S1);
//
StudentPrint(S2);
Результат:
ДО SWAP:
Фамилия - Иванов2  Имя - Петр2  Курс - 3  Стипендия - 10000.00 р.
Фамилия - Петров  Имя - Иван  Курс - 1  Стипендия - 2000.00 р.
ПОСЛЕ SWAP:
Фамилия - Петров  Имя - Иван  Курс - 1  Стипендия - 2000.00 р.
Фамилия - Иванов2  Имя - Петр2  Курс - 3  Стипендия - 10000.00 р.
```


[illegible]

Красным выделены случайные значения курса и стипендии. Значения (Фамилия и имя) заносятся фиксированными ("Фамилия ", " Имя ") и не генерируются.
MMMMMMMM...“”: это неопределенные данные.

5.10. 5.10 Функция вычисления интегрированных данных по массиву структур

Создать и проверить функцию вычисления среднего параметра для массива своих структур (например средней оценки на экзамене в группе). Массив структур передается параметром указателем.

5.11. 5.11 Функция поиска одинаковых данных в двух массивах структур

Создать и проверить функцию вычисления сравнения двух массивов структур с определением количества совпадающих структурных переменных и их распечатки (например одинаковых студентов в двух массивах). Массивы структур передается параметров указателями.

6. 7. Контролируемые требования.

1. **Создание проекта из 2- модулей** [5.1](#)
2. **Описание своей структуры** [5.2](#)
3. **Функция Распечатки структуры** [5.3](#)
4. **Функция Распечатки структуры через указатель** [5.4](#)
5. **Инифиализация массива структур. И функция Распечатки массива структур** [5.5](#)
6. **Функция копирования структур.** [5.6](#)

7. **Функция обмена структур, через указатель 5.7**
8. **Заполнение массива структур 5.9**
9. **Функция вычисления интегральных по массиву структур 5.10.**
10. **Функция поиска одинаковых структур по двум массивам структур 5.11.**

6. Варианты заданий для студентов СУЦ.

Варианты заданий приведены ниже. Номер варианта должен соответствовать номеру студента в групповом журнале.

№ п/п	Структура ДЗ 5 полей, 3 первых из них числовые	Перечисления	Размер массива структур	Диапазон целых	Поле для сортиров ки (числовое д.т.)	Диапазон вещественных (д.т.)
1.	Кафедра	Дни недели	10	0 - 5	1-е (В)	10.0 – 1000.0
2.	Книга	Месяцы	15	1-10	2-е (У)	1.0 – 100.0
3.	Файл	Цвета (7)	20	1-4	3-е (В)	0.0 – 20.0
4.	Автомобиль	Времена года	10	5 - 10	1-е (У)	10.0 – 50.0
5.	Компьютер	Дни недели	15	1-2	2-е (В)	10.0 – 1000.0
6.	Группа	Месяцы	20	0 - 5	3-е (У)	1.0 – 100.0
7.	Человек	Цвета (7)	10	1-10	1-е (В)	0.0 – 20.0
8.	Книжный стел- лаж	Времена года	15	1-4	2-е (У)	10.0 – 50.0
9.	Дом	Дни недели	20	5 - 10	3-е (В)	0.0 – 20.0
10.	Преподаватель	Месяцы	30	1-2	3-е (У)	10.0 – 50.0
11.	Картотека	Месяцы	30	1-2	3-е (В)	10.0 – 50.0
12.	Дисциплина курса	Вид фигуры	40	20-30	3-е (В)	10.0 – 50.0
13.	Студент	Месяцы	30	1-2	1-е (В)	10.0 – 50.0

7. 7. Дополнительные требования для студентов СУЦ (д.т.).

Для продвинутых студентов, по желанию, можно построить программу с дополнительными требованиями. Дополнительные требования выполняются в дополнение основным требованиям ЛР.

7.1. 7.1 Д.Т. Сортировка массива по целому полю.

Описать, инициализировать массив своих структурных переменных. Отсортировать массив по полю варианта (1-е, 2-е или 3-е) по номеру в структуре. Направление сортировки указано в таблице: (В) - возрастание или (У) - убывание.

