Funkcje o zmiennej liczbie argumentów

- Należy włączyć **`<stdarg.h>`**
- Przy deklaracji funkcji musi występować
  - co najmniej jeden ustalony parametr
  - oraz wielokropek (...)
- Jeden z ustalonych parametrów musi dostarczać informacji jak wiele parametrów jest przekazanych jako (...)
- Funkcja musi znać typ każdego argumentu
  - W przykładzie użyjemy jednego typu, więc nie będzie to problem
  - W innych przypadkach możemy użyć podejścia znanego np. z funkcji **`printf`**

Dostęp do argumentów

- **`stdarg.h`** zawiera cztery makra:
  - **`va_list`** :
    - typ wskaźnikowy
  - **`va_start()`** :
    - makro używane do inicjalizacji listy argumentów
  - **`va_arg()`** :
    - makro używane do dostępu do argumentów
  - **`va_end()`** :
    - makro wywoływane po pobraniu wszystkich argumentów

Dostęp do argumentów

- Każdy z poniższych kroków jest wymagany w funkcji przyjmującej zmienną liczbę argumentów:
  - Deklaracja wskaźnika do argumentu typu **`va_list`**.
    - używany do dostępu do poszczególnych argumentów (**`arg_ptr`**)
  - Wywołanie **`va_start()`**
    - podając jako argumenty **`arg_ptr`** i nazwę ostatniego ustalonego argumentu
    - makro inicjalizuje **`arg_ptr`** w taki sposób, że wskazuje na pierwszy opcjonalny argument
  - Wywołanie **`va_arg()`** przekazując **`arg_ptr`** i typ kolejnego argumentu
    - zwraca wartość kolejnego argumentu
    - jeżeli podano **`n`** argumentów, należy makro wywołać **`n`** razy
  - Wywołanie **`va_end()`**
    - z argumentem **`arg_ptr`**

Przykład

```
#include <stdio.h>
#include <stdarg.h>
float average (int num, ...);
int main ()
{
  float x;
  x = average (10, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
  printf ("The first average is %f.\n", x);
  x = average (5, 121, 206, 76, 31, 5);
  printf ("The second average is %f.\n", x);
  return 0;
}

float average (int num, ...)
{
  va_list arg_ptr;
  int count, total = 0;
  va_start (arg_ptr, num);

  for (count = 0; count < num; count++)
    total += va_arg (arg_ptr, int);

  va_end (arg_ptr);

  return ((float) total / num);
}
```

`string.h`

- **`#include <string.h>`**
- Funkcje dwóch rodzajów:
  - Funkcje operujące na pamięci - **`mem`**... **`()`**
    - Manipulacje na blokach pamięci określonego rozmiaru.
    - Traktowane jako tablice bajtów
  - Funkcje operujące na łańcuchach – **`str`**... **`()`**
    - Manipulacje na łańcuchach znaków zakonczonych znakiem NUL.

`string.h` - operacje na pamięci

- **`void *memset(void *p, int c, size_t n)`**
  - Wypełnia pierwsze **`n`** bajtów obszaru pamięci wskazywanego przez **`p`** bajtami o wartości **`c`**.
- **`void *memcpy(void *to, const void *from, size_t n)`**
  - Kopiuje **`n`** bajtów z obszaru **`from`** do obszaru **`to`**. Obszary nie mogą się nakładać.
- **`void *memmove(void *to, const void *from, size_t n)`**
  - Działa tak samo, jak funkcja wyżej, ale obszary mogą się nakładać.

Przykład użycia memmove

```
#include <stdio.h>
#include <string.h>
int
main ()
{
 char x[] = "Home Sweet Home";

 printf ("%s%s\n", "The string in array x before memmove is: ", x);
 printf ("%s%s\n",
     "The string in array x after memmove is: ",
     (char*)memmove(x, &x[5], 10));
 return 0;
}
```

```
The string in array x before memmove is: Home Sweet Home
The string in array x after memmove is: Sweet Home Home
```

string.h - operacje na pamięci

- **int memcmp(const void *p, const void *q, size_t n)**
  - Leksykograficzne porównanie sekwencji bajtów
  - Zwraca wartość mniejszą, równą lub większą od zera w zależności od wyniku porównania
- **void *memchr(const void *p, int c, size_t n)**
  - Przeszukuje pierwsze **n** bajtów obszaru pamięci wskazywanego przez **s** w poszukiwaniu pierwszego wystąpienia znaku **c**. Wartość **c** jest interpretowana jako **unsigned char**. Zwraca wskaźnik do pasującego bajtu albo **NULL** jeżeli poszukiwany znak nie występuje w podanym obszarze pamięci.

