Основы С. Интерактивный курс. Extended

Возможности:
- Работа с указателями на физические ячейки
- Компилируемость и строгая типизация.

Нужен для:
- Решения задач на уровне аппаратуры и ОС.

Для начала установим компилятор и среду разработки.
Ubuntu. Скачаем eclipse.
установим пакет окружения: apt install default-jre

Win10. Скачаем Java runtime enviroment с сайта oracle.com.
компилятор скачаем с MinGW.org.
В процессе установки выделить из списка все „GNU C compiler"

Для VS. Выберем новый проект console C++ project.
создадим пустое приложение.
создадим класс, изменив расширение на .c.
удалить все лишнее из файлов.

Компилятор GCC.
настроим путь: Переменные среды PATH.
C:\MinGW\bin
Теперь в блокноте можно писать программы указав
расширение bat. Нужно вписать в файл:
@echo off
gcc C:\папка\файл.c -o C:\папка\файл.exe
C:\папка\файл.exe

Базовые понятия

Шаблоны программы:
комментарии однострочные // и многострочные /* */
директивы препроцессора это команды, выполняющиеся до компиляции. Подключаются внешние файлы и абсолютные значения
#include <stdio.h> - библиотека ввода/вывода.
int main - точка вход в программу.
return 0; - сообщение ОС, что программа отработала правильно
Все операторы должны заканчиваться ;

Вывод в консоль: printf( ); Используются
- экранированные последовательности: \n, \t, \\, \0
- запоминатели %d %s %c %f %p.
  форма записи: %.2f, %5d, %%

Переменные это именованные контейнеры знаковые и беззнаковые.
Нет булевой переменной, есть int true=1;
int false=0;
char хранит число, которое интерпретируется как символ.
Int = 4 байта
Char знаковый, но используются только положительные.
однобайтный
float может быть 4 байта.
double = long float = 8 байт.
Редко используется short = 2 байта.
Знаковые числа могут быть отрицательными
можно сделать беззнаковыми и наоборот: signed/unsigned.

У каждой переменной есть адрес в памяти. Его можно использовать. переменная: %d - a;
её адрес: %p - &a;
Пользовательский ввод scanf( );
указываем запоминатель и адрес переменной
Стандартная арифметика: + - * /
& | ! ^ << >>
Присвоение = либо += -= *= /=
Деление происходит только нацело.
Получим остаток от деления var %4;
Инкремент и декремент ++; --;

Условия и булева алгебра

```
if ( ){
} else {
}
```

```
if ( ){
} else if ( ){
} else {
}
```

else if может быть сколько угодно.

```
if ( ){
  if ( ){
  }}else{
  }
```

Вложений также может быть сколько угодно.

Тернарный оператор: a=(a>b)?a:0;
В зависимости от условия выполняется первое или второе.

Условия выполняются, когда значение в скобках равно 1 и не выполняются, когда значение равно 0.

Операторы в скобках: >, <, >=, <=, ==
&&, ||, !, !=, ^(XOR)

Смежные операторы: if(X>=0 && X<=10)
if((X>=0) && (X<=10)) лучше отделять скобками

Все что заключено в {} (блок кода), заключено в область видимости и существует только в ней.

Циклы

Базовый цикл: while(){}
Цикл с предусловием. while(a>b){ a++;}

Цикл с постусловием: do{
}while();
одна итерация обязательно пройдет. Затем проверит условие на выполнение еще одной итерации.

Домашнее задание:
1) Написать формулу расчета параметров в $U_o = U_i * (R1/(R2+R1))$
$U_i = U_o * (R_1 + R2)/R_1$;
$R_1 = R2 * U_o / (U_i - U_o)$;
$R_2 = (U_i - U_o) * R_1 / U_o$;
2) Проверить введенное число на соответствие диапазону:
```
const int min=0;
const int max=100;
int in;
scanf("%d",&in);
printf("...",in,((in>=min)&&(in<=max))?" ":"не",min,max);
```
3) Посчитать среднее арифметическое введенных чисел.
```
int n,i=0; float mean;
do{ printf("...:%d",i+1);
    scanf("%d",&in);
    mean+=in; i++;
}while(i<n); printf("...",mean/i);
```

4) Нарисовать ёлочку из знаков ^.

```
int str;
scanf("%d", &str);
for(i=0; i<str; i++){
    for(n=1; n<str-i; n++){
        printf(" ");}
    for(n=(str-i+2); n<=str; n++){
        printf("^");}
    printf("\n");}
```

Функции

тип название(аргументы и их тип){
    код
return 0;}

Если ничего возвращать не нужно, то тип будет void, код в теле не нужен.

