

Лекція 5

Структури. Визначення власних типів. Робота з файлами



Структура

Сукупність типів об'єднана в один тип:

```
struct Назва {  
    Тип поля1 НазваПоля1;  
    Тип поля2 НазваПоля2;  
    **  
    Тип поляN НазваПоляN;  
};
```

Декларація структури

1. Декларація змінних типу структура (локально або глобально)

```
struct Student // визначаємо структуру з назвою Student
{
    char name [25]; /* Поле 1: Прізвище та ініціали – тип рядок з 25 символів */
    char group[3]; // Поле 2: Група – тип рядок з 3 символів
    int pract_mark; // Поле 3: Бали за практику – тип ціле
    int course_project1; // Поле 4: Бали за перший проект – тип ціле
    int course_project2; // Поле 5: Бали за другий проект – тип ціле
    float additional_mark; /* Поле 6: середній додатковий бал -тип дійсне число */
} st1, st2; // змінні типу Student
```

Декларація структури

// 2 - Point2D - точка (два цілі числа)

```
struct Point2D {
```

```
    int x, y;
```

```
}; //
```

// 3 - VectorND - вектор (натуральне + вказівник на double)

```
struct VectorND {
```

```
    size_t n;  double * v;} VectorND;
```

```
int main(){
```

```
    struct Point2D p1, p2;
```

```
    struct VectorND v1;
```

Ініціалізація структури

```
int main(){  
    struct Credit{ int bals[2][2]; float marks; } cred[10] = { {  
        {{8,8},{8,8}},33, } } ; /* ініціалізація складної структури */  
  
    struct Point2D z = {1, 0}; // звичайна ініціалізація  
  
    struct VectorND v = {.n=5}; /* направлена ініціалізація - з  
        C95 */  
}
```

Присвоєння полів структури

```
struct Point2D z;  z.x =1;   z.y = 1; // доступ до полів
struct VectorND v;
    v.n  = 10; // так само
    v.v = (double*) calloc(v.n, sizeof(*(v.v))); /* виділення пам'яті
та ініціалізація нулями */
    for(int i=0;i < v.n;i++){
        scanf("%lf,", &(v[i].v)); // введення через консоль
    }
***
free(v.v); // не забули звільнити
```

Присвоєння полів структури

```
cred[1].marks = 2.1f; // присвоєння полів
```

```
cred[2].bals[0][1] = 10; // теж присвоєння
```

```
scanf("%s %s %d %d %d %f", st1.name, st1.group,  
    &st1.pract_mark, &st1.course_project1,  
    &st1.course_project2, &st1.additional_mark);
```

```
st2 = st1; // присвоєння структур
```

```
strcpy(st2.name, "Vasya");
```

Визначення типів - typedef

```
typedef unsigned char BYTE // баайт (BYTE) – тип: 0..255
```

```
typedef signed int * IPOINTER // вказівник на signed int
```

```
typedef unsigned long long int ULL // 3 символи замість 24
```

```
typedef unsigned UARRAY[10] // масив теж можна
```

```
int main() {
```

```
    BYTE a, *a1; // a – байт, a1 – вказівник на байт
```

```
    IPOINTER ptr, ptr1[2]; // ptr- вказівник, ptr1 – масив вказівників
```

```
    ULL n; // змінна типу дуже довгого цілого
```

```
    UARRAY arr; // arr – масив з 10 натуральних
```

```
    ***}
```


Визначення типів – typedef для структур

```
typedef struct { // визначили тип Point3D – як структуру
    int x,y,z;
} Point3D;

typedef struct List{ // рекурсивна структура Список Point3D
    Point3D data;
    List* next;
} List;

int main(){
    Point3D t = {0,0,0}; // тепер можна не писати всюди struct
    List lst;
}
```

Приклад створення структур

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

typedef struct Credit {
    int marks[2][2];
    float addit;} Credit;

typedef struct {
    char * name;
    float mark;
    float proj_mark[2];
    float addit;
    Credit exam; } MarkProgram;
```

Функції зі структурами

```
void printCredit(Credit c){ // функція від структури
```

```
    printf("%d %d %d %d",c.marks[0][0], c.marks[0][1],c.marks[1][0],c.marks[1][1]);
```

```
}
```

```
Credit inputCredit(){ // повертаємо структуру
```

```
    Credit c = {0,0,0,0,0.f};
```

```
    printf("Input 4 marks and additional mark");
```

```
    scanf(" %d %d %d %d", &c.marks[0][0], &c.marks[0][1], &c.marks[1][0],  
    &c.marks[1][1]);
```

