Programmētāju skola
3. līmeņa grupa

faili

Faila struktūra

Faili tiek glabāti diskā secīgo baitu veidā, kuru nozīmes ir noteicamas ar faila formātu. Pēc apstrādāšanas principa faili dalās tekstā failos, kas glabā dažāda garuma teksta rindas, un binārā failos, kas glabā datus tādā pašā veidā, kā tie tiek glabāti operatīva atmiņā.

PNG faila fragments (dump)

```
89 50 4E 47 0D 0A 1A 0A | 00 00 00 0D 49 48 44 52
00 00 0E 35 00 00 04 79 | 08 02 00 00 01 C7 20 C1
B3 00 00 00 06 74 52 4E | 53 00 FF 00 FF 00 FF 37
58 1B 7D 00 00 00 09 70 | 48 59 73 00 00 5C 46 00
00 5C 46 01 14 94 43 41 | 00 00 00 09 74 45 58 74
43 6F 6D 6D 65 6E 74 00 | 00 89 2A 8D 06 00 00 20
00 49 44 41 54 78 9C EC | DD E9 B3 75 55 79 28 7A
FE 02 BF A4 CA 5E 51 13 | 23 36 28 44 63 20 36 98
98 18 BB 78 8C 1A DB C4 | 34 48 34 8D A1 97 D0 A9
D8 A0 12 11 05 14 51 02 | 08 08 08 48 2B 20 DB 2E
1A D3 D8 62 83 E6 9C 9C | 4A D5 8D 9A 73 6E DD 24
F7 43 72 EA DC 3A 31 66 | DF 4D 2D 6A B9 58 DD 9E
6B CD 31 C7 78 C6 98 BF | 5F 3D 1F 60 BF 7B CF F5
CC 67 8E F9 CC 67 8D BD | F7 FB 1E B0 0B 50 B3 03
CE 3D 62 2F 4A 67 F1 63 | 07 94 4E 00 60 7B 93 96
BA 75 57 9D 7E F9 D6 07 | 59 3C 82 AE 0A D4 2D 55
4B DD AE B1 EA AA 40 2F | 19 DE 6E 17 79 47 DF A7
...
```

8 – signature (‰PNG←↓↕↓)

4 – length (10)

4 – header (IHDR)

4 – width (3637)

4 – height (1145)

1 – bit depth (32)

1 – color type (2)

1 – compression (0)

1 – filter (0)

1 – interlace (1)

...

Teksta faili

Teksta fails sastāv no baitiem, katrs no kuriem glabā viena simbola ASCII kodu. Ar speciālo baitu secīgumu apzīmē pozīcijas, kas nobeidz rindas. Windows sistēmā tam pielieto divus secīgus baitus **CR** (13 = 0D) un **LF** (10 = 0A).

Teksta faila fragments (dump)

```
46 72 6F 6D 20 66 61 69 | 72 65 73 74 20 63 72 65 | From fairest cre
61 74 75 72 65 73 20 77 | 65 20 64 65 73 69 72 65 | atures we desire
20 69 6E 63 72 65 61 73 | 65 2C 0D 0A 54 68 61 74 |  increase,←↓That
20 74 68 65 72 65 62 79 | 20 62 65 61 75 74 79 27 |  thereby beauty'
73 20 72 6F 73 65 20 6D | 69 67 68 74 20 6E 65 76 | s rose might nev
65 72 20 64 69 65 2C 0D | 0A 42 75 74 20 61 73 20 | er die,←↓But as
74 68 65 20 72 69 70 65 | 72 20 73 68 6F 75 6C 64 | the riper should
20 62 79 20 74 69 6D 65 | 20 64 65 63 65 61 73 65 |  by time decease
2C 0D 0A 48 69 73 20 74 | 65 6E 64 65 72 20 68 65 | ,←↓His tender he
69 72 20 6D 69 67 68 74 | 20 62 65 61 72 20 68 69 | ir might bear hi
73 20 6D 65 6D 6F 72 79 | 2E                      | s memory.
```

vesels skaitlis	bināra formā	teksta formā
123 456 →	01 E2 40 →	30 31 32 33 34 35

Secīga piekļuve

Parasti faila datiem tiek piekļūts secīgi, t.i. lai nolasītu kaut kādu baitu, ir jāizlasa visi baiti, kas atrodas pirms tā.

