Wykład 16 — Wejście–wyjście w C

W języku C do kmunikacji pomiędzy programem a światem zewnętrznym wykorzystuje się tzw. *strumienie*. Strumieniem w C nazywamy każde źródło danych wejściowych i każde miejsce zapisywania wyników. Prototypy funkcji operujących na strumieniach, jak i odpowiednie definicje typów są zapisane w bibliotece <stdio.h>. Do strumieni odwołujemy się za pomocą zmiennych wskaźnikowych typu FILE $*$. Oto przykład deklaracji takich zmiennych.

```
FILE *plik1, plik2;
```

W C dostępne są trzy standardowe strumienie, których nie trzeba deklarować, ani też otwierać i zamykać. Należą do nich:

1. stdin — standardowe wejście, domyślnie klawiatura;
2. stdout — standardowe wyjście, domyślnie ekran monitora;
3. stderr — standardowe miejsce umieszczania komunikatów o błędach, domyślnie ekran monitora.

Wszystkie dotychczas używane funkcje — printf, scanf, putchar, getchar, puts, gets — odnosiły się do standardowego wejścia i standardowego wyjścia. Pewne systemy operacyjne (UNIX, DOS) umożliwiają zmianę domyślnego przypisania strumieni. Taki mechanizm nazywa się *przeadresowywaniem*. W DOSie lub UNIXie można zmienić źródło stdin wywołując program jak w przykładzie:

```
demo <dane.we
```

Podobnie można przeadresować wyjście:

```
demo >dane.wy
```

lub jednocześnie i wejście i wy"jscie:

```
demo <dane.we >dane.wy
```

Plikiem w C nazywamy ciąg bajtów. Wyróżnia się dwa rodzaje plików. *Pliki tekstowe* to te, w których każdy bajt jest interpretowany jako jeden znak. W *plikach binarnych* grupy kolejnych bajtów mogą reprezentować elementy innych typów danych, jak na przykład liczby typu long lub float. Na przykład liczbę 32767 pamiętamy w pliku tekstowym (w kodzie ASCII) na 5 bajtach:
00110011 00110010 00110111 00110110 00110111
ale potrzeba tylko dwóch bajtów na zapamiętaniu jej w pliku binarnym:
01111111 11111111
Jaka jest najważniejsza różnica pomiędzy tymi plikami. Plik tekstowy jest podzielony na wiersze, każdy kończący się znakiem końca wiersza. Na końcu pliku znajduje się (lub powinien się znajdować) znak końca pliku. Uwaga: te zasady bardzo zależą od systemu operacyjnego.
Oto podstawowe operacje na plikach (strumieniach):

- Otwieranie pliku
  FILE *fopen(const char *filename, const char *mode);
  Oto przykłady otwierania pliku.

  ```
  #include <stdio.h>
  main()
  {
  ```

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

  scanf("%d%d", &i, &j);
  f = fopen("c:\\moje\\test1.we", "w");
  fprintf(f, "%d %d\n", i, j);
  fclose(f);

  return 0;
}
```

Jeśli plik nie może zostać otwarty zwracany jest wskaźnik NULL. Plik może zostać otwarty w jednym z następujących trybów:
PLIKI TEKSTOWE
"r" — tylko do czytania
"w" — tylko do zapisu (nie musi wcześniej istnieć)
"a" — tylko do uzupełniania (nie musi wcześniej istnieć)
"r+" — otwarty do czytania i zapisu (od początku)
"w+" — otwarty do czytania i zapisu (najpierw jest czyszczony)
"a+" — otwarty do czytania i zapisu (uzupełnianie, gdy istnieje).
PLIKI BINARNE
Tryby są takie same jak poprzednio, ale uzupełnione literą b, np. "rb" lub "w+b" (lub "rb+).

- Zamykanie pliku
  int fclose(FILE *stream);

fclose zamyka poprzednio otwarty plik. Jeżeli zamknięcie pliku kończy się sukcesem zwracana jest wartość 0, w przeciwnym razie stała EOF predefiniowana w stdio.h. Oto przykład użycia fopen i fclose.

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

#define NAZWA_PLIKU "dane"

main()
{
  FILE *f;

  f = fopen(NAZWA_PLIKU, "r");
  if ( f == NULL ){
    printf("Nie moge otworzyc pliku %s\n",
            NAZWA_PLIKU);
    exit(EXIT_FAILURE);
  }
  ...
  fclose(f);
  return 0;
}
```

- Wiązanie pliku do strumienia
  FILE *freopen(const *filename, const char *mode, FILE *stream);
  Tej funkcji używa się do związania nowego pliku z jednym ze standardowych strumieni. Plik poprzed-

nio związany ze strumieniem zostanie zamknięty.

```
...
if (freopen("nowy", "w", stdout) == NULL {
  /* blad; pliku nie mozna otworzyc */
}
```