Przykład użycia memchr

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

int
main ()
{
 char s[] = "This is a string";

 printf ("%s%s\n", "The remainder of s after character 'r' is found is: ",
     (char *) memchr (s, 'r', 16));
 return 0;
}
```

```
The remainder of s after character 'r' is found is: ring
```

`string.h` - funkcje operujące na łańcuchach

- **`size_t strlen(const char *s)`**
  - Zwraca długość łańcucha **`s`**, nie wliczając kończącego znaku NUL
- **`char *strcpy(char *s1, const char *s2)`**
  - Kopiuje łańcuch wskazywany przez **`s2`** (razem z kończącym znakiem `\0') do tablicy wskazywanej przez **`s1`**. Obszary nie mogą się nakładać, a tablica **`s1`** musi być wystarczająco duża aby pomieścić kopiowany łańcuch.
- **`char *strncpy(char *s1, const char *s2, size_t n)`**
  - Podobnie, ale kopiowane jest nie więcej niż **`n`** znaków **`s2`**. Jeżeli pośród pierwszych **`n`** bajtów **`s2`** nie ma znaku NUL, wynik nie będzie zakończony znakiem NUL.

`string.h` - funkcje operujące na łańcuchach

- **`char *strcat(char *s1, const char *s2)`**
- **`char *strncat(char *s1, const char *s2, size_t n)`**
  - **`strcat()`** dołącza łańcuch **`s2`** do końca łańcucha **`s1`** nadpisując znak `\0' na końcu **`s1`** i dodając na końcu połączonych łańcuchów znak `\0'. Łańcuchy nie mogą się nakładać, a w buforze docelowym musi być wystarczająco dużo miejsca na wynik.
  - **`strncat()`** działa podobnie, ale jedynie pierwsze **`n`** znaków **`s2`** jest dołączanych do końca **`s1`**.
- **`int strcmp(const char *s1, const char *s2)`**
- **`int strncmp(const char *s1, const char *s2, size_t n)`**
  - Porównanie leksykograficzne
  - Zwraca wartość mniejszą, równą bądź większą od zera w zależności od wyniku porównania
  - **`strncmp()`** porównuje nie więcej niż **`n`** początkowych znaków

Przykład użycia strcat/strncat

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

int
main ()
{
 char s1[16] = "Happy ";
 char s2[] = "New Year ";
 char s3[22] = "";

 printf ("s1 = %s\ns2 = %s\n", s1, s2);
 printf ("strcat( s1, s2 ) = %s\n", strcat (s1, s2));
 printf ("strncat( s3, s1, 6 ) = %s\n", strncat (s3, s1, 6));
 printf ("strcat( s3, s1 ) = %s\n", strcat (s3, s1));
 return 0;
}
```

```
s1 = Happy
s2 = New Year
strcat( s1, s2 ) = Happy New Year
strncat( s3, s1, 6 ) = Happy
strcat( s3, s1 ) = Happy Happy New Year
```

Podział na tokeny - `strtok`

- **`char *strtok(char *s1, const char *s2)`**
  - Wywołanie inicjujące (z podanym **`s1`**) i następnie kolejne wywołania z **`s1=NULL`**
  - Zwraca kolejny token za każdym wywołaniem.
  - Niszczy oryginalny łańcuch

```
#include <stdio.h>
#include <string.h>
int
main ()
{
 char string[] =
 "This is a sentence with 7 tokens";
 char *tokenPtr;
 printf ("%s\n%s\n\n%s\n",
      "The string to be tokenized is:"
      , string, "The tokens are:");
 tokenPtr = strtok (string, " ");
 while (tokenPtr != NULL)
  {
   printf ("%s\n", tokenPtr);
   tokenPtr = strtok (NULL, " ");
  }
 return 0;
}
```

```
The string to be tokenized is:
This is a sentence with 7 tokens

The tokens are:
This
is
a
sentence
with
7
tokens
```

string.h - przeszukiwanie łańcuchów

- **`char *strchr(const char *s, int c)`**
- **`char *strrchr(const char *s, int c)`**
  - **`strchr()`** zwraca wskaźnik do pierwszego wystąpienia znaku **`c`** w łańcuchu **`s`**.
  - **`strrchr()`** zwraca wskaźnik do ostatniego wystąpienia znaku **`c`** w łańcuchu **`s`**.
  - Obie funkcje zwracają **`NULL`** jeżeli nie znajdą szukanego znaku.
- **`size_t strspn(const char *s,`**
  `const char *accept)`
- **`size_t strcspn(const char *s,`**
  `const char *reject)`
  - **`strspn()`** zwraca liczbę znaków początkowego segmentu **`s`** składającego się jedynie ze znaków z łańcucha **`accept`**.
  - **`strcspn()`** zwraca liczbę znaków początkowego segmentu **`s`**, który nie zawiera znaków z łańcucha **`reject`**.