Katram failam ir saistīts rādītājs (pos) uz vietu, kur atrodas pirmais neizlasītais baits. Pēc atvēršanas šis rādītājs parasti norāda uz faila sākumu (pos = 0).

Pēc pirmā baita izlasīšanas faila rādītājs tiek nobīdīts pie nākamā baita

Pēc katras lasīšanas operācijas faila rādītājs tiks nobīdīts uz izlasīto baitu skaitu

Lai ieviestu brīvpiekļuvi, ir iespēja pārvietot faila rādītāju uz vēlamo pozīciju vai nolasīt rādītāja pašreizējo vērtību.

Faila vārds

Atverot failu ir jānorāda tā nosaukums kopā ar atrašanās vietu. Šo kombināciju sauc par faila piekļuves ceļu (path). Piekļuves ceļa formāts dažādās operētājsistēmās atšķiras.

Windows sistēmās piekļuves ceļš sākas ar diska burtu, kam seko atdalīti ar apvērstām slīpsvītrām mapju nosaukumi, un beidzas ar faila vārdu un paplašinājumu.

Šo faila piekļuves ceļu sauc par absolūto. Praksē labāk izmantot relatīvos ceļus, kas apraksta faila atrašanās vietu, sākot no tekošas mapes (parasti no mapes kur atrodas programma).

```
.\folder1\folder2\name.ext
```

```
folder1\folder2\name.ext
```

Pēc noklusējuma, ja faila mapju saraksts nav norādīts, tiek pieņemts, ka fails atrodas tekošajā mapē.

```
name.ext
```

Funkcijas darbam ar failiem

bibliotēka `cstdio`

`fopen_s(&file, name, mode)`	Atvērt failu ar vārdu name piekļūšanas režīmā mode (r, w, a, t, b). Pēc atvēršanas funkcija atgriež faila identifikatoru file, ko ir jānorāda citas failu funkcijas izsaukšanas laikā. Kļūdas gadījumā atgriež nulli.
`fclose(file)`	Aizvērt failu.
`char = fgetc(file)`	Izlasīt vienu simbolu (baitu) no faila un ierakstīt to uz mainīgā char.
`fputc(char, file)`	Ierakstīt vienu simbolu char uz faila.
`fgets(str, len, file)`	Izlasīt simbolu rindu no faila un ierakstīt to uz mainīgā str ar maksimālo izmēru len. Rinda būs izlasīta kopa ar CR simbolu, kas tā ierobežo.
`fputs(str, file)`	Ierakstīt simbolu rindu str uz faila.
`fscanf_s(file, fmt, ptr)`	Izlasīt no faila datus saskaņā ar formātu fmt un ierakstīt izlasītas vērtības uz mainīgos ptr.
`fprintf(file, fmt, var)`	Ierakstīt uz faila var mainīgos vērtības saskaņā ar formātu fmt.
`feof(file)`	Noteikt vai lasāmais fails ir pabeigts: funkcija atgriež true ja failā vēl nav neizlasītus datus, ja fails nav pabeigts un lasīšana var būt turpināta funkcija atgriež false.

Lasīšanas no teksta faila piemērs

Piemēram, vajag nolasīt visas teksta faila rindas un parādīt tās ekrānā

```
#include <iostream>
#include <cstdio>
#include <string>

using namespace std;

bool showFile(const string& fileName)
{
    FILE* file;
    fopen_s(&file, fileName.c_str(), "rt");
    if (!file) return false;
    char str[100];
    while (fgets(str, 100, file)) cout << str;
    fclose(file);
    return true;
}

int main()
{
    if (!showFile("test.txt")) cout << "Failed to open file\n";
}
```

Ierakstīšanas uz teksta faila piemērs

Piemēram, teksta fails ir jāaizpilda ar simbolu rindām, kas ievadītas no tastatūras.

```
#include <iostream>
#include <string>
#include <cstdio>

using namespace std;

bool fillFile(const string& fileName)
{
    FILE* file;
    fopen_s(&file, fileName.c_str(), "wt");
    if (!file) return false;
    string str;
    do {
        cout << "> "; getline(cin, str);
        fputs(str.c_str(), file);
        fputs("\n", file);
    } while (str != "");
    fclose(file);
    return true;
}
```

```
int main()
{
   if (fillFile("test.txt"))
      cout << "File is created\n";
   else
      cout << "Failed to create file\n";
}
```

Formatējošā rakstīšana

Formatējošās funkcijas ļauj pārveidot dažāda tipa datus teksta virknē.