Функция - обособленная часть программы, которую можно вызывать в любом месте программы.

Внутри функции могут быть другие функции.

Аргументы создаются при вызове функции и существуют только внутри нее.

Пример:

```
int sum(int x, int y){
    int result = x+y;
    Return result;}
int main(){
    sum(50,60);
```

Передавать в качестве аргументов можно переменные и другие функции.

```
if (d == 2) return 1; else return 0;
  return (d==2); тоже самое.
```

Если функция находится после main, то нужно создать ее прототип: int sum (int x, int y); - прототип
int main ( ) { }

Часто функции находятся в другом файле.
Для их использования нужно подключить заголовочный файл функции.

#include ... <> - компилятор ищет в своей папке
" " - ищет в папке проекта.
#include "header.h"

Указатели

У всех переменных и констант есть уникальное имя и адрес.
Доступ к значению можно получить по имени и адресу.
Новая переменная - указатель, хранит адрес переменной.

int *p; - объявление указателя.
p = &a; - присвоение адреса переменной.
Вывод указателя %p.
printf("...", *p); - получить доступ к переменной через указатель.
Таким образом можно изменять значения переменных:
int a=50;
p=&a;
*p= 70; // a будет равно 70
Так же можно передавать ссылки в функции.
Это полезно, когда нужно возвращать несколько значений,
так же функция не будет создавать внутри себя новые переменные, а будет использовать существующие.

int swap(int *x, int *y){ // передача
*x= ...; // и использование
*y= ...;} // ссылок.
swap(&f, &S) - вызов функции

Массивы

Кроме #define еще существуют и другие.
Полезная директива #define ... (создает константы)
позволяет заменять значения во всей программе.
#define ARR 50 аналогична const int ARR=50;
статический массив - фиксированного размера.
int arr[ARR];
Заполнение: arr[0]=10; arr[5]={0,1,2,3,4};
arr[1]=20;
//...
Проверим генератор чисел:
#include <stdlib.h> // библиотеки
#include <time.h> // работы с генератором
#define NUM 100000

```
int main(){
    srand(time(NULL));
int freq[ARR]={0};
int a,i;
   for(i=0;i<NUM;i++){
    a=rand()%ARR;
    freq[a]++;}
    for(i=0;i<ARR;i++){
      printf("...",i,freq[i],freq[i]/NUM*100);}
```

Массив это ссылочный тип данных, то есть в идентификаторе массива хранится ссылка на первый элемент. Доступ к другим элементам можно получить сдвигом.

```
arr[0]=20; // смещение от начала 0, элемент равен 20
arr[1]=50; // смещение на 1, элемент равен 50
```

Нужно следить за размером массива.

```
While(i<ARR){
        scanf("%d",arr+i); // другой вариант ввода.
        i++;}
```

Доступ к элементу массива: res+=*(arr+i);

идентификатор массива не просто указатель. Он указывает на первый элемент массива и сам по себе хранится в массиве. Значит что не нужно использовать оператор взятия адреса.

```
Void Arr(int *arr, int i){
   res+=*(arr+i);}
int main(){
   Arr(arr,Arr);
```

массив и его длина.

Домашнее задание:

1) Решение квадратного уравнения.

```
int cSE(float a, float b, float c, float *x1, float *x2){
    double D=b*b-4*c*a;
        if(a==0){*x1=-c/b;
                 Return 0;}
    else if(D>0){ *x1=(-b±sqrt(D))/(2*a);
                 Return 1;}
    else if(D==0){*x1=-b/(2*a);}
     else return -1;}
```

2) Удвоить нечетные элементы массива.

```
int odd (int *arr, int L){
    int i, ch=0;
    for (i=0; i<L; i++){
        if (arr[i]%2 != 0){arr[i] *=2;
                           ch=1;}
    return ch;}
```

3) Перевести int в short массив.

```
void int2short (unsigned int *a, int l){
    int i;
unsigned short *sh=a;
        for (i=0; i<l*2; i++)
            printf ("%d", *(sh+i));}
int main (){
 float a, b, c, x1, x2;
 int res;
    res=cSE(a,b,c, &x1, &x2);
 int size, i;
 int ARR[size];
        if (odd (ARR, size))
            for(i=0; i<size; i++)
                printf("%d", *(ARR+i));
 unsigned int arr[size];
        int2short (arr, size);
 return 0;}
```

Строки

Строкового типа данных в С нет. Строка это всегда сложный тип данных.