Przykład użycia `strspn`

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

int
main ()
{
 const char *string1 = "The value is 3.14159";
 const char *string2 = "aehi lsTuv";

 printf ("%s%s\n%s%s\n\n%s\n%s%u\n",
     "string1 = ", string1, "string2 = ", string2,
     "The length of the initial segment of string1",
     "containing only characters from string2 = ",
     strspn (string1, string2));
 return 0;
}
```

```
string1 = The value is 3.14159
string2 = aehi lsTuv

The length of the initial segment of string1
containing only characters from string2 = 13
```

`string.h` - przeszukiwanie łańcuchów

- **`char *strpbrk(const char *s,`**
  `const char *accept)`
  - Zwraca wskaźnik do znaku w łańcuchu **`s`** któey jest jednym ze znaków w łańcuchu **`accept`** lub **`NULL`** jeżeli nie znaleziono takiego znaku.
- **`char *strstr(const char *haystack,`**
  `const char *needle)`
  - Znajduje pierwsze wystąpienie podłańcucha **`needle`** w łańcuchu **`haystack`**. Zwraca wskaźnik do początku podłańcucha lub **`NULL`** jeżeli nie znaleiono podłańcucha.

Funkcje konwersji łańcuchów

- Konwersja na wartości liczbowe, wyszukiwanie, porównywanie
- **`<stdlib.h>`**
- Większość funkcji pobiera jako argument **`const char *`**
- Nie modyfikują łańcucha

Funkcje konwersji łańcuchów

- **`double atof( const char *nPtr )`**
  - Konwertuje łańcuch na liczbę zmiennoprzecinkową (double)
  - Zwraca 0, jeżeli konwersja nie jest możliwa

- **`int atoi( const char *nPtr )`**
  - Konwertuje łańcuch na liczbę całkowitą
  - Zwraca 0, jeżeli konwersja nie jest możliwa

- **`long atol( const char *nPtr )`**
  - Konwertuje łańcuch na liczbę typu **`long int`**
  - Jeśli **`int`** i **`long`** mają ten sam rozmiar, **`atoi`** oraz **`atol`** są identyczne

Obsługa błędów w standardowej bibliotece C

- Większość funkcji bibliotecznych zwraca specjalną wartość oznaczającą, że operacja się nie powiodła. Ta specjalna wartość to -1, pusty wskaźnik lub stała taka jak EOF zdefiniowana specjalnie w tym celu.
- Aby zorientować się o jaki konkretnie błąd chodzi, należy odczytać kod błędu przechowywany w zmienej **`errno`**. Zmienna ta jest zadekalrowana w pliku nagłówkowym **`<errno.h>`**
  - Początkowa wartość **`errno`** po uruchomieniu programu wynosi zero.
  - Wiele funkcji bibliotecznych ustawia tę zmienną na pewną wartość niezerową po napotkaniu różnych rodzajów błędów. Błędy te są wyspecyfikowane w opisie każdej z funkcji.
  - Funkcje nie zmieniają **`errno`** kiedy ich wywołanie się powiedzie, dlatego wartość zmiennej **`errno`** po bezbłędnym wykonaniu funkcji nie musi być zerowa.
    - Nie należy używać **`errno`** w celu stwierdzenia, czy dane wywołanie funkcji się powiodło.

perror

- W celu poinformowania użytkownika o błędzie możemy użyć funkcji **perror**
  - **void perror(const char *s)**
- Funkcja drukuje komunikat na standardowym wyjściu błędów, opisujący błąd zawarty w zmiennej **errno**. Najpierw wypisywany jest łańcuch **s**, następnie dwukropek, spacja, komunikat i znak nowej linii.