`fprintf(file, format, data,...)`	konvertēt datus uz rindu un ierakstīt uz faila file
`sprintf(str, format, data,...)`	konvertēt datus uz rindu un saglabāt mainīgajā str
`printf(format, data,...)`	konvertēt datus uz rindu un izvadīt uz ekrāna

Teksta virknes formātu apraksta formāta rinda, kas ir teksts ar īpašiem specifikatoriem, kas apzīmē vietas tekstā, kur ir jāievieto dati. Specifikatoram ir jāsākas ar procentu zīmi (%), pēc kura seko simbols, kas nosaka datu tipu. Starp procentu zīmi un tipa simbolu varat norādīt papildu parametrus, kas nosaka, kā dati tiek pārvērsti tekstā.

```
int a = 15; float b = 0.5; string c = "Hi";
fprintf(file, "a = %i, b = %f, c = %s\n", a, b, c.c_str());
// a = 10, b = 0.500000, c = "Hi"
```

%i	15	%5i	␣␣␣15	%05i	00015
%f	0.500000	%.2f	0.50	%7.2f	␣␣␣0.50
%s	Hi	%5s	␣␣␣Hi	%-5s	Hi␣␣␣

Teksta datu faila rakstīšanas piemērs

Uzrakstīt uz teksta faila veselā skaitļu masīvu. Skaitļiem failā jābūt atdalītiem viens no otra ar rindas beigām.

```
bool writeData(const string& fileName, const vector<int>& array) {
    FILE* file; fopen_s(&file, fileName.c_str(), "wt");
    if (!file) return false;
    for (int i: array) fprintf(file, "%i\n", i);
    fclose(file);
    return true;
}

int main()
{
    srand(time(nullptr));
    vector<int> v;
    for (int i = 0; i < 100; i++) v.push_back(rand() % 2001 - 1000);
    if (!writeData("data.txt", v)) cout << "File write error\n";
    else cout << "Faile successfully filled\n";
}
```

Formatējošā lasīšana

Formatējošas lasīšanas funkcijas ļauj ne tikai izlasīt rindu, bet arī sadalīt to dažādu tipu laukos (parsing) un katra lauka vērtību ierakstīt uz atsevišķa mainīgā.

`fscanf_s(file, format, var,...)`	izlasīt rindu no file faila un sadalīt to laukos
`sscanf_s(str, format, var,...)`	sadalīt rindu no str mainīgā laukos
`scanf_s(format, var,...)`	izlasīt rindu no klaviatūras un sadalīt to laukos

Avota teksta struktūra ir aprakstīta formātā, kurā specifikatori atzīmē nepieciešamo datu atrašanās vietas un tā tipus.

Var sarakstā ir jānorada mainīgās adreses, kuros būs ierakstītie dati. Katram specifikatoram no formāta rindas ir jāatbilst viens atbilstoša tipa mainīgais

Formatējošās lasīšanas funkcijas atgriež veselo skaitli, kas nozīmē cik lauku tika nolasītie.

```
int a; float b;
if (fscanf_s(file, "%i, %f", &a, &b) != 2) std::cout << "Read error\n";
```