Строка это набор символов. Можно описать как указатель или как массив.

```
char str[256] = "Hello";
char *str1 = "Hello";
```

Вывод строки: printf("%s", string);

В массиве можно изменять элементы:

```
str[5] = 'X';
```

С указателем на тип char так сделать не получится, но его

Можно вернуть из функции.
```
Char *hello (){
    return "Hello";}
```
Функции для работы со строками.
puts() = printf("%s\n",...);
gets() = scanf("%s",...);
Объединение строк: strcat(...,...);
Функции возвращают функционированную строку. Передадим указатель в функцию.
скопировать строку: strcpy(...,...);
Сравнить строки: strcmp(...,...);
Функции хранятся в библиотеке <string.h>
Функции для работы с символами <ctype.h>

isalpha()	isupper()	toupper() переводит в...
isdigit()	islower()	tolower()
isspace()		

Возвращает единицу если...
Перевод цифр в строке в числа: atoi(), atof().

Структуры данных

Простая дробь:
```
struct frac {
    int i;
    int d;
    int div;};
```
Объявление структуры и полей в ней, выделение памяти.
Создадим тип данных: typedef struct Frac frac;
объявим: frac f1, f2, res; - три структуры.
f1.i = -1; - инициализация
f2.d = 1; полей
f1.div = 1;
Во внутрь функции структуру передаем по значению либо ссылке.
```
void print(frac f){
    printf("%d %d", f.div, f.div);}

Void mult(frac f1, frae f2, frae *res){
    res->div = f1.d * f2.d;
    res->div = f1.i * f2.i;}
    Mult(f1, f2, &res); - вызов функции
```

Доступ к полям указателей внутри структуры осуществляется через стрелку.

Файловая система

Бывает текстовая и бинарная. Это структура.
Указатель на структуру: FILE *f;
Присваим реальный файл: f=fopen("file.txt" "w");
Основные типы открытия файла: w, r, a, wb, rb, ab.
Полный адрес файла: C://../../..txt
В примере адрес будет писать в папке проекта.
Если будет ошибка при открытии файла, то в указатель запишет NULL
Проверка: if (f==NULL) return 1;
Запись в файл: fprintf(f, "Hello %s", "we did it!\n");
файл нужно закрыть: fclose(f);
Считывание из файла: char word[256];
fscanf(f, "%s", &word);
Работает до пробела. Прочитаем все.
while(!feof(f)){
fscanf(f,"%s", &word);}

Динамическое выделение памяти

void malloc(); - Возвращает указатель на неразмеченную область памяти. Чтобы разметить, нужно привести тип.
int *area = (int*) malloc(123); - 123 байта, размеченные под 4х байтный int.
Это почти одно и тоже, что int array[10]
Для разметки так же помогает функция sizeof(тип);
Возвращает размер типа данных.
Чтобы очистить память под новую разметку. Используют функцию: int *area = (int*) calloc(size, sizeof(int));
После работы память нужно освободить: free(area);

Для динамического изменения памяти используют realloc();
area = realloc(area, sizeof(int)*(newsize+10));
Таким образом можно добавлять или убирать память.
Память чаще всего выделяется блоками, степенью двойки.

Домашняя работа.