7.2. 7.2 Д.Т. Двумерный массив структур, инициализация и распечатка

Описать двумерный массив структур (два измерения массива) для своего варианта. Инициализировать его при описании и распечатать в цикле, оформив вывод оригинально.

7.3. 7.3 Д.Т. Заполнение данных случайными числами в массиве структур

Придумать осмысленную структуру, которая может содержать случайные числовые данные (см. варианты – д.т.). Описать массив этих структур. Заполнить динамический массив структур псевдослучайными числами. Распечатать массив с помощью специальной функции.

7.4. 7.4 Д.Т. Поиск экстремума в массиве структур, его номера и распечатка

Создать функцию поиска максимума/минимального значения в массиве структур, по строковому параметру. Функция возвращает номер максимума/минимума и указатель на максимальную/минимальную структуру. После выхода из функции минимум и номер распечатать.

7.5. 7.5 Д.Т. Сортировка массива структур своего варианта ДЗ по символьному параметру

Описать массив структур для своего варианта ДЗ. Создать функцию сортировки массива структур, в которой в качестве параметра используется номер поля для сортировки. Для выбора поля использовать переключатель. Распечатать отсортированный массив своими функциями.

7.6. 7.6 Д.Т. Сортировка массива указателей на структуры

Описать и заполнить массив указателей на структуры своего варианта ДЗ. Выполнить сортировку данного массива любому по числовому параметру. Результат распечатать в цикле.

7.7. 7.8 Д.Т. Функция для создания и заполнения динамического массива структур

Создать функцию для динамического создания и случайного заполнения массива структур своего варианта ДЗ. Число элементов задается параметром. Хотя бы одно из полей должно быть динамической строкой. Динамическая память для нее выделяется в функции. Возврат функции – указатель на массив нужного типа. Задать формальные параметры (диапазон чисел) для случайного заполнения. Результат выделения распечатать. Для строк память должна динамически выделяться.

7.8. Д.Т. Функция для корректного удаления динамического массива структур ДЗ

Создать функцию (**FreeStruct**) для корректного удаления динамического массива структур с динамическими строками. Удаляются и строки и массив. Удаление в функции производится в цикле.

7.9. 7.9 Д.Т. Swap для структуры с динамическими строками

Создать и проверить функцию для своей структуры, в которой выделено поле указатель на динамические строки. При выполнении функции учитывается то, что строки в структуре могут иметь разную длину.

ОП ГУИМЦ 2024 ЛР№6

7.10. 7.10 Д.Т. Функция ввода структуры

Создать функцию для ввода одной структуры.

7.11. 7.11 Д.Т. Функция изменения структуры

Создать функцию для изменения полей одной структуры. Новые значения полей задаются параметрами. Все строки статические.

7.12. 7.12 Д.Т. Сортировка массива структур

Инициализация ручная. Сортировка по одному из полей, которое задано вариантом (номер поля). Использовать для этого функцию **Swap для динамических строк**. На базе своего варианта ДЗ сортировка выполняется по числовому параметру. Распечатать массив своей функцией.

Примечание: Для того, чтобы ЛР была зачтена с выполнением дополнительных требований достаточно выполнить любые четыре из заданий этого раздела.

8. 8. Демонстрация, защита ЛР и отчет по ЛР.

После выполнения всех необходимых шагов по ЛР, работающую программу нужно продемонстрировать преподавателю, проводящему ЛР, о чем он в журнале делает отметку. Далее студент на основе шаблона и примера оформляет отчет по ЛР. После оформления отчета, который может быть представлен преподавателю в электронном виде, выполняется защита ЛР. Студент дает ответы на вопросы по отчету и на контрольные вопросы приведенные ниже. ЛР считается полностью зачтенной, если выполнены все перечисленные требования и действия: демонстрация, отчет и защита ЛР.

9. 9. Контрольные вопросы по ЛР.

1. Каковы основные подходы для формирования обозримого способа хранения данных?