```
#include <stdio.h>

int
main ()
{
  FILE *pFile;
  pFile = fopen ("unexist.ent", "rb");
  if (pFile == NULL)
    perror ("This error has occurred");
  else
    fclose (pFile);
  return 0;
}
```

```
This error has occurred: No such file or directory
```

Funkcje konwersji łańcuchów

- **`double strtod(const char *nPtr, char **endPtr)`**
  - Zwraca wynik konwersji pierwszego argumentu do **`double`**
  - Ustawia drugi argument na lokalizację pierwszego znaku za przekonwertowanym fragmentem łańcucha
  - Jeżeli nie udało się wykonać konwersji, zwracane jest zero, a w lokalizacji wskazywanej przez **`endPtr`** jest umieszczana wartość **`nPtr`**
  - Jeżeli wynik konwersji spowodowałby nadmiar lub niedomiar, wartość zmiennej **`errno`** jest ustawiana na **`ERANGE`**

  - strtod("123.4this is a test", &stringPtr);
    - Zwraca 123.4
    - stringPtr wskazuje na "this is a test"

strtod

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

int main (int argc, char *argv[])
{
  char *stringPtr;
  double v;
  int i;

  for (i = 1; i < argc; i++)
    {
      v = strtod (argv[i], &stringPtr);
      if (errno == ERANGE)
        printf ("Argument %d is an invalid number\n", i);
      else if (argv[i] == stringPtr)
        printf ("Argument %d is not a number\n", i);
      else
        printf ("Argument %d is a number %f\n", i, v);
    }
  return 0;
}
```

```
./strtod She is 17 now 1e123131231
Argument 1 is not a number
Argument 2 is not a number
Argument 3 is a number 17.000000
Argument 4 is not a number
Argument 5 is an invalid number
```

Funkcje konwersji łańcuchów

- **`long strtol(const char *nPtr, char **endPtr, int base )`**
  - Zwraca wynik konwersji pierwszego argumentu na **`long`**
  - Ustawia drugi argument na lokalizację pierwszego znaku za przekonwertowanym fragmentem łańcucha
  - Trzeci argument jest podstawą systemu liczbowego dla konwertowanej wartości
    - Dowolna liczba 2 - 36
    - 0 oznacza ósemkowy, dziesiętny lub szesnastkowy
  - W przypadku przepełnienia **`errno`** jest ustawiane na **`ERANGE`**
- **`unsigned long int strtoul(const char *nptr, char **endptr, int base)`**
  - Jak wyżej, dla **`unsigned long`**

Obsługa plików w języku C

- Dane przechowane w zmiennych, tablicach lub strukturach dynamicznych są nietrwałe
  - po zakończeniu programu wszystkie dane są utracone
- Do przechowywania danych (często dużych ilości) służą pliki
- Dane w postaci tekstowej (ASCII):
  - Znak ‘1’ ‘2’ ‘3’
  - Wartość dziesiętna ASCII 49 50 51
  - Wartość binarna ASCII 00110011 00110010 00110011
- Dane binarne: liczba dziesiętna 123 będzie przechowana jako liczba binarna 01111011, zajmując mniej miejsca niż postać tekstowa.

Pliki binarne

- Pliki binarne są bardzo podobne do tablic struktur, z tą różnicą, że są umieszczone na dysku, a nie w pamięci. Z tego powodu można stworzyć bardzo obszerne ich kolekcje, ograniczone jedynie przez wolne miejsce dostępne na dysku. Pliki są również trwałe - przechowywane po zakończeniu programu. Wadą plików binarnych w stosunku do tablic umieszczonych w pamięci jest ich powolność związana z czasem dostępu do dysku.

Pliki binarne

- Pliki binarne mają dwie podstawowe cechy odróżniające je od plików tekstowych:
  - Można przemieścić się bezpośrednio do dowolnej struktury w pliku, co pozwala na dostęp swobodny taki jak w przypadku tablicy; można zmienić zawartość struktury położonej w dowolnym miejscu pliku.
  - Pliki binarne zazwyczaj można szybciej zapisywać i odczytywać niż tekstowe, ponieważ binarny obraz rekordu jest przechowywany na dysku. W przypadku pliku tekstowego konieczna jest czasochłonna konwersja z/na postać tekstową.

Strumienie

- Operacje wejścia i wyjścia w języku C wykonywane są przy pomocy strumieni
- Strumień tekstowy jest sekwencją znaków podzielonych na linie
  - każda linia składa się z 0 lub większej liczby znaków i jest zakończona znakiem nowego wiersza ‘\n’
- Strumienie pozwalają na tworzenie kanałów komunikacyjnych między plikami i programami
  - Strumień może zostać podłączony do pliku poprzez otwarcie pliku, a odłączony poprzez zamknięcie pliku.