```
    return c;
```

```
}
```

Функції зі вказівниками на структури

```
void printMark(MarkProgram* m){ // функція від вказівника
    printf("Mark of %s:", m->name);
    printf("%f %f %f", m->mark, (*m).proj_mark[0], m->proj_mark[1]);
    printCredit(m->exam); // 2 типи доступу до полів
}
```

```
int inputMark(MarkProgram* z){
    printf("Input Name"); // виділяємо пам'ять та ініціалізуємо!!!
    const size_t N = 20; char name[N];
    fflush(stdin); fgets(name, N, stdin);
    z->name = (char*) malloc(sizeof(name));
    strcpy(z->name, name);
    printf("Input 3 marks:");
    scanf("%f %f %f", &z->mark, &z->proj_mark[0], &z->proj_mark[1]);
    z->exam = inputCredit();
    return 0; // успішне введення
}
```

Робота зі вказівниками на структури

```
void deleteMark(MarkProgram m){ // фактично деструктор
    free(m.name);
}

int main(){
    MarkProgram* m; int n;
    printf("Number of students");
    scanf("%d ", &n);
    m = (MarkProgram*) malloc(n * sizeof(*m));
    for(int i=0; i<n;i++) { inputMark(&m[i]); }
    for(int i=0; i<n;i++) {
        printMark(&m[i]);
        deleteMark(m[i]);
    }
```

Робота з файлом

Файл – це поіменована сукупність даних, яка зберігається на пристрої

Текстовий файл - текстовий потік даних, що має ім'я та зберігається на пристрої. Текстові файли містять байти, що є кодами алфавітних, цифрових символів та знаків пунктуації (пробілів, табуляцій та символи переходу на новий рядок)

Бінарний файл – це сукупність будь якого типу даних, що знаходиться в пристрої. При цьому для того, щоб зчитувати конкретний файл, потрібно знати які дані, в якому порядку записані на пристрої.

Потоки вводу-виводу - це об'єкти типу FILE, до яких можна отримати доступ і маніпулювати ними лише за допомогою вказівників типу **FILE ***

Схема роботи з файлом

- 0) Програма **створює допоміжну змінну** (скажемо, *f*) типу **вказівник на структуру (FILE*)**, через які вже операційна система отримує доступ до конкретного місця на диску.
- 1) Перш ніж робити з файлом на диску будь-які дії (читати дані з файлу чи писати файл), програма повинна його **відкрити**. Файл, розташований на диску, **зв'язується** зі змінною *f* (тобто до змінної заносяться службові дані для доступу саме до даного файлу).
- 2) Після цього, для **запису** чи **читання** даних з файлу, програміст викликає **спеціальні функції читання та запису**, передаючи їм через *один аргумент* змінну *f* канал доступу до файлу, а через *решту аргументів* – *які саме дані читати чи писати*.
- 3) На закінчення роботи програма повинна **закрити** файл і розірвати зв'язок між змінною *f* та файлом на диску, при цьому **звільняються** ресурси операційної системи, що використовуються для доступу до файлу.

Відкриття файлу

```
FILE* fopen(const char* filename, const char* mode);
```

//З допомогою змінної file1, file2 будемо мати доступ до файлу

```
FILE *file1, *file2, *file3;
```

//Відкриваємо текстовий файл з правами на запис

```
file1 = fopen("C:/c/test.txt", "w+t");
```

```
FILE* f1 = fopen("C:\\test1.txt", "r"); // читання
```

```
file2 = fopen("\\home\\test2.txt", "r+b"); // бінарне читання
```

```
file3 = fopen("\\home\\test2.txt", "a+"); // редагування
```


Режими відкриття файлу

	Опис
r	Читання. Файл повинен існувати.
w	Запис нового файлу. Якщо файл с з таким іменем вже існує, то його зміст буде знищено. Інакше він створиться
a	Запис в кінець файлу. Операції позиціонування (fseek, fsetpos, frewind) ігноруються. Файл створюється, якщо не існував.
r+	Читання та оновлення. Можна як читати, так і записувати. Файл повинен існувати.
w+	Запис і оновлення. Створюється новий файл. Якщо файл с з таким іменем вже існує, то його зміст буде знищено. Можна як читати, так і записувати.
a+	Запис в кінець файлу й оновлення. Операції позиціонування працюють лише для читання, для запису ігноруються. Якщо файлу не існувало, то він створиться.

Обробка відкриття файлу

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
const int ERROR_FILE_OPEN = 0x3; // код помилки відкриття файлу
```

```
int main() {
```

```
    FILE *output = NULL; int number;
```

```
    output = fopen("output.bin", "wb");
```

```
    if (output == NULL) { // перевіряємо коректність виконання
```

```
        printf("Error opening file"); getchar(); // виводимо напис про помилку
```

```
        return ERROR_FILE_OPEN; // та завершуємо виконання з кодом помилки
```

```
    }
```

```
    scanf("%d", &number); // записуємо в файл якщо все ОК
```

```
    fwrite(&number, sizeof(int), 1, output);
```

```
    fclose(output);
```

```
return 0;
```

```
}
```

