ЛЕКЦИЯ 17

Организация ввода-вывода

В языке C++ (и в C тоже) не предусмотрены какие-либо возможности для организации ввода-вывода. Весь ввод-вывод перенесен в специальные библиотеки. Это позволило добиться эффекта “независимости от платформы”. Иначе говоря, получить аппаратно независимый язык разработки программ для различных платформ.

В программах на языке C++ можно равноправно использовать две библиотеки ввода-вывода: стандартную библиотеку функций языка C (стандарт ANSI C) и библиотеку классов, специально созданную для языка C++. Эти библиотеки используют два принципиально различных механизма для организации ввода-вывода:
первый основан на концепции файлов; второй работает со специальными классами, ориентированными на ввод-вывод – потоковыми классами.

Работа с библиотекой

Чтобы получить доступ к функциям библиотеки, приложение должно подключить файл **`stdio.h`**. Этот файл содержит объявления констант, типов данных и структур, используемых этими функциями, а также прототипы самих функций и описания служебных макросов.

Многие константы, содержащиеся в файле **`stdio.h`**, находят достаточно широкое применение в приложениях:
константа **`EOF`** возвращается функциями ввода при достижении конца файла, (обычно=-1);
константа **`NULL`** служит нулевым ("пустым") указателем;
`FILE` – имя типа структуры стандартного файлового ввода-вывода;
константа **`BUFSIZ`** - стандартный размер в байтах буфера для обмена данными с файлом.

Максимальная длина имени файла описывается переменной **`FILENAME_MAX`** (для Windows - 260). Максимальное количество файлов, которое можно открыть одновременно, - **`FOPEN_MAX`** (для Windows - 20).

Обработка файлов

Файл – это последовательность байтов, хранящихся на внешнем носителе информации. Каждый файл имеет имя. Работа с файлом поддерживается операционной системой, которая имеет средства:
• создания и уничтожения файлов;

• поиска файлов на внешнем носителе;
• чтения и записи данных из файлов и в файлы;
• открытия и закрытия файлов;
• позиционирования файлов.

Для осуществления доступа к файлу нужно знать его организацию. Файл может состоять:
из непрерывного потока символов; последовательности строк переменной длины;
форматированных текстовых полей, разделенных разделителями;
последовательности записей (блоков) постоянной длины.

Следовательно, для работы с файлами должны иметь функции:
доступа к файлу (открытие файла);
посимвольный ввод-вывод;
построчный ввод-вывод;
форматированный ввод-вывод;
ввод-вывод записей.

Разделяют файлы текстовые и бинарные.

В текстовом режиме в файл записываются и читаются текстовые строки, которые заканчиваются символом ‘\n’ и могут содержать символы ‘\t’.

При записи и чтении данных может происходить преобразование этих данных.

В бинарном режиме данные записываются и читаются один к одному.

Файл – это последовательность байтов, хранящихся на внешнем носителе информации.

Поток – логический интерфейс (программа) который обеспечивает доступ программы-пользователя к файлу.

Прежде чем использовать поток для доступа к файлу, его необходимо соединить с файлом.

Эта информация хранится в структуре типа FILE, поэтому поток имеет вид FILE* (их часто отождествляют).

Файлы описываются с помощью указателей на внутреннюю структуру FILE.
Примеры описания:

```
FILE *f1, *f2;
```

Программист имеет возможность работы с предопределенными файлами **stdin, stdout, stderr, stdaux и stdprn.**

Работа с файлом начинается с его открытия.

Открытие файла

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

fname – имя файла:
mode – режим открытия.
В случае удачи возвращает указатель на поток, иначе – 0.
mode может принимать значения:
r существующий файл открывается для чтения;
w создается новый файл, который открывается для записи; если такой файл уже существует, то предыдущее содержимое стирается;
a существующий файл открывается для добавления информации в его конец;
r+ существующий файл открывается для чтения и записи; но не разрешается запись в конец файла, т.е. увеличение файла;
w+ создается новый файл, который открывается для чтения и записи;
a+ существующий файл открывается для чтения и добавления информации в его конец или создается при его отсутствии.

Кроме того, в режиме открытия можно указывать символы **t** или **b**, которые соответствуют текстовому или бинарному режиму (в первом случае признаки конца строки обрабатываются по-особому, во втором – они не выделяются из другой информации).

Перенаправление потока

```
FILE* freopen (const char* filename, const char* mode,
FILE* stream);
```

Закрывает файл, соединенный с потоком **stream**, и соединяет этот поток с файлом **filename** в режиме **mode**.