Strumienie

- Po rozpoczęciu wykonywania programu trzy pliki i skojarzone z nimi strumienie są podłączone do programu automatycznie
  - strumień standardowego wejścia
  - strumień standardowego wyjścia
  - strumień standardowego wyjścia błędów

Strumienie

- Standardowe wejście jest dołączone do klawiatury
  - umożliwia programowi odczyt danych z klawiatury
- Standardowe wyjście jest dołączone do ekranu
  - umożliwia programowi wypisywanie informacji na monitorze
- Standardowe wyjście błędów jest podłączone do ekranu
  - wszystkie komunikaty o błędach powinny być wypisywane na standardowe wyjście błędów
- Systemy operacyjne często pozwalają na przekierowanie tych strumieni do innych urządzeń

Strumienie

- Dostęp do plików realizowany jest za pośrednictwem wskaźnika do struktury **`FILE`**

- Struktura **`FILE`**
  - Zawiera informacje używane do przetwarzania pliku
  - Jest zadeklarowana w **`<stdio.h>`**
  - Programista nie musi znać szczegółowych informacji o zawartości struktury
  - Może być ona różna w różnych systemach

- Dostęp do standardowego wejścia, standardowego wyjścia i standardowego wyjścia błędów uzyskuje się za pośrednictwem wskaźników **`stdin`**, **`stdout`** i **`stderr`**

Otwieranie plików

- W celu uzyskania dostępu do zawartości pliku należy go otworzyć
- Otwarcie pliku:
  - `FILE *fopen (const char *name, const char *mode)`

- **`fopen`** ma dwa argumenty
  - **`name`**: łańcuch zawierający nazwę pliku
  - **`mode`**: łańcuch zawierający tryb otwarcia pliku - sposób zachowania przy otwarciu i rodzaj dostępu dla jakiego jest otwierany

- **`fopen`** zwraca wskaźnik do pliku używany następnie do dostępu do pliku

Otwieranie plików

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

- Dopuszczalne są następujące tryby:
  - r — Otwarcie pliku tekstowego do odczytu. Strumień wskazuje początek pliku.
  - w — Usunięcie zawartości pliku lub utworzenie nowego pliku do zapisu. Strumień wskazuje początek pliku.
  - a — Otwarcie do dopisywania (zapisu na końcu pliku). Jeśli plik nie istnieje, zostaje utworzony. Strumień wskazuje na koniec pliku.
  - r+ — Otwarcie pliku do odczytu i zapisu. Strumień wskazuje na początek pliku.
  - w+ — Otwarcie do odczytu i zapisu. Jeśli plik nie istnieje, zostaje utworzony, w przeciwnym wypadku jego zawartość zostaje usunięta. Strumień wskazuje początek pliku.
  - a+ — Otwarcie do odczytu i dopisywania (zapisu na końcu pliku). Jeśli plik nie istnieje, zostaje utworzony. Strumień wskazuje na koniec pliku.
  - b — Może być dodany do powyższych trybów w celu zaznaczenia, że pliki mają być otwarte w trybie binarnym, a nie tekstowym.

Tryby fopen

tryb	odczyt możliwy	zapis możliwy	skró-cenie pliku	utwo-rzenie pliku	pozycja począt-kowa
"r"	t	n	n	n	początek
"r+"	t	t	n	n	początek
"w"	n	t	t	t	początek
"w+"	t	t	t	t	początek
"a"	n	t	n	t	koniec
"a+"	t	t	n	t	koniec

Pierwsza linia oznacza, że "r" otworzy strumień do odczytu, nie otworzy strumienia do zapisu, nie usunie zawartości pliku, nie utworzy pliku jeżeli plik nie istnieje i ustawi pozycję strumienia na początek pliku.

Zamykanie plików

- Po zakończeniu korzystania z pliku należy go zamknąć
- Do zamknięcia pliku służy funkcja
  - `int fclose (FILE *fptr)`
- **`fclose`** pobiera jeden argument
  - wskaźnik do zamykanego pliku
- **`fclose`** zwraca zero, jeżeli nie wystąpiły błędy, w przeciwnym wypadku zwraca **`EOF`**

Niesformatowane pliki tekstowe

- **`<stdio.h>`** dostarcza wielu funkcji do odczytu i zapisu niesformatowanych plików tekstowych
  - **`int fgetc (FILE *stream)`**
    - czyta kolejny znak ze strumienia
    - zwraca znak (jako liczbę całkowitą) lub **`EOF`** w przypadku końca pliku lub błędu
  - **`int fputc (int c, FILE *stream)`**
    - zapisuje znak **`c`** do strumienia
    - zwraca zapisywany znak lub **`EOF`** w przypadku błędu