1) Перевести десятичное число в двоичное строковой функцией.

```
void dec2bin (int dec, char *res) {
   int i=0, shift=dec; char bin[64];
   while (shift !=0){ shift =shift >>1;
                      i++; }

   bin [i]='\0';
    while(dec >0){
       bin [--i ] = (dec %2)? '1':'0';
       dec /= 2; }
       strcat(res, bin);
```

2)
```
typedef struct rectangle{
   int h; int w; int ar; in per;
   } Rec;
void ar_per_calc (Rec *r){
      r-> ar = r->h * r->w;
      r-> per =(r->h + r->w)*2; }
```

3)
```
int *p_alloc(int l){
   Return calloc(l, sizeof(int));}
```

4)
```
typedef struct Point {
     int x; int y;
   } point;
 typedef struct Line{
   float length; point p1; point p2;
   } line;
line calc_line(int x1, int x2, int y1, int y2){
     line L;
     L.p1.x = x1;
     L.p1.y = y1;
     L.p2.x = x2;
     L.p2.y = y2;
     L.length= sqrt(pow(x2-x1,2)+pow(y2-y1,2));
       return L;}

int main () {
```

```
dec2bin (num, binary);
    Rec rec;
    ar_per_calc (&rec);
    int *arr = p_alloc (size);
Line L = calc_line (x1, x2, y1, y2);
          Printf ("%d", L.p1.x);
return 0; }
```

Основы языка С

Возможности: • Работа с указателями на ячейки ОЗУ.
• Процедурный код с мощной типизацией
Нужен для задач уровня аппаратуры и ОС.
Скачаем и установим eclipse-IDE на Ubuntu.
Установим пакет: apt install default-jre

Создадим проект С.
добавим пустой С файл и напишем:

```
#include <stdio.h>      стандартная библиотека
int main(){
PRintf("Hello world");
Return 0;
}
```

Установка на Win10.
скачаем и установим JRE с сайта oracle.com
Компилятор скачаем с sourceforge.net - MinGw
Можно запустить Visual Studio, но не нужно
Также можно сделать связку gcc+Notepad++, прописав путь к исполняемому файлу.

Базовые понятия

Комментарии: однострочные //, многострочные /*...*/
Директивы препроцессора: выполняются до компиляции. Обычно определяют библиотеки, внешние файлы и параметры.

```
#include<stdio.h> - стандартный ввод/вывод
int main (int argc, const char* argv[])
Return 0;        main - точка входа в программу.
                 Return 0; - знак того, что программа отработала правильно.
                 другие возвратные коды говорят об ошибке
Все операторы должны заканчиваться ;
```

Общение происходит с помощью консоли.

Printf() - функция для вывода в консоль (форм. вывод)
Типы вывода: экранирование и заполнение
1) \n \t \\ \0 = новая строка, табуляция, \, конец строки

2) %d %5 %f, %C ... для переменных типа.
%.2f - число знаков после запятой
%05d - пять цифр с лидирующими нулями.
%% - знак процента.

Переменные: делятся на целочисленные, символьные, указатели, с плавающей точкой.
Знаковые и беззнаковые, т.е. старший бит отводится под знак.
int n = -56;
char s = 'A';
float r = 5.345
Булевого типа в С нет. Вместо него используют int true = 1;
int false = 0;
char - хранится число которое интерпретируется как символ
int - 4х байтное значение
char - однобайтный знаковый
unsigned char - беззнаковый

Числа с плавающей точкой представляются:
4 байта - float
8 байтный - double. (long float)

Переменная имеет имя и адрес в памяти
взять адрес: &a.
посмотреть адрес %p.
Ввод чисел с клавиатуры: scanf("%..", &..);
тип → ← адрес переменной
Внутри printf можно использовать арифметику, функции и комментарии.
Поддерживает операции: + - * / & | ! ^ << >>
сокращенная запись: V = V+1; <=> V += 1;
Деление переменных типа int отбрасывает дробную часть.
Его можно увидеть с помощью модуля % чисел
V%4 - остаток от деления V на 4.
инкремент и декремент: ++ / --

Условный оператор

```
if ( ){
} else {
}
```

```
if ( ) {
} else if ( ) {
} else {
}
```

Операторов else if может быть сколько угодно, кроме if/else

```
Printf("Do you want me to salute you? y/n");
    Scanf("%c", &answer);
    if(answer == 'y') {
            printf("Hello, user!");
    } else if(answer == 'n') {
            printf("I didn't want to salute you!");
    } else {
            printf("I cant understand you!");
    }
```

Тернарный оператор:

```
c = (a>b)? b: a;   ~   if(a>b) c=b;
                       else c=a;
```

Ввод булевых значений:

```
printf("%s", (1>0)? "true": "false");
```

Арифметические сравнения: ==, !=, <, >, >=, <=

Логические операторы: &&, ||, !, ^(xor), <<, >>, сдвиг a=a<<3;

Условия могут быть вложены. Ограничений нет.

```
if(1){
    if(1){
        ...
    }}
```

Сложные условия:

```
int x=7;
if((x>=0)&&(x<=10)){}
```

Фигурные скобки - операторные скобки.