Обробка помилки відкриття файлу

```
# include <stdio .h>
```

```
int main () {
```

```
    char fileName [ 80 ];
```

```
    FILE *f;
```

```
    do {
```

```
        printf ( " Введіть ім 'я файлу або крапку : " );
```

```
        scanf ( "%s", fileName );
```

```
        if( strcmp ( fileName , "." ) == 0 ) return 0; // вихід з циклу
```

```
        f = fopen ( fileName , "r" );
```

```
    } while ( f == NULL ); // вводимо ім'я поки відкриття не вдалося
```

```
/* далі нормальна обробка файлу */
```

```
...
```

```
fclose ( f );
```

Команди читання з текстового файлу

int fscanf (FILE * stream, const char * format, ...); /*

Читання з текстового файлу stream форматowanego рядка як в scanf() до першого роздільника*/

char * fgets (char * str, int num, FILE * stream); /* Читання з текстового файлу stream рядка з num символів як в gets() */

int fgetc (FILE * stream); /* читання поточного символу (з місця де вказує маркер) як getchar() */

int feof (FILE * stream); // індикатор кінця файлу

```
# include <stdio .h>
```

```
# define LEN 256
```

```
int main () {
```

```
    FILE *f;
```

```
    int m, n;
```

```
    double dt;
```

```
    char s[ LEN ];
```

```
    f = fopen ( " data . txt ", "r");
```

```
    n = fscanf (f , "%d %lf %s", &m, &dt , s);
```

```
    printf ( " Прочитано %d значень :\n", n);
```

```
    printf ( " Ціле %d, дійсне %lf , рядок %s\n", m, dt , s);
```

```
    fclose ( f );
```

```
}
```

Введення/виведення порядково

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
int main() {
```

```
    FILE *file;
```

```
    char buffer[128];
```

```
    file = fopen("C:/c/test.txt", "w"); // відкрили для запису
```

```
    fprintf(file, "Hello, World!"); // записали текст
```

```
    freopen("C:/c/test.txt", "r", file); // перевідкрили для читання
```

```
    fgets(buffer, 127, file); // зчитали текст
```

```
    printf("%s", buffer); // вивели на консоль
```

```
    fclose(file); // не забули закрити файл
```

```
}
```

Команди запису в текстовий файл

int fprintf (**FILE** * stream, **const char** * format, ...);

int fprintf (вказівник_на_файл,рядок форматування, перелік змінних);

Аналог команди printf

int fputs (**const char** * str, **FILE** * stream);

int fputs (рядок_виводу, вказівник_на_файл);

Аналог команди puts

int fputc (**int** character, **FILE** * stream);

int fputc (символ, вказівник_на_файл);

Аналог команди putc

int fprintf (вказівник_на_файл,рядок
форматування, перелік змінних);

```
#include <stdio.h>
```

```
int main(){
```

```
int i,j;  FILE *lds;
```

```
lds=fopen("epa.txt","w"); /*Відкриття файлу на диску для запису. Якщо він не  
існує, то створюється автоматично*/
```

```
for(i=1;i<=10;i++){
```

```
    for(j=1;j<=10;j++){
```

```
        /*Запис даних до файлу: Перший аргумент – вказівник на файл,  
        другий і третій такі ж як і для команди printf */
```

```
        fprintf(lds,"%d%c",i+j-1,((j==10)?'\n':' '));}
```

```
    }
```

```
    fprintf(lds,"\n");}
```

```
fclose(lds); /*Закриття файлу*/
```


**int fprintf (вказівник_на_файл,рядок
форматування, перелік змінних);**

```
#include <stdio.h>
```

```
int main(){
```

```
int i,j; FILE *lds;
```

```
lds=fopen("epa.txt","w"); /*Відкриття файлу на диску для запису. Якщо він не  
існує, то створюється автоматично*/
```

```
for(i=1;i<=10;i++){
```

```
    for(j=1;j<=10;j++){
```

```
        /*Запис даних до файлу: Перший аргумент – вказівник на файл,  
        другий і третій такі ж як і для команди printf */
```