Przykład - kopiowanie pliku

```
#include <stdio.h>

int
main (int argc, char *argv[])
{
  FILE *ifp, *ofp;
  int c;
  if (argc != 3) /*check for valid input */
    printf ("Incorrect number of parameters");
  else if ((ifp = fopen (*++argv, "r")) == NULL)
    printf ("Cannot open %s for reading \n", *argv);
  else if ((ofp = fopen (*++argv, "w")) == NULL)
    printf ("Cannot open %s for writing \n", *argv);
  else
    while ((c = fgetc (ifp)) != EOF)
      fputc (c, ofp);
  fclose (ifp);
  fclose (ofp);
  return 0;
}
```

Niesformatowane pliki tekstowe

- **`char* fgets (char *s,int n,FILE *stream)`**
  - czyta najwyżej **`n-1`** kolejne znaki ze strumienia do tablicy **`s`**. Odczyt kończy się po napotkaniu końca pliku lub znaku przejścia do nowej linii. Jeżeli odczytano znak końca linii, umieszczany jest on w buforze. Za ostatnim znakiem w buforze wpisywany jest znak **`'\0'`**.
  - zwraca **`s`** lub **`NULL`** w przypadku końca pliku lub błędu

- **`int fputs (const char *s, FILE *stream)`**
  - zapisuje łańcuch **`s`** do strumienia,
  - zwraca **`EOF`** w przypadku błędu

Niesformatowane pliki tekstowe

- Pozyższe funkcje mogą być używane w podobny sposób, jak funkcje czytające i zapisujące znaki i łańcuchy z klawiatury i na ekran
  - **`int fgetc (FILE *stream)`**
    - jest podobna do **`getchar()`**
  - **`int fputc (int c, FILE *stream)`**
    - jest podobna do **`putchar(int)`**
  - **`char *fgets (char *s,int n,FILE *stream)`**
    - jest podobna do **`gets(char* )`** (nigdy nie używać **`gets()`**!)
  - **`int fputs (const char *s, FILE *stream)`**
    - jest podobna do **`puts(const char*)`**

Przykład - konwersja wielkości liter w pliku

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

int
main ()
{
  char szLine[1000];
  char *p = 0;
  FILE *fileIn, *fileOut;
  fileIn = fopen ("mary.txt", "r");
  if (!fileIn) {
      fprintf (stderr, "Cannot open input file\n");
      exit (1); };
  fileOut = fopen ("mary.new", "w");
  if (!fileOut) {
      fprintf (stderr, "Cannot open output file\n");
      exit (2); };
  while (fgets (szLine, 1000, fileIn))
    {
      for (p = szLine; *p; ++p)
        *p = toupper (*p);
      fputs (szLine, fileOut);
    }
  fclose (fileOut);
  fclose (fileIn);
  return 0;
}
```

Sformatowane pliki tekstowe

- Do odczytu i zapisu do sformatowanych plików tekstowych używa się poniższych funkcji:
  - `int fprintf (FILE *fptr, const char* fmt,...)`
  - `int fscanf (FILE *fptr, const char* fmt, ... )`
- Powyższe funkcje działają identycznie jak **`printf`** i **`scanf`**, ale wyjście/wejście odnosi się do pliku reprezentowanego przez wskaźnik **`fptr`**

Funkcja `feof`

- Funkcja **`feof`** sprawdza, czy napotkano koniec pliku
- **`int feof(FILE *fptr)`**

- **`feof`** pobiera wskaźnik do **`FILE`**
- **`feof`** zwraca wartość niezerową w przypadku końca pliku lub zero w przeciwnym przypadku

- Używana podczas odczytu z pliku w celu sprawdzenia, czy osiągnięto koniec pliku

Problem zarządzania kontami

- Rozważmy prosty system księgowy do rozliczeń z klientami firmy
- Dla każdego klienta przechowujemy następujące informacje
  - numer konta klienta
  - nazwisko klienta
  - saldo konta klienta
- Te informacje tworzą rekord klienta, który musi być przechowywany przez program

Przetwarzanie pliku tekstowego

- Załóżmy, że do przechowywania informacji o kontach klientów uzyjemy pliku tekstowego
- Będziemy używać funkcji formatowanego wejścia/wyjścia w celu odczytu i zapisu do pliku
  - **`fscanf`** i **`fprintf`**
- Operacje na plikach które muszą być zaimplementowane to
  - odczyt z pliku
  - wyszukiwanie rekordów w pliku
  - dodawanie nowych klientów do pliku
  - uaktualnianie rekordów klientów w pliku

Zapisywanie do pliku tekstowego

- Zapisywanie rekordu do pliku:

```
fprintf (fptr,"%d %s %.2f\n",
          accNo,name,balance);
```

- gdzie następujące zmienne zostały zadeklarowane i odpowiednio zainicjalizowane

```
int   accNo;
char  name[21];
float balance;
```

- **`fprintf`** zwraca liczbę zapisanych znaków lub liczbę ujemną w przypadku wystąpienia błędu

Przykład zapisu do pliku tekstowego

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

int main (int argc, char *argv[])
{
  FILE *fptr;
  int accNo = 1234;
  char names[][21] = { "Achison", "Agnew", "Barry", "Cunningham", "White" };
  float balance;
  int i;
  if ((fptr = fopen ("clients.dat", "w")) == NULL)
    {
      perror ("Error opening clients.dat");
      exit (1);
    }
  srand (time (NULL));
  for (i = 0; i < sizeof (names) / sizeof (names[0]); i++)
    {
      balance = (1000.0 * rand () / (RAND_MAX + 1.0)) - 500.0;
      /* random balance in range [-500,500[ */
      if (fprintf (fptr, "%d %s %.2f\n", accNo, names[i], balance) < 0)
        {
          perror ("Error writing data to clients.dat");
          exit (2);
        }
      accNo += 16;
    }
  if (fclose (fptr) != 0)
    {
      perror ("Error closing clients.dat");
      exit (3);
    }
  return 0;
}
```

Odczyt z pliku tekstowego

- Odczyt rekordu z pliku:
- **`fscanf (fptr,"%d %20s %f",`**
  `&accNo,name, &balance);`

- gdzie następujące zmienne zostały zadeklarowane i odpowiednio zainicjalizowane

```
int   accNo;
char  name[21];
float balance;
```

- **`fscanf`** zwraca liczbę przypisanych pól poprawnie odczytanych z pliku lub **`EOF`**, jeżeli błąd lub koniec pliku wystąpił przed wczytaniem pierwszego znaku
- **`rewind(FILE* ptr)`** - przesuwa bieżącą pozycję w pliku na jego początek.

Przeszukiwanie pliku

- Znalezienie konkretnego rekordu wymaga
  - sekwencyjnego odczytu od początku pliku
  - testowania każdego rekordu w celu sprawdzenia, czy spełnia warunki wyszukiwania
  - przesunięcia bieżącej pozycji w pliku na jego początek w celu umożliwienia kolejnego wyszukiwania

- Przykład:
  - Wypisz informacje o wszystkich kontach z ujemnym saldem

Przeszukiwanie pliku tekstowego

```
while (1)
  {
    int args = fscanf (fptr, "%d %20s %f", &accNo, name, &balance);
    if (args == EOF)
      break;
    if (args != 3)
      {
        fprintf (stderr, "Error in clients.dat");
        exit (2);
      }
    if (balance < 0)
      printf ("%6d %30s %6.2f\n", accNo, name, balance);
  }
rewind (fptr);
```

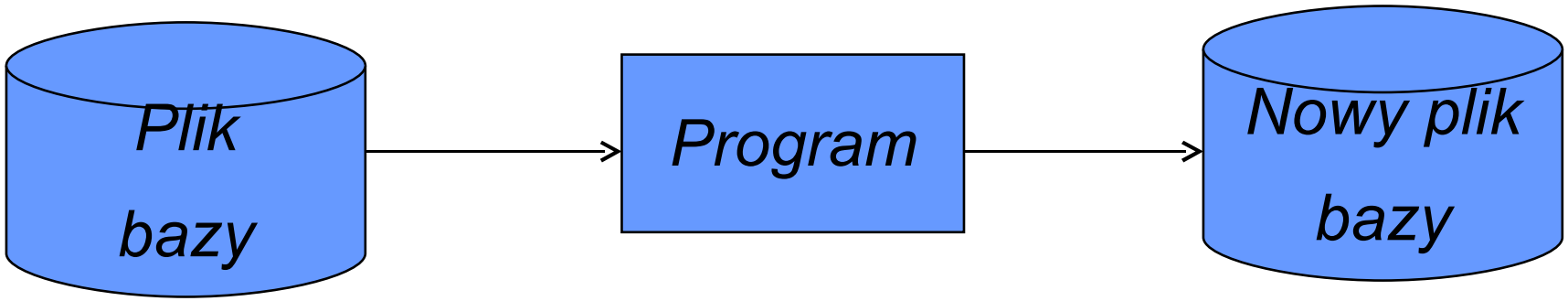
Uaktualnianie rekordów w pliku tekstowym