Пространство между этими скобками называется областью видимости, а данные - блоком кода. Данные внутри блока кода видны только в области видимости.

Циклы

Базовый цикл - while

```
while(a<b){
}
```

Цикл с предусловием. Каждый проход цикла называется итерацией

Цикл с постусловием:

```
do{...} while();
```

Выполняет одну итерацию и потом решает, нужно ли повторять процедуру.

Оператор continue позволяет пропустить часть итерации и вернуться в начало цикла.
оператор break принудительно выходит из цикла.

Dev-C++ (русские буквы в консоли) #include <locale.h>
int main(){
setlocale(LC_ALL, "Rus");

Цикл с предусловием FOR.
for(i=0; i<sig; i++){}
начальное значение, условие, приращение.
Применяется тогда, когда кол-во циклов известно.

Оператор множественного выбора switch.
switch(){ Проверка условия и
case 1: break; Выполнение соответствующего
case 2: break; блока case, либо default.
:
default: ..
}

Домашнее задание:
1) $U_0 = U_1 * (R_1/(R_2+R_1))$;
$U_1 = U_0 * (R_1+R_2)/R_1$;
$R_1 = R_2 * U_0/(U_1-U_0)$;
$R_2 = (U_1-U_0) * R_1/U_0$;
2) const int min=0;
const int max=100;
int i;
Printf("Введите число:");
scanf("%d", &i);
Printf("Ваше число'%d' %S диапазоне (%d - %d)\n",
i, (i>=min && i<=max)? "В" : "Не в", min, max);
3) int i=0, c=0;
float m=0;
while (i<10){
printf("Write a Number:");
scanf("%", &c);
m=m+c;
i++; }

printf("Mean: %.2f", m/i);

4)
```
int s;
printf("Write a number: ");
scanf("%d", &s);
int i, j;
for(i=0; i<s; i++){
    for(j=1; j<s-i; j++) printf(" ");
    for(j=s-i*2; j<=s; j++) print("^");
printf("\n"); }
```

Функции в Си

Обособленная часть кода, выполняющаяся несколько раз.
Обязательно должно быть:
- тип возвращаемого значения
- имя функции
- список аргументов
- тело функции

void - тип функции, которая ничего не возвращает.
В аргументы можно ничего не передавать, либо константы с переменными.
В теле функции могут быть другие функции.
int main() {}
Функции с нефиксированным числом аргументов типа printf();
функции могут быть не в одном файле с main.
Можно подключить файл функции: #include "header.h"
Если функция идет после main, нужно описать ее прототип:
int main(int ...); - без тела функции.
Заголовочные файлы - header, должны храниться в папке с проектом. Очень удобны при модульной разработке.

Указатели

Можно получить доступ к переменной не только по имени, но и по адресу. Адрес может храниться в специальной переменной - указателе.
Объявление указателя: int *p;
присвоение адреса: p = &a;
получить доступ к переменной через указатель: *p.
Изменить значение переменной через указатель: *p = 70;

Таким образом можно писать функции не создавая новые переменные, а передавая ссылки на существующие переменные

```
int swap(int x, int y){} - передача параметров по значению
                           создает локальные переменные.
int swap(int *x, int *y){} - передача по ссылке.
   swap (&var1, &var2)  будет ссылаться на существующие
                        переменные.
void swap(int *X, int *y){
      *X = 15;
      *y = *X + *y;}
int main(){
   swap (&var1, &var2)
   printf("%d %d", var1, var2);
```

Массивы

Директива препроцессора.

#define - позволяет заменить или описать константу. дает указание компилятору.

Можно заменить целые вызовы функций.

