Język ANSI C

część 6

biblioteka standardowa wejścia - wyjścia
1. sformatowane wejście - wyjście
2. rekordowe (binarne) wejście - wyjście

Jarosław Gramacki
Instytut Informatyki i Elektroniki

sformatowane wejście-wyjście

– z grupy funkcji "typu printf" tylko poniżej wymienione są zdefiniowane w standardzie ANSI
`printf(), sprintf(), fprintf(), snprintf()`

plus ekwiwalentne do nich funkcje, których nazwy rozpoczynają się na "v" (np. vsprintf). Szczegóły - patrz manuale

– kody formatujące !

– pełna lista wszystkich elementów kodów formatujących:
`man fprintf`

```
int printf(const char *format, ...);
  Writes its output into the standard output

int sprintf(char *str, const char *format, ...);
  Writes its output into a string

int snprintf(char *str, size_t size, const char *format, ...);
  Writes its output into the given string but never exceeding "size" chars

int fprintf(FILE *stream, const char *format, ...);
  Writes its output into a file
```

ten mechanizm języka C kiedyś omówimy

patrz też Wykład 1

sformatowane wejście-wyjście

– z grupy funkcji "typu scanf" tylko poniżej wymienione są zdefiniowane w standardzie ANSI
`scanf(), sscanf(), fscanf()`

plus ekwiwalentne do nich funkcje, których nazwy rozpoczynają się na "v" (np. vsscanf). Szczegóły - patrz manuale

– pełna lista wszystkich elementów kodów formatujących:
`man fscanf`

– podstawowe kody:

Type	**Format**
char	%c
short	%hd
int	%d or %i
long	**%ld**
long long	**%lld**
float	%f or %e
double	**%lf or %le**
long double	**%Lf or %Le**
string	%s

wyróżnione sprawiają czasami problemy (zapominamy o nich w sytuacjach koniecznych)

```
int scanf(const char *format, ...);
  reads  input from the standard input stream stdin

int fscanf(FILE *stream, const char *format, ...);
  reads input from the stream  pointer  stream

int sscanf(const char *str, const char *format, ...);
  reads its input from the character string pointed to by str
```


typy float oraz double

```
int main()
{
  double x;       // for float x; the described error does not exist!
  scanf("%f", &x);
  printf("x = %e", x);
  return 0;
}
```

– uwaga na `%f` w `scanf()` i `printf()`

– w przykładzie jak powyżej możemy otrzymać bardzo dziwne wyniki:
`0.6 Enter`
`x = 4.767030e-314`

– powód 1: `%f` w `scanf()` każe przechowywać daną w pamięci o adresie podanym przez `&x` jako zmienną pojedynczej precyzji. `scanf()` NIE zna typu tej zmiennej ale tylko jej ADRES

– powód 2: w `printf()` wartość konwertowana jest do postaci dziesiętnej (bo tak pojawia się na ekranie) przy założeniu, że `x` jest przechowywana w postaci `double`

– rozwiązanie problemu: `scanf("%lf", &x);`

– uwaga: `printf()` poprawnie zadziała tylko z `%f` (lub `%e`) gdyż ZAWSZE oczekuje on danej typu `double` i następuje automatyczna konwersja z `float` do `double`

– Aby w `printf()` daną `double` wyświetlić z maksymalną precyzją należy pamiętać o "ręcznym" ustawieniu ilości znaczących cyfr, tu: `printf("x= %21.15e", x);` Inaczej ilość cyfr odpowiadała będzie typowi `float`

format wyniku

- Wybór właściwego formatu w funkcji `printf()` jest ważny
- format decyduje m.in. o sposobie zaokrąglenia wyniku
- przykład: wybory do parlamentu niemieckiego w 1992
  `http://catless.ncl.ac.uk/Risks`
  Forum On Risks To The Public In Computers And Related Systems
- ... plus wiele innych spektakularnych błędów spowodowanych przez błędy w oprogramowaniu !

```
"...German elections are quite complicated to calculate. First, there is the 5%
clause: no party with less than 5% of the vote may be seated in parliament.
...
On Sunday the votes were being counted, and it looked like the Green party was
hanging on by their teeth to a vote percentage of exactly 5%.
...
After midnight (and after the election results were published) someone discovered
that the Greens actually only had 4,97% of the vote. The program that prints out
the percentages only uses one place after the decimal, and had *rounded the
count up* to 5%! This software had been used for *years*, and no one had thought
to turn off the rounding at this very critical (and IMHO very undemocratic)
region!..."
```