- Nazwy plików jako argumenty programu
  Powiedzmy, że nasz program demo operuje na dwóch plikach, które mają być argumentami programu. Wówczas funkcja główna w tym programie musi mieć postać:

```
main(int argl, char *argw[])
{
  ...
}
```

Zeby wywołać ten program dla plików nazwiska.we, daty.we piszemy
`demo nazwiska.we daty.we`.
Oto przykład programu, który sprawdza, czy można otworzyć dany plik.

```
#include <stdio.h>

main(int argl, char *argw[])
{
  FILE *f;
  if (argl != 2){
    printf("zla liczba argumentow\n");
    return 2;
```

```
  }

  if ((f = fopen(argw[1], "r")) == NULL) {
    printf("%s nie mozna otworzyc\n", argw[1]);
    return 1;
  }
  printf("%s mozna otworzyc\n", argw[1]);
  fclose(f);
  return 0;
}
```

- Pliki tymczasowe
  W rzeczywistych programach często występuje potrzeba użycia plików tymczasowych. W C do tego celu służą dwie funkcje:
  FILE *tmpfile(void);
  char *tmpnam(char *s);
  tmpfile tworzy tymczasowy plik, który istnieje tylko do chwili jego zamknięcia lub zakończenia programu. Zauważmy, że ten plik nie ma nazwy, a więc nie można z niego zrobić pliku stałego. Jeśli chcemy z pliku tymczasowego zrobić stały należy użyć funkcji tmpnam. tmpnam generuje nazwę dla pliku tymczasowego. Jeśli argumentem dla tmpnam jest NULL, funkcja zwraca wskaźnik do wygenerowanej nazwy, w przeciwnym razie funkcja kopiuje wygenerowaną nazwę do tablicy dostarczonej przez użytkownika.

  ```
  char *nazwa_pliku;
  ```

```
nazwa_pliku = tmpnam(NULL);
------------------------------
char nazwa_pliku[L_tmpnam];

tmpnam(nazwa_pliku);
```

- Opróżnianie bufora
  Buforowanie służy do przyśpieszania operacji wejścia-wyjścia. Przy zapisywaniu (odczytywaniu) danych na dysk są one zapisywane w wydzielonym fragmencie pamięci operacyjnej zwanym buforem. Faktycznie przesłanie danych na dysk następuje po wypełnieniu całego bufora. Podobnie, dane z dysku są pobierane całymi fragmentami i umieszczane w buforze. Jeżeli dane, które odczytujemy są w buforze to nie potrzeba wykonywać (zawyczaj wolnej) operacji dyskowej. Funkcje w stdio.h zazwyczaj wykonują buforowanie automatycznie. Programista może opróżniać bufor za pomocą funkcji
  int fflush(FILE *stream);
  Wywołanie fflush(f) opróżnia bufor związany z f, natomiast fflush(NULL) opróżnia wszystkie bufory.

- Inne operacje na plikach
  int remove(const char *filename);
  int rename(const char *old, const char *new);
  remove usuwa wskazany plik, natomiast rename zmienia starą nazwę pliku na nową.
  ```
  remove("plik");
  rename("stara", "nowa")
  ```

Formatowane wejście-wyjście

Do wypisywania na strumienie wyjściowe służą funkcje
int fprintf(FILE *stream, const char *format, ...);
int printf(const char *format, ...);
Jedyna różnica pomiędzy printf i fprintf jest taka, że printf pisze zawsze na stdout.
Do czytania ze strumieni wejściowych służą funkcje
int fscanf(FILE *stream, const char *format, ...);
int scanf(const char *format, ...);
scanf zawsze czyta ze strumienia stdin podczas gdy fscanf czyta ze strumienia będącego argumentem.
Obie funkcje zwracają liczbę danych wczytanych bezbłędnie; jeśli żadna dana nie może być wczytana zwracane jest EOF.
Oto sposób wczytywania liczb całkowitych

```
while (scanf("%d", &i) == 1){
   ...
}
```

Wykrywanie końca pliku i powodu błędu

Jeśli chcemy żeby funkcja ...scanf wczytała n danych, to po prawidłowym wczytaniu danych wartością funkcji jest n. W przeciwnym razie mamy trzy rodzaje błędu.

- Napotkany został koniec pliku.
- Wystąpił błąd poza kontrolą funkcji.
- Dane były niezgodne z formatem.