массив: int array[1][0] = 10; объявление

int array2[5] = {0,1,2,3,4} массива

Проверим генератор случайные числа.

```
#include <stdlib.h>
#include <time.h>
srand(time(NULL));
int freq[ARRAY] = {0};
   for(i=0; i<numam; i++){
      a = rand()%ARRAY;
      freq[a]++;}
```

Масив это ссылочный тип данных, то есть в идентификаторе хранится ссылка на первый элемент. Доступ к другим элементам осуществляется смещением относительно первого элемента.

Выход за пределы массива никак не отслеживается.

Идентификатор массива по сути указатель.

Заполним массив:

```
while (i<ARR){
    scanf("%d", arr+i);
    i++;}
```

Арифметика: `res = res + *(arr+i);`

Передача в функцию:

```
float average (int *arr){ }
    average (arr);
```

← массив.

Мы можем передавать идентиф. массива без использования оператора взятия адреса, так как сам по себе является указателем на свое начало.

Вызов функции из файла

Создадим Source file.

Опишем в нем функцию.

Создадим прототип функции внутри основного файла, либо подключим его через #include.

Теперь можно вызвать функцию.

Многомерные массивы

Массив содержит данные.

Двумерный массив это массив из массивов с данными и т.д.

Объявление:

```
int twoDim[5][5];
int threeDim[3][3][3];
```

Сформируем таблицу Пифагора (умножения)

```
int table[10][10];
int r, c;
for(r=0; r<10; r++){
    for(c=0; c<10; c++)
        table[r][c]=(r+1)*(c+1);}
for (r=0; r<10; r++){
    for(c=0; c<10; c++)
        printf("%3d", table[r][c]);}
```

Массив может быть любого типа, в том числе и указателем.

Можно записать массив из строк.

`char* Arr[3] = {"Hello", "C", "World"}` - это указатели на строки.

Домашнее задание:

1) `int cse(float *x1, float *x2){`

```
    *X1 = (-b + sqrt(D)) / (2*a);
    *X2 = (-b - sqrt(D)) / (2*a); }
int main(){
  float X1, X2;
   cse(1, 2, -3, &X1, &X2); }
```

2)
```
int mac(int *AR, int leng){
  int n, r=0;
      for(n=0; n<leng; n++){
          if(AR[n]%2!=0){
              AR[n] *=2;
              r=1; }}
          Return r; }
int main(){
  int arr[leng];
  int i, res;
      res = Mac(arr, leng);
        for(i=0; i<leng; i++){
      Printf("Новый массив: %d\n", *(arr+i));
```

3)
```
void IntToShort(unsigned int *a, int len){
    int i;
    unsigned short *out = a;
      for(i=0; i<len*2; i++) Printf("%d", *(out+i)); }
```

Прочитаются по области памяти в 4 байта, но по 2 байта.

Строки

Типа string в Си нет.

Строка это всегда ссылочный тип данных.

Строки можно описать как указатель, либо как char массив.

Объявим строку: `char str[25] = "this is a str";`

Изменим элемент: `string[5] = 'X';`

`Printf("%s\n", string);`

Подобная схема не работает с типом `char *`, но строку можно вернуть из функции:
```
char *Hello(){
    return "Hello"; }
Printf("%s\n", Hello());
```

Существуют специальные функции.

puts() - вывод строк. puts() = Printf("%s")

gets() - ввод строки с консоли. gets() = Scanf("%s",...)

strcat() - склеивает строки в одну.
Из функции можно получить только фиксированную строку.
Мейн должен дать функции указатель.

```
void hel(char *name, char *out){
    char wel[256]="Hello, ";
    strcat(wel, name); - склеим строку и указатель
    strcpy(out, wel); - скопируем строку в указатель на выход.
}
    Char name[256];
    Char res[256];
      gets(name);
      hel(name, res);
      puts(res);
```

Strcmp(.., ..) сравнение строк.

Функции по обработке символов находятся в <stdlib.h>
Например: isalpha(), isspace(), islower(), tolower().
isdigit(), isupper(), toupper(),
Часто используется: atoi(), atof()
перевод символов в строке в числа.

```
Char num[64];
int number = atoi(num);
```

Структуры данных

Структура объединяет в себе несколько примитивов либо ссылок.
Создадим структуру "простая дробь"

```
struct fraction{
    int integer;
    int divisible;
    int divisor; };
```

переменные могут быть неинициализированы,
но компилятор разместит память под них.