buforowanie

```
int main()
{
  double x;       // for float x; the error does not exist
  scanf("%f", &x);
  printf("x = %e", x);
  getchar();      // ???
  getchar();      // ???
  return 0;
}
```

- `getchar()` x 2. Pierwsza funkcja pobiera Enter pozostały w buforze (klawiatury) po `scanf()`, druga czeka na naciśniecie jakiegoś klawisza przed opuszczeniem programu
- kwestia bufora wyjścia
- `int fflush(FILE *stream);  // #include <stdio.h>`

```
OPIS
Funkcja fflush wymusza zapis wszystkich buforowanych danych dla danego
strumienia wyjściowego stream poprzez podległą strumieniowi funkcję zapisu.
Stan strumienia nie jest zmieniany, jest on nadal otwarty.

RETURN VALUE
Jeśli funkcja zakończyła się pomyślnie zwracane jest 0. W przeciwnym przypadku
zwracana jest wartość EOF, a zmienna globalna errno przyjmuje wartość
określającą rodzaj błędu.
```

- `fflush()` czyści jedynie bufor wyjściowy

kwestia obsługi buforów jest w ogólności dosyć skomplikowanym zagadnieniem

buforowanie

```
// dane umieszczane są w buforze. Po zakończeniu programu bufor jest czyszczony
// i wszystkie komunikaty pojawią się na ekranie
// (oczywiście ważna jest tu wielkość bufora - gdy się skończy, jest czyszczony)

int main()
{
  // etap 1 programu
  printf("#1 finished);

  // etap 2 programu
  printf("#2 finished);

  // etap 3 programu
  printf("#3 finished); }
```

```
// użycie fflush

int main()
{
  // etap 1 programu
  printf("#1 finished);   fflush(stdout);

  // etap 2 programu
  printf("#2 finished);   fflush(stdout);

  // etap 3 programu
  printf("#3 finished);   fflush(stdout);
}
```

- w praktyce `fflush` ma zastosowanie przy zapisie danych do plików dyskowych. Wtedy poprawne opróżnianie bufora jest b. ważne na przykład w przypadku awarii komputera na którym np. edytujemy plik

buforowanie

```
#include <stdio.h>
...
char user[100];
char oldpasswd[100];
char newpasswd[100];
...
printf("User name: ");
fflush(stdout);
gets(user);


printf("Old password: ");
fflush(stdout);
gets(oldpasswd);


printf("New password: ");
fflush(stdout);
gets(newpasswd);
...
```

the stream shall be flushed after an output operation and before performing an input operation !

```
#include <stdio.h>
char mybuffer[80];
int main()
{
   FILE * pFile;
   pFile = fopen ("example.txt","r+");
   if (pFile == NULL) perror ("Error opening file");
   else {
     fputs ("test",pFile);
     fflush (pFile);    // flushing required
     fgets (mybuffer,80, pFile);
     puts (mybuffer);
     fclose (pFile);
     return 0;
   }
}
```

scanf(), kilka przykładów

```
// Here are some common errors to avoid:

int integer;   short shortint;
char buffer[80];
char* str = buffer;       // pointer initialized with a table

// providing the variable instead of a pointer to it
scanf("%d", integer);     // wrong
scanf("%d", &integer);    // right

// providing a pointer to the wrong type (or a wrong format to the right
pointer)
scanf("%d", &shortint);   // wrong
scanf("%hd", &shortint);  // right

// providing a pointer to a string pointer instead of the string pointer
itself.(some people think "once &, always &)
scanf("%s", &str);        // wrong
scanf("%s", str);         // right

int main(void) {
 int i; char c;
 scanf("%d", &i);
 scanf("%c", &c);
 printf("%d %c\n", i, c); }
Assume you type 45\n in response to the first scanf. The 45 is copied into
variable i. When the program encounters the next scanf, the remaining \n
is quickly copied into the variable c. The fix is to put explicitely a \n,
like this:
...
scanf("%d\n", &i);
```