Закрытие файла

```
int fclose(FILE* stream);
```

Закрывает файл, при этом освобождая все буферы потока. В случае удачи возвращает 0, иначе – **EOF**.

Часто объявление файла объединяется с его открытием. Таким образом, канва программы, работающей с файлами, имеет следующий вид:

```
 if ((FILE *f = fopen("myfile.txt","rt")) != NULL)
{  // работа с файлом
   fclose(f);
}
```

Проверка на конец файла

Каждый поток содержит индикатор конца файла, который хранится в структуре файла и устанавливается в ненулевое значение функцией чтения из файла при достижении конца файла.

Состояние потока определяется функцией:

```
int feof(FILE* file);
```

Возвращает 0, если конец файла не достигнут. При достижении конца файла содержимое последней прочитанной порции информации не определено.

Операции ввода-вывода

Ввод-вывод в файл можно осуществлять различными способами, в зависимости от поставленной задачи.

Работа со строками

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

Пишет строку **str** в поток **stream**, не включая завершающий нулевой байт. В случае удачи возвращает 0 (или ASCII-код последнего вводимого символа), иначе возвращает **EOF**.

Для записи строки в стандартный выходной поток **stdout** применяется функция

```
int puts(char* str);
```

При выводе завершающий нулевой байт строки преобразуется в символ новой строки.

```
char* fgets(char* str, int size, FILE* stream);
```

Читает строку из потока **stream** в строку **str**, остановится: • прочитан **size+1** символ;

- встретился символ новой строки;
- достигнут конец файла.

Символ **\n** копируется в строку, к концу строки добавляется нулевой байт **\0**. Возвращает при успехе указатель на **str**, иначе возвращает **NULL**. Строка **str** не изменяется, если ни один символ не прочитан и встретился конец файла.

Для чтения из стандартного потока **stdin** применяется функция

```
char* gets(char* str);
```

Нет способа ограничить число символов при вводе, поэтому массив, адресуемый указателем **str**, может переполниться.

Пример 18.1. ***Построчное чтение из текстового файла***

```
void ReadStringFile( FILE *in,char* filename )
{char *str=(char*)malloc (81);
 if ((in = fopen(filename,"r"))!= NULL)
 {
    fgets(str,80,in); // gets(str);
    if (feof(in)) printf("Файл пустой\n");
    while (!feof(in))
    {
        fputs(str,stdout);
        fgets(str,80,in); // gets(str);
    }
     fclose(in);
  }
  else
    printf("Файл не открывается\n");
}
```

Пример 18.2. ***Построчное создание текстового файла***

```
void CreateStringFile(FILE *out,char* filename)
{int k;
 char str[80];
 out = fopen(filename,"w");
 gets(str);
 while(!feof(stdin))
 {  fputs(str,out);
    gets(str);// Не заносит в строку
              //символ возврата каретки, Заносит \0
 }
 fclose(out);
}
```

Пример 18.3. ***Текстовый файл input.txt содержит строки длиной не более 100 символов. Найти число строк, начинающихся с пробела.***

```
int main()
{    int N = 0;
     FILE* f;
     char *S = new char[101];
     if ((f = fopen("input.txt", "rt"))!= NULL)
```

```
    {
        while (fgets(S, 101, f)!= NULL)
          if (S[0] == ' ') N++;
        fclose(f);
        cout << N;
    }
    else
        printf("Файл не открывается\n");
    return 0;
}
```

Посимвольный ввод-вывод

```
int fputc(int c, FILE* stream);
int putc(int c, FILE* stream);
```

Записывает символ **c** в поток **stream** и продвигает индикатор позиции на следующий символ. Перед записью в файл параметр **c** типа **int** преобразуется в тип **unsigned char**. При успехе возвращает **c**, в случае неудачи **EOF** (-1).

Функции идентичны, но **putc** определена обычно как макрос.

```
int fgetc(FILE* stream);
int getc(FILE* stream);
```

Работают аналогично **fputc**, но читают символ. В случае достижения конца файла возвращают **EOF** и устанавливают индикатор конца файла.

В консольном режиме значению **EOF** соответствует нажатие клавиш **Ctrl+Z**.