Для сокращения записи опишем новый тип данных:

```
typedef struct fraction Fraction;
```

Доступ к переменным структуры осуществляется через точку.
инициализируем переменные: Fraction f1, f2, res;

```
f1.integer = -1;        f1.divisor = 5;
f2.integer = 1;         f2.divisible = 1;
f1.divisible = 1;       f2.divisor = 5;
```

Чтобы не перепутать локальные структуры и указатели на внешние структуры, доступ к полям внутри указателя структуры осуществляется с помощью стрелки.

result->divisible = f1.divisible * f2.divisible;

Файловая система

Чтение и запись в файловую систему.
Файловая система это структура.
Опишем структуру: FILE *f;
f=fopen("file.name.txt", "w"); присвоит указатель на файл.
Если файл не удалось открыть, то в указатель запишется NULL.
fprintf( ) - запись в файл.
Файл после обработки нужно закрыть. fclose(f);
Для считывания из файла нужно:
Открыть для чтения: f=fopen("file.txt" "r");
fscanf(f, "%s", &word); char word[256];
Здесь мы указали сколько нужно считать.
Чтобы прочитать все, нужно использовать функцию feof();

```
while (!feof(f)){
    fscanf(f,"%s", &word);}
```

Она сканирует все данные типа строка до конца файла, затем возвращает ноль.

Динамическое выделение памяти

memory allocation.
malloc( ) - выделение памяти.
Функция выделяет память, но не размечает её.
Разметим самостоятельно.

```
int var;
int *area = (int*) malloc(123);
```

Создали указатель на область памяти из 123 байт, размеченных под int ячейки, и каждой ячейке присвоить свой адрес.
sizeof( ) - возвращает размер переменной.
(int*) malloc (sizeof(int)*10); - 10 ячеек по 4 байта.
Фактически это одно и тоже: int arr[10];
arr[i] = *(arr+i);
Очистить память: calloc( );
Очищает память и создает новую.
calloc (SIZE, sizeof(int));

Память нужно освобождать: free(area);
Сделать память динамической: area = realloc(area, sizeof(int)*10);
(расширить массив)

Домашнее задание:

1) void decTObin(int dec, char *res){
char out[64]; // временное переменное
int i=0, tmp=dec;
while(tmp!=0){ // считаем
tmp = tmp >>1; // сколько
i++;} // разрядов
out[i] = '\0'; // конец массива.
while(in>0){ // [--i] сдвиг влево
out[--i] = (in%2)?'1':'0'; // перевод dec в дв. с
in/=2;} // делением на два.
strcat(res, out);}

2) typedef struct rec { // структура новый тип
int L, W, P, A;} Rec; // прямоугольник данные.
Void AP(Rec *ap){
ap->A = ap->L * ap->W; // расчет площади и периметра
ap->P = (ap->L + ap->W)*2; // по указателю.
int main(){
Rec rec; // создали новую структуру. Инициализируем.
scanf(... &rec.L); // ввели параметры
scanf(... &rec.W);
AP(&rec); // вызвали функцию
printf(... rec.A, rec.P);}

3) int main(){
const int ARR = 5; // размер массива
int *arr = arrInit(ARR); // указатель на массив
for(i=0; i<ARR; i++) // заполняем
*(arr+i) = i*10;} // целый массив
int *arrInit(int i){ // функция принимает
return calloc(i, sizeof(int));} // размер массива и
// возвращает чистую память под массив arr.

4) Вложенные структуры:

```
typedef struct point{          // структура точка
    int x,y;} Point;           // координаты
typedef struct line{           // структура линия
    float length;              // длина и две точки
    Point p1,p2;} Line;        // начало и конец

Line initLine (int x1, int x2, int y1, int y2){
    Line line;
    Line.p1.x = x1;            // инициализируем точки
    Line.p1.y = y1;
    Line.p2.x = x2;
    Line.p2.y = y2;
    Line.lenght = sqrt(pow(x2-x1,2)+pow(y2-y1,2));
    return line;}
int main (){
    Line l = initLine(5, 4, 8, 9);   // вызываем функцию
    Printf(...l.p1.x, l.p1.y);...}
```

В курсе представлены основные инструменты языка. Для более глубокого изучения стоит прочитать документацию к языку и изучить специальные библиотеки, созданные под определенные задачи.