będzie o tym mowa jeszcze

scanf(), kilka ważnych uwag praktycznych

- zawsze należy stosować właściwy kod formatujący kiedy korzystamy ze `scanf()`. Funkcja z reguły nie domyśli się właściwego typu
- dla każdej wczytywanej zmiennej należy dostarczyć wskaźnik do odpowiedniego obszaru pamięci (jawnie przez & lub niejawnie)
- białe znaki (spacje) z przodu są zawsze ignorowane, ale oprócz `%c`
- wczytując dana `float` używaj `%f`, wczytując `double` używaj `%lf`
- `scanf()` ignoruje znaki nowej linii (przeskakuje je). Jednak gdy w kodzie formatującym użyjemy `\n` wymusimy obsługę znaku nowej linii
- nie należy używać obok siebie `scanf()` oraz `getchar()`. Uwaga ta nie dotyczy pary `printf()` i `putchar()`
- `scanf()` kończy skanowanie danych na pierwszej pozycji, która nie pasuje do wzorca oczekiwanego przez dany kod formatujacy. Przykładowo gdy `scanf()` oczekuje liczby, a użytkownik wprowadził 456x67, `scanf()` odmówi współpracy. Gdy podamy 456x, to wczyta tylko 456. Z tego powodu zawsze lepiej używać `sscanf()`

fprintf(), kilka przykładów

```
// writes formatted data to a file

#include <stdio.h>
#include <string.h>

void main(void)
   {
   FILE *fptr;
   char name[40];
   int code;
   float height;

   fptr = fopen( "agent.txt", "w" );
   do {
      printf( "Type name, code number, and height: " );
      scanf( "%s %d %f", name, &code, &height );
      fprintf( fptr, "%s %3d %5.2f", name, code, height );
      }
   while( strlen(name) > 1 );           // empty name?

   fclose ( fptr );
   }
```

– podobne funkcje: `snprintf()` oraz `sprintf()`

fprintf(), kilka przykładów

```
// reads formatted data from a file

#include <stdio.h>

void main(void)
   {
   FILE *fptr;
   char name[40];
   int code;
   float height;

   fptr = fopen("agent.txt", "r");
   while( fscanf(fptr, "%s %3d %5.2f", name, &code, &height) != EOF )
      printf("%s %5d %5.1f\n", name, code, height);
   fclose(fptr);
   }
```

musi odpowiadać kodom użytym w poprzednim przykładzie (jeśli maja ze sobą "współpracować")

do wydruku inny format

– podobnie będzie dla funkcji: `sscanf()`

```
  ...
  float f1, f2;
  int stat;

  stat = scanf( "%f-%f", &f1, &f2 );
  printf( "floats: %f, %f; stat: %d", f1, f2, stat );
 // scanf zwraca użyteczny status !
```

```
c:\Temp>Project1.exe
12-67
floats: 12.000000, 67.000000; stat: 2
c:\Temp>Project1.exe
12 67
floats: 12.000000, 0.000000; stat: 1
```

czy można napisać tak:
`scanf( "%5.2f", &f1 );` ???