Функции

```
int getchar();
int putchar(int ch);
```

частные случаи функций **getc и putc**.

```
void ReadChFile(FILE *in, char* filename )
   // Чтение текстового файла посимвольно
{
    int ch;
    if ((in = fopen(filename, "r")) != NULL)
    {
     ch = getc(in);
     if(ch  == EOF) printf("Файл пустой\n");
     while (ch  != EOF)
     {
           putc(ch,stdout);
```

```
            ch = getc(in);
     }
      fclose(in);  // закрываем файл
    }
      else printf (" Файл не открывается\n");
   }

void CreateChFile(  FILE *out, char* filename )
//Создание  текстового файла по одному символу
{
    int ch;
    out = fopen(filename, "w");
   ch=getchar();
    while (ch!=EOF)
    {
          putc(ch,out);
          ch=getchar();
    }
    fclose(out);
}
```

Форматированный ввод-вывод

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

Форматированный вывод в файл **stream**. Работает аналогично **sprintf**, в случае удачи возвращает число выводимых символов, иначе возвращает -1.

```
int printf(const char * формат, …)
```

```
int fscanf(FILE* stream,  const char* format
[ адреса_переменных ],…);
```

Форматированный ввод из файла **stream**, в случае успеха возвращает количество единиц прочитанных данных, иначе возвращает **EOF**.

```
int scanf(const char * формат,
[ адреса_переменных ],…)
```

```
void CreateFStringFile( FILE *out, char* filename )
//Создание  текст. файла, используя форматированный ввод строк
{  int k;
```

```
    char str[80];
    out = fopen(filename, "w") ;
    scanf("%s",str);
    //считывает до первого пробела либо конца строки
    while(!feof(stdin))
    {
      fprintf(out,"%s\n",str);
      scanf("%s",str);
    }
    fclose(out);
}


void ReadFStringFile(  FILE *in, char* filename )
{
   char *str=(char*)malloc (80);
    if ((in = fopen(filename, "r") ) != NULL)
    {
       fscanf(in,"%s",str);
       if (feof(in))
          printf("Файл пустой\n");
       else
       {
          do
          {
              printf("%s\n",str);
              fscanf(in,"%s",str);
          }
          while (!feof(in));
       }
       fclose(in);
    }
    else
      printf (" Файл не открывается\n");
}
```

Блочный ввод-вывод

Блоком называется область оперативной памяти, содержимое которой записывается в байтах

.

Ввод-вывод блоками используется бинарными потоками.

Запись блоков в файл:

```
size_t fwrite (const void* ptr,   size_t size, size_t
nitems,  FILE* stream );
```

где **ptr** указывает на блок памяти, который записывается в файл ;
длина блока указывается как **size*nitems**
(здесь '*' знак умножения). Функция возвращает число записанных единиц. В случае удачи это число должно быть равно **nitems**.

Чтение:

```
size_t fread(void* ptr,   size_t size, size_t nitems,
    FILE* stream);
```

Параметры имеют то же смысл, что и в **fwrite.**

Пример 18.4. ***Создание бинарного файла***

```
void CreateBinFile (void)
{struct emp
 {  int code;
    char name[20];
 }
 FILE* out;
 struct emp s;
 if(!(out=fopen("z.bin","wb")))
 {   printf("Open file failed.\n");
    return;
 }
 printf("Input code,name.\n");
 printf("Press Ctrl+z to exit.\n");
 printf("1. ");
 int i=2;
 scanf("%d%s",&s.code,&s.name);
 while (!feof(stdin))
 { fwrite(&s,sizeof(struct emp), 1, out);
   printf("%d. ",i);
   i++;
   scanf("%d%s",&s.code, &s.name);
 }
 fclose(out);
}
```

В чем недостаток этой функции???????? Как исправить???

Пример 18.5. ***Чтение бинарного файла***

```
 void  ReadBinFile(void)
struct emp
{    int code;
     char name[20];
};
FILE* in;
struct emp s;
unsigned i;
if(!(in = fopen("z.bin", "rb")))
{ printf("Open file failed.\n");
  return;
}
fread(&s, sizeof(struct emp),1, in);
do
{ printf("\tcode = %d name = %s\n", s.code, s.name);
  fread(&s, sizeof(struct emp), 1, in);    }
while (!feof(in));
fclose(in);}
```

В чем недостаток этой функции???????? Как исправить???

Для сравнения:

C++	***Паскаль***

`fread(&p1,sizeof p1,1,f1);`
`while (!feof(f1))`
`{`
`//обработать запись файла`
`fread(&p1,sizeof p1,1,f1);`
`}`
	`while not eof(f1) do`
	`begin`
	`read(f1, p1);`
	`// обработать запись файла`
	`end;`

Или:
`while (fread(&p1,1,`
`sizeof p1,f1)==sizeof p1)`
`{`
`//обработать запись файла`
`}`	