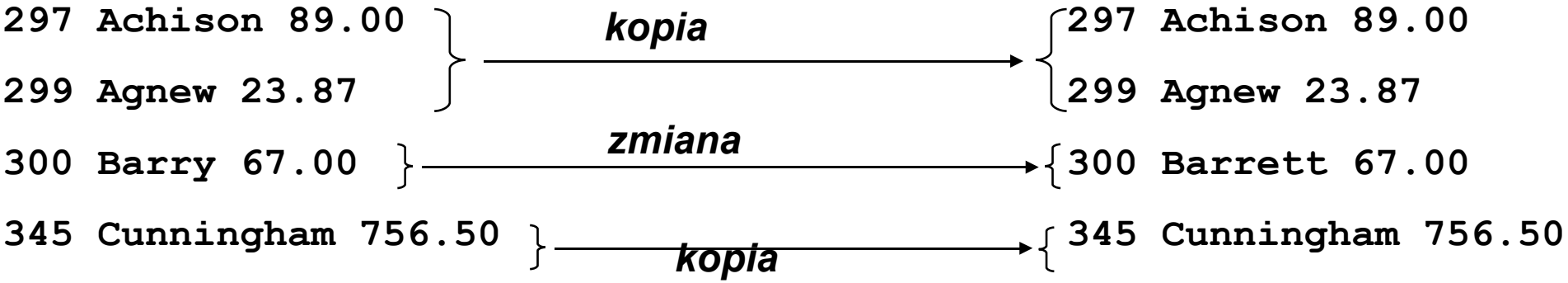
- Plik tekstowy zawiera rekordy o zmiennej długości
  - **`300 Barry 67.00`**
  - **`345 Cunningham 756.50`**
- Rekord nie może być zmodyfikowany "w miejscu", bo może to uszkodzić inne dane w pliku
- Rozważmy zmianę Barry na Barrett
  - **`300 Barry 67.00 345 Cunningham 756.50 ...`**

  zmienia się na
  - **`300 Barrett 67.0045 Cunningham 756.50`**

Uaktualnianie rekordów w pliku tekstowym

- Rekordy nie mogą być uaktualniane "w miejscu"
- Należy przepisać cały plik od nowa





Przykład uaktualniania pliku tekstowego

```
while (1)
  {
    char *tname;
    int args = fscanf (infile, "%d %20s %f", &accNo, name, &balance);
    if (args == EOF)
      break;
    if (args != 3)
      {
        fprintf (stderr, "Error in clients.dat");
        exit (3);
      }
    if (strcmp (name, "Barry"))
      tname = name;
    else                          /* replace Barry with Barrett */
      tname = "Barrett";
    if (fprintf (outfile, "%d %s %.2f\n", accNo, tname, balance) < 0)
      {
        perror ("Error writing data to clients.new");
        exit (4);
      }
  }
```

Ustawianie bieżącej pozycji w pliku

- Funkcja **`fseek`** ustawia bieżącą pozycję w pliku
- Kolejne odczyty i zapisy będą dotyczyć danych w pliku zaczynających się od nowej pozycji
  - **`int fseek (FILE *fp, long offset,int origin);`**

- **`fseek`** pobiera 3 argumenty
  - **`fp`** wskaźnik do rozważanego pliku
  - **`offset`** jest pozycją w pliku do jakiej chcemy się przemieścić
  - **`origin`** wskazuje względem czego pozycja jest liczona

Ustawianie bieżącej pozycji w pliku

- **`origin`** może przyjąć jedną z wartości
  - **`SEEK_SET`** - początek pliku
  - **`SEEK_CUR`** - bieżąca pozycja w pliku
  - **`SEEK_END`** - koniec pliku

- Zmiana pozycji w pliku może być konieczna w pewnych przypadkach, kiedy wykonywane jest wiele operacji na pliku:
  - np. wyszukiwanie rekordu w pliku może pozostawić bieżącą powycję w środku pliku
  - dopisanie nowego rekordu do pliku wymaga przesunięcia bieżącej pozycji na koniec pliku