2. rekordowe (binarne) wejście - wyjście

fwrite(), przykład

```
#include <stdlib.h>                  // for exit(), atof(), etc.

int main(void)  {
   struct
     {
      char name[40];
      int agnumb;
      double height;
     } agent;                        // structure variable

   char numstr[81];  FILE *fptr;

   if( (fptr = fopen( "agents.rec", "wb" )) == NULL )
      { printf("\nCan't open file agents.rec"); exit(1); }
   do
     {
       printf( "\nEnter name: " );       gets( agent.name );
       printf( "Enter number: " );       gets( numstr );
       agent.agnumb = atoi( numstr );
       printf( "Enter height: " );       gets( numstr );
       agent.height = atof( numstr );
       // write struct to the file
       fwrite ( &agent, sizeof(agent), 1, fptr );
       printf("Add another agent (y/n)? ");
      }
   while(getchar() == 'y');
   fclose(fptr);
   return(0);
}
```

nie znamy jeszcze struktur

inne wartości

fwrite(), przykład

- `fwrite()` zapisuje do strumienia `fptr` niesformatowany ciąg bajtów
- należy określić jaki obszar pamięci ma być zapisany (gdzie: począwszy od adresu `&agent` zapisz `sizeof(agent)` bajtów)
- należy określić ile bloków takich danych zapisać (tu: 1)
- plik należy otworzyć w trybie binarnym (aby nie doszło na niektórych platformach do niezamierzonej konwersji ew. bajtów o wartościach znaków końca wiersza)

fwrite(), plik wynikowy (z zapisaną jedną strukturą)

```
struct
     {
      char name[40];
      int agnumb;
      double height;
     } agent;


D:\temp>Project1.exe

Enter name: abc
Enter number: 8
Enter height: 0.1
Add another agent (y/n)? n

D:\temp>
```

o formacie zapisów liczb zmiennoprzecinkowych na razie nie mówimy

```
00000000: 61 62 63 00 18 FF 22 00|08 00 00 00 E0 FF 22 00 | abc■■"'"■■■■■ŕ'"■
00000010: 94 5C C3 77 70 20 C1 77|FF FF FF FF E3 C2 C2 77 | "\Ăwp Áw'''' ăÂÂw
00000020: 14 B8 C3 77 50 24 3E 00|08 00 00 00 08 00 00 00 | ■¸ĂwP$>■■■■■■■■■
00000030: 9A 99 99 99 99 99 B9 3F|                        | Šᵀᴴᵀᴴᵀᴴᵀᴴą?
```

```
00000000 00000000 00000000 00001000
00000000 00001000 00000000 00000000        // pierwsza zamiana
00001000 00000000 00000000 00000000        // druga zamiana
   08       00       00       00
```

little endian

format danych

– format danych binarnych w pamięci i na pliku

– little endian / big endian

```
// zapis jednaj liczby całkowitej na plik

int a = 8388608 + 1; // 2^23 + 2^0
FILE *fptr;

fptr = fopen( "file.bin","wb" );
fwrite ( &a, sizeof(a), 1, fptr );
```

Co się stanie gdy tu zmienimy na 2 ?

```
wynik działania (zawartość pliku szesnastkowo):
01 00 80 00

wpisaliśmy jednak "inną" liczbę

dwójkowa jej postać:
00000000 10000000 00000000 00000001              // 2^23 + 2^0

czyli na dysku powinno być:
00 80 00 01
```

– wewnętrzna architektura procesora wymaga zamiany kolejności bajtów. Dla liczby 4. bajtowej zamiana taka jest wykonywana podwójnie

– w takiej "zamienionej" postaci dane trafiają na plik dyskowy

```
00000000 10000000 00000000 00000001
00000000 00000001 00000000 10000000        // pierwsza zamiana
00000001 00000000 10000000 00000000        // druga zamiana
   01       00       80       00
```


fread(), przykład

```
#include <stdlib.h>

int main(void)    {
   struct {
      char name[40];
      int agnumb;
      double height;
      } agent;

   FILE *fptr;

   if( (fptr = fopen("agents.rec", "rb")) == NULL )
      { printf("Can't open file agents.rec"); exit(1); }

   while( fread ( &agent, sizeof(agent), 1, fptr) )     // dopóki prawda
      {
      // read the file
      printf("\nName: %s\n", agent.name);
      printf("Number: %03d\n", agent.agnumb);
      printf("Height: %.2lf\n", agent.height);
      }
   fclose(fptr);  return(0);    }
```

format danych zapisanych w poprzednim przykładzie i obecnie jest taki sam.