2. Что такое структура данных в языке СИ?
3. В чем массивы и структуры отличаются?
4. Как описывается шаблон структуры, что такое поля структуры?
5. Как описываются и используются переменные структурного типа?
6. Могут ли в структуре использованные другие структурные переменные?
7. Можно ли в структуре описать переменную своего же типа? Указатель своего типа?
8. Как выполнить инициализацию структуры при ее описании?
9. Что такое вложенные структуры?
10. Как описать указатель на структуру?
11. Как работать с полями структуры через указатель?
12. Что такое многоуровневая квалификация и как она возникает?
13. Как создать динамическую структуру, динамическую строку в структуре?
14. Как можно передать структурную переменную в функцию? Массив структур?
15. Как описать массив структур и работать с полями его элементов?
16. Как структуры располагаются в ОП, в каких случаях можно выиграть в затратах ОП?
17. Для каких случаев можно использовать динамические структуры?
18. Для чего в структурах могут использоваться ссылки на себя?
19. Что такое перечисление, как оно используется в программах и для чего?
20. Что такое объединение, для чего они используются?

21. Какие элементы блок схем вы знаете?

10. 10. Литература.

Основная литература

1. Список литературы, доступные книги и необходимые пособия для ЛР ОП размещены на сайте www.sergebolshakov.ru на страничке “2-й к СУЦ”. Пароль для доступа можно взять у преподавателя или старосты группы.
2. Керниган Б., Ритчи Д. К Язык программирования С, 2-е издание: Пер. с англ. – М. : Издательский дом “Вильямс”, 2009. – 304с.: ил. – Пар. Тит. англ.
3. Касюк, С.Т. Курс программирования на языке Си: конспект лекций/С.Т. Касюк. — Челябинск: Издательский центр ЮУрГУ, 2010. — 175 с.
4. MSDN Library for Visual Studio 2005 (Microsoft Document Explorer – входит в состав дистрибутива VS. Нужно обязательно развернуть при установке VS VS или настроить доступ через Интернет.)
5. Фридланд А.Я. Информатика и компьютерные технологии: Основные термины: Толк.слов.: Более 1000 базовых понятий и терминов. – 3-е изд., испр. и доп./ А.Я Фридланд, Л.С. Хананова, И.А. Фридланд – М.:ООО «Издательство Астрель»: ООО «Издательство АСТ», 2003. - 272 с.

Дополнительная литература

6. Общее методическое пособие по курсу для выполнения ЛР и ДЗ (см. на сайте 1-й курс www.sergebolshakov.ru) – см. кнопку в конце каждого раздела сайта!!!
7. Другие методические материалы по дисциплине с сайта www.sergebolshakov.ru.
8. Керниган Б., Ритчи Д. К36 Язык программирования Си.\Пер. с англ., 3-е изд., испр. - СПб.: "Невский Диалект", 2001. - 352 с.: ил.
9. Конспекты лекций по дисциплине “Основы программирования”.
10. Подбельский В.В. Язык Си++: Учебное пособие. – М.: Финансы и статистика, 2003.
11. 5. Подбельский В.В. Стандартный СИ++: Учебное пособие. – М.: Финансы и статистика, 2008.
12. Г. Шилдт “С++ Базовый курс”: Пер. с англ.- М., Издательский дом “Вильямс”, 2011 г. – 672с
13. Г. Шилдт “С++ Руководство для начинающих” : Пер. с англ. - М., Издательский дом “Вильямс”, 2005 г. – 672с
14. Г. Шилдт “Полный справочник по С++”: Пер. с англ.- М., Издательский дом “Вильямс”, 2006 г. – 800с
15. Бьерн Страуструп "Язык программирования С++"- М., Бином, 2010 г.

Новые темы для структур

Копирование структур (присваивание структур)

Структуры для стр. данных (списки, точки, прямоугольники и т.д.)

Массивы структур

Массивы в структурах

Передача структуры в функцию

Передача указателя на структуры в функцию

Функция возвращает структуру

Битовые поля в структурах

Вложенные и локальные структуры