```
        fprintf(lds,"%d%c",i+j-1,((j==10)?'\n':' '));}
```

```
    }
```

```
    fprintf(lds,"\n");}
```

```
fclose(lds); /*Закриття файлу*/
```

Команди роботи з файлами

int fflush (вказівник_на_файл);

int fclose(вказівник_на_файл);

int remove(“ім'я_файлу”);

int rename(“старе_ім'я_файлу”, “нове_ім'я_файлу”);

Запис у бінарний файл

`size_t fwrite (void *p, size_t b, size_t n, FILE *f);`

`void *p` – вказівник на те місце в оперативній пам'яті, де починається послідовність блоків даних, яку треба записати у файл;

`size_t b` – довжина в байтах одного блоку;

`size_t n` – число блоків;

`FILE *f` – вказівник на потік: до якого файлу записати дані.

Результат – кількість записаних даних

Приклад запису у бінарний файл

```
#include <stdio .h>

# define N 5

int main () {

    double w[ N ] = { 2.0 , 1.4142 , 1.1892 , 1.0905 , 1.0443 };

    char fileName [] = "data.dat"; // імя файлу

    FILE * out ;

    int i;

    out = fopen ( fileName , "w" ); // відкрили файл

    for ( i = 0; i < N; ++i ){

        fwrite ( &(amp;w[i]), sizeof(double), 1, out ); // записали з вказівника 8*1 байт в
        out

    }

    fclose ( out );

}
```

Читання з бінарного файлу

`size_t fread (void *p, size_t b, size_t n, FILE *f);`

`void *p` – вказівник на те місце в оперативній пам'яті, де починається послідовність блоків даних, яку треба записати у файл;

`size_t b` – довжина в байтах одного блоку;

`size_t n` – число блоків;

`FILE *f` – вказівник на потік: до якого файлу записати дані.

Результат – кількість зчитаних даних

Приклад читання з файлу

```
# include <stdio .h>

# define N 5

int main () {
    double w[ N ];
    char fileName [] = " data.dat ";
    FILE *inFile;
    int k, i;
    inFile = fopen ( fileName , "w" );
    k = fread ( w, sizeof ( double ), N, inFile ); // зчитали відразу весь масив!!!
    printf ( "З файлу прочитано %d чисел ", k );
    for ( i = 0; i < k; ++i ){
        printf ( "%lf\n", w[i] );
    }
    fclose (inFile );
}
```

```
#include <stdio .h>
```

```
# define N 5
```

```
int main () {
```

```
    double w[ N ];
```

```
    char fileName [] = " data.dat ";
```

```
    FILE *inFile;
```

```
    int k, i=0;
```

```
    inFile = fopen ( fileName , "wb" );
```

```
    while (!feof(inFile)){ // читаємо доки не  
        досягли кінця файлу
```

```
        double x;
```

```
        k = fread ( &x, sizeof ( double ), 1,  
inFile ); // зчитали по одному
```

```
}
```

```
    w[i++] = x;
```

```
    k=i;
```

```
    printf ( "З файлу прочитано %d чисел ", k );
```

```
        for ( i = 0; i < k; ++i ){
```

```
            printf ( "%lf\n", w[i] );
```

```
        }
```

```
    fclose (inFile );
```

```
}
```

Приклад роботи з текстовим файлом

```
// С Програма роботи зі структурою
// Поля id, name and age
#include <stdio.h> // Базове та файлове введення-виведення basic IO, file IO
#include <stdlib.h> // memory work, EXIT_SUCCESS, EXIT_FAILURE
#include <string.h> // string copy

// Опис структури
typedef struct Student {
    char name[10];
    int id;
    char age;
}Student;
```



```
int createTextFile(const char* fname){ // функція запису в текстовий файл
    FILE* f = fopen(fname, "wt"); // відкрили для запису
    if(f==NULL) return EXIT_FAILURE; // якщо невдало – повертаємо error

    for(;;){ // нескінчений цикл для запису в файл

        printf("input id or 0\n"); // підказка для користувача
        int id; int age; char name[10]; // тимчасові змінні
        fflush(stdin); // очистити буфер
        scanf("%d", &id); // ввели 1 поле
        if(id==0) break; // перервали цикл якщо ввели 0
        scanf("%d",&age); //ввели 2 поле
        scanf("%s",name); //ввели 3 поле

        fprintf(f,"%d %c %s\n", id, (char)age, name); // записали в текстовий файл
    }
    fclose(f); //закрили файл
    return EXIT_SUCCESS; // успішно вийшли
}
```