– `fread()` czyta ze strumienia `fptr` niesformatowany ciąg bajtów do pamięci

– należy określić gdzie do pamięci ma być zapisany ten strumień (gdzie: począwszy od adresu `&agent` zapisz `sizeof(agent)` bajtów)

– należy określić ile bloków takich danych zapisać (tu: 1)

operator sizeof()

– tablice, funkcje, operator sizeof()

– określić wielkość tablicy

```
#include <stdio.h>

void fun(int []);

int main()
{
  int MyTab[10];
  printf("size (of a global table)= %d\n", sizeof(MyTab));  // 40, right
  fun( MyTab );
  return 0;
}

void fun( int Table[] )
  {
    printf("size (inside a function) = %d\n", sizeof(Table));// 4, wrong
  }
```

– w pierwszym przypadku operator `sizeof` oblicza wielkość tablicy

– w drugim przypadku operator `sizeof` oblicza wielkość wskaźnika

fseek(), poruszanie się po pliku binarnym

```
#include <process.h>

int main(void)   {
   struct {
      char name[40];
      int agnumb;
      double height;
      } agent;

   FILE *fptr;
   int recno;                              // record number
   long int offset;                        // note: type long

   if( (fptr = fopen("agents.rec", "rb")) == NULL )
      { printf("\nCan't open file agents.rec"); exit(1); }


   printf("\nEnter record number: ");
   scanf("%d", &recno);
   offset = (recno-1) * sizeof(agent);     // calculate offset
   if( fseek( fptr, offset, 0) != 0 )      // go there
      { printf("\nCan't move pointer there."); exit(1); }

   fread( &agent, sizeof(agent), 1, fptr );      // read one record
   printf("Name: %s\n", agent.name);
   printf("Number: %03d\n",  agent.agnumb);
   printf("Height: %.2lf\n", agent.height);
   fclose(fptr);
   return(0);   }
```

przestaw wskaźnik plikowy na bajt pliku, od którego chcemy zacząć czytać ten plik

wskaźnik plikowy ustawiony, zacznij czytać dane do pamięci

poruszanie się po pliku, pozostałe funkcje

```
   ...
   if( fseek(fptr, offset, 0) != 0)        // go there
   ...


fseek( fptr, offset, 0)  - przesunięcie liczone od początku pliku
fseek( fptr, offset, 1)  - przesunięcie liczone od akt. poł. wskaźnika
fseek( fptr, offset, 2)  - przesunięcie liczone od końca pliku

lub

fseek( fptr, offset, SEEK_SET)     // see <stdio.h>
fseek( fptr, offset, SEEK_CUR)
fseek( fptr, offset, SEEK_END)
```

```
int fseek(FILE *stream, long offset, int whence);
long ftell(FILE *stream);         // return the current pointer position

void rewind(FILE *stream);
int fgetpos(FILE *stream, fpos_t *pos);
int fsetpos(FILE *stream, fpos_t *pos);
```

```
określ wielkość pliku

fseek(fptr, 0, 2);                // set pointer to 0 bytes from the end
FileSize = fteel(fptr);           // actual pointer position == file size
```


poruszanie się po pliku, przykład 2

```
// wczytaj plik do pamięci (do struktury)

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

typedef struct   {
    char *bufor
    long int wielkosc
  } structF;


int main(void) {
  long int dlugoscPliku;
  structF dane;
  ...
  fseek ( fp, 0 , SEEK_END );
  dlugoscPliku = ftell ( fp );
  fseek ( fp, 0 , SEEK_SET );

  dane.bufor = ( char * ) malloc ( sizeof(char) * dlugoscPliku);
  if ( !dane.bufor ) exit(1);           // błąd przydziału pamięci

  fread ( dane.bufor, sizeof(char), dlugoscPliku, fp);
  // pobranie pliku do pamięci
  dane.wielkosc = dlugoscPliku;

...
return(0);
}
```

alokacja pamięci na plik. Plik zapamiętamy w polu bufor struktury structF

lub: fread (dane.bufor, dlugoscPliku, 1, fp);