Lasīšanas formāta specifikātori

Specifikators	Dati	Mainīgais sarakstā
%d, %i	vesels skaitlis	int tipa mainīgais
%f	reālais skaitlis	float tipa mainīgais
%c	viens simbols	char tipa mainīgais
%s	ierobežota ar tukšumzīmi simbolu rinda	char tipa masīvs un tā garums piem. str, 100
%Wc	simbolu secīgums ar W simboliem (masīvā nebūs ierakstīta ierobežojoša nulle) piem. %15c	
%[…]	simbolu rinda, kas var saturēt tikai pārskaitītos simbolus piem. %[A-Za-z] – tikai latīniskie burti	
%[^…]	simbolu rinda, kas var saturēt jebkurus simbolus izņemot norādītus piem. %[^A-Za-z] – visi simboli, izņemot latīniskus burtus	

Pēc procenta zīmes var norādīt zvaigznītes zīmi (*), tad attiecīgais lauks būs izlasīts, bet nebūs ierakstīts uz mainīgā (t.i. būs izlaists).Piemēram, %*10c – izlaist 10 baitus.

Teksta datu faila lasīšanas piemērs

Izlasīt no teksta faila veselā skaitļu masīvu. Skaitļi failā var būt atdalītie viens no otra ar tukšumzīmēm vai ar rindas beigām.

```
bool readData(const string& fileName, vector<int>& array) {
    FILE* file; fopen_s(&file, fileName.c_str(), "rt");
    if (!file) return false;
    int val;
    while (fscanf_s(file, "%i", &val) == 1) array.push_back(val);
    fclose(file);
    return true;
}

int main()
{
    vector<int> v;
    if (!readData("data.txt", v)) cout << "File read error\n";
    else for (int i: v) cout << i << "\n";
}
```

Teksta rindas parsēšana

```
char str[] = "Ivars Vingers, 10.08.2005., skolnieks, 11. klase, 1.vidusskola";

char name[50];
int day, month, year;
int group;
char school[50];

int cnt = sscanf_s(
    str,
    "%[^,], %d.%d.%d., %*s %d. klase, %s",
    name, 50, &day, &month, &year, &group, school, 50
);

if (cnt != 6) printf("String format error\n");
else printf("Skolnieks %s, %d. klase\n", name, group);


   // Skolnieks Ivars Vingers, 11. klase
```

Buferizācija

Visas failu lasīšanas un rakstīšanas operācijas kešejas. Piemēram, ierakstītie dati uzreiz nenonāk failā, bet vispirms tiek uzkrātie buferī atmiņā. Rakstīšana no bufera uz failu notiek, kad buferis ir pilns, kad fails aizveras vai ar īpašu komandu.

Faila raksturojumu iegūšana

Ja vajag uzzināt faila raksturojumus (piem., izmēru, izveidošanas vai pēdējās modifikācijas laiku…) var izmantot funkciju `stat` no bibliotēkas `sys/stat.h`

Funkcija `stat` saglabā informāciju par failu ar doto vārdu uz `stat` tipa mainīgā:

```
struct stat fileInfo;
stat(fileName, &fileInfo);
```

Pēc `stat` funkcijas pabeigšanas faila raksturlielumus var nolasīt no mainīgo laukiem:

`fileInfo.st_size`	izmērs baitos
`fileInfo.st_ctime`	izveidošanas laiks
`fileInfo.st_mtime`	pēdējās modifikācijas laiks

Faila laikspiedoli tiek glabāti Unix Epoch jeb Unix Timestamp formātā (`time_t`), kas ir vesels skaitlis, kas nozīme sekunžu skaitu kopš 1970. gada 1. janvāra.

`11.01.2024 16:30:00` → `1 704 983 400`

Faila raksturojumu iegūšanas piemērs

```
#include <iostream>
#include <sys/stat.h>

using namespace std;

int main()
{
    string name;
    cout << "File name: "; cin >> name;

    struct stat info;
    stat(name.c_str(), &info);

    cout << "Size: " << info.st_size << "\n";
    cout << "Created (Unix epoch time): " << info.st_ctime << "\n";
    cout << "Modified (Unix epoch time): " << info.st_mtime << "\n";

    char buf[1000];
    ctime_s(buf, 100, &info.st_ctime); // pārveidot laikspiedolu uz teksta rindu
    cout << "Created: " << buf;

    ctime_s(buf, 100, &info.st_mtime); // pārveidot laikspiedolu uz teksta rindu
    cout << "Modified: " << buf;
}
```