```
int readTextFile(const char* fname, Student* mas){ /* прочитали text file, записали в mas */
    int id;    char age;    char name[10]; // тимчасові змінні
    int i=0;
    FILE* f = fopen(fname, "rt"); // відкрили для читання
    if(f==NULL) return EXIT_FAILURE; // вийшли з помилкою, якщо не відкрили файл

    do{ // читаємо допоки можливо
        fscanf(f,"%d %c %s", &id, &age, name); // читаємо текстові дані(dangerous)

        mas[i].age = age; /* записуємо дані в mas */
        mas[i].id = id;
        strcpy(mas[i].name , name); // string копіюємо за допомогою strcpy
        i++;
    }while(!feof(f)); // перевіряємо чи є кінець файлу
    fclose(f); //закриваємо файл
    return i-1; // повертаємо скільки раз ми все вдало прочитали
}
```

```
int main(int argc, char **argv){ // головна програма
    const char fname[] = "studs.txt"; // встановили імя файлу
    createTextFile(fname); // викликали функцію створення
    Student studs[10]; // масив для результату
    int k = readTextFile(fname, studs); // викликали функцію читання
    if (k<0) return EXIT_FAILURE; // вийшли якщо помилка
    int i=0;

    for(Student* strt=studs;strt<&studs[k]; ++strt){ //вивід в консоль для fprintf
        fprintf(stdout, "\n Stud [%d]= %d %d %s", i++,
                strt->id, (int)strt->age, strt->name);
    }

    for (i = 0; i < k; i++) { // вивід для printf
        printf("Id = %d, Name = %s, Age = %d \n",
            studs[i].id, studs[i].name, studs[i].age);
    }
}
```



```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
// struct person with 3 fields
```

```
typedef struct Student {
```

```
    char name[10];
```

```
    int id;
```

```
    int age;
```

```
}Student;
```



// створюємо бінарний файл з даним іменем

```
int createBinFile(const char* fname){  
    FILE* f = fopen(fname, "wb"); // відкриваємо бінарний файл для запису  
    if (f==NULL) return EXIT_FAILURE; // exit on failure
```

Student s; // змінна типу Student

```
for(;;){  
    printf("input id or 0\n"); // читаємо дані з консолі  
    scanf("%d", &s.id);  
    if(s.id==0) break; // виходимо при введенні 0  
    scanf("%d",&s.age);  
    scanf("%s",s.name);  
    fwrite(&s, sizeof(s), 1, f); // записуємо одну структуру в файл  
}  
fclose(f); //закриваємо файл  
return EXIT_SUCCESS; // успішний return  
}
```

```
int readBinFile(const char* fname, Student* mas){ // читаємо бінарний файл
    FILE* f = fopen(fname, "rb"); //відкриваємо для читання
    if (f==NULL) return EXIT_FAILURE; // вихід при помилці
```

```
    Student* iter = &mas[0]; /* встановлюємо вказівник на початок результатного
масиву */
```

```
    int k = 0; // лічильник даних
    do{
        fread(iter, sizeof(*mas), 1, f); // читаємо одну структуру Student
        iter++; // наступний блок для результату
        k++; // збільшуємо вказівник
    }while(!feof(f)); // робимо поки не досягли кінця файлу
    fclose(f); // закриваємо файл
    return k-1; // повертаємо кількість зчитаних даних
}
```

```
Student getNth(const char* fname, size_t n){ // читаємо n-го Student'a
    FILE* f = fopen(fname, "rb"); // відкриваємо файл
    fseek (f, 0, SEEK_END); /*встановлюємо маркер на кінець файлу(non-
portable!) */
    size_t size = ftell(f); // видаємо позицію кінця файлу
    if(n>size){ // якщо n більше розміру файлу - вихід
        fclose(f);
        return ((Student){0,0,0}); // return something
    }

    fseek(f, n, SEEK_SET); // встановлюємо файл на n-те місце
    Student res; // результат
    fread(&res, sizeof(res), 1, f); //читаємо дане на n-му місці
    fclose(f);
    return res; //повертаємо результат
}
```

```
int main(int argc, char **argv){

    const char fname[] = "studs.txt"; // імя файлу
    createBinFile(fname); // функція запису файлу
    Student studs[10]; // масив результатів
    int k = readBinFile(fname, studs); // виклик функції читання
    if (k<0) return EXIT_FAILURE; // перевірка коректності читання

    // знаходимо другого студента, та записуємо його в кінець масиву
    studs[k] = getNth(fname, 1);

    // виводимо результат
    for(Student* strt=studs; strt<=&studs[k]; ++strt){
        fprintf(stdout, "\n Stud [%d]= %d %d %s", i++,
                strt->id, strt->age, strt->name);
    }
}
```