Z każdym strumieniem związane są dwa wskaźniki: wskaźnik błędu i wskaźnik końca pliku. Oba wskaźniki są czyszczone z chwilą otwarcia pliku. Kiedy wskaźnik zostaje "zapalony", pozostaje zapalonym do momentu wyczyszczenia za pomocą funkcji clearerr. clearrer czyści oba wskaźniki.

```
clearrer(f);
```

Powyższe wywołanie czyści wskaźniki błędu i końca pliku związane z plikiem f. Do odczytu wskaźników służą funkcje

```
int feof(FILE *stream);
int ferror (FILE *stream);
```

feof zwraca wartość niezerową, gdy napotkany został koniec pliku, natomiast ferrorf zwraca wartość niezerową, gdy wystąpił błąd odczytu, ale różny od niezgodności z formatem. Oto przykład zastosowania powyższych funkcji. Funkcja find_int znajduje w pliku wejściowym wiersz zaczynający się od liczby typu int. Zwraca numer tego wiersza i tą liczbę; gdy wystąpi błąd zwracany jest kod błędu.

```
int find_int(const char *filename, int *ptr)
{
  FILE *fp = fopen(filename, "r");
  int line = 1;

  if (fp == NULL)
    return -1; /*pliku nie mozna otworzyc */

  while (fscanf(fp, "%d", ptr) != 1) {
    if (ferror(f)) {
      fclose(f);
      return -2; /*blad wejscia*/
```

```
    }
    if (feof(f)) {
      fclose(f);
      return -3; /*nie znaleziono liczby*/
    }
    fscanf(f, "%*[^\n]"); /*pomin reszte wiersza*/
    line++;
  }

  fclose(f);
  return line;
}
```

Znakowe wejście-wyjście

Do wypisywania znaków służą funkcje
int fputc(intc c, FILE *stream);
int putc(int c, FILE *stream);
int putchar(int c);
putc i fputc robią dokładnie to samo, ale putc jest zazwyczaj implementowana jako makro. putchar jest normalnie implementowana jako makro.
`#define putchar putc((c), stdout)`
Jeśli podczas zapisu nastąpi błąd zapalany jest wskaźnik błędu dla danego strumienia i zwracana jest wartość EOF.
Do wczytywania znaków służą funkcje
int fgetc(FILE *stream);
int getc(FILE *stream);
int getchar(void);
int ungetc(int c, FILE *stream);
Różnica pomiędzy getc i fgetc jest podobna, jak pomiędzy

putc i fputc. Pierwsze trzy funkcje zachowują się tak samo, gdy wystąpi błąd. Po napotkaniu końca pliku zapalany jest wskaźnik końca pliku, a gdy wystąpi błąd wskaźnik błędu. Wartością funkcji jest wtedy EOF. Oto sposób wczytywania kolejnych znaków z pliku.

```
while ((ch = getc(fp)) != EOF) {
  ...
}
```

Uwaga: deklaruj ch jako int.
Funkcja ungetc "zwraca" strumieniowi wczytany znak. Oto przykład wczytania kolejnych cyfr, aż do napotkania znaku różnego od cyfry.

```
while (isdigit(ch = getc(fp))) {
  ...
}
ungetc(ch, fp);
```

Oto przykład programu kopiującego jeden plik na drugi.

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

main(int argc, char *argv[])
{
  FILE *source_fp, *dest_fp;
  int ch;

  if (argc != 3) {
```

```
    fprintf(stderr, "zla liczb argumentow\n");
    exit(EXIT_FAILURE);
  }

  if ((source_fp = fopen(argv[1], "rb")) == NULL) {
    fprintf(stderr, "nie mozna otworzyc %s\n",
    argv[1]);
    exit(EXIT_FAILURE);
  }

  if ((dest_fp = fopen(argv[2], "wb")) == NULL) {
    fprintf(stderr, "nie mozna otworzyc %s\n",
    argv[2]);
    exit(EXIT_FAILURE);
  }

  while ((ch = getc(source_fp)) != EOF)
    putc(ch, dest_fp);

  fclose(source_fp);
  fclose(dest_fp);
  return 0;
}
```

Biblioteka wejścia-wyjścia umożliwia wczytywanie i wypisywanie całych napisów. Do wypisywania służą funkcje
int fputs(const char *s, FILE *stream);
int puts(const char *s);
puts zawsze na końcu wypisuje `"\n"`, w przeciwieństwie do fputs. Obie funkcje zwracają EOF, gdy wystąpi błąd,

a nieujemną wartość w przeciwnym przypadku.
Do wczytywania napisów służą funkcje
char *fgeta(char *s, int n, FILE *stream);
char *gets(char *s);
Oto najczęstszy sposób użycia fgets
`fgets(str, sizeof(str), fp);`
gets nigdy nie zapamiętuje znaku końca wiersza, w przeciwieństwie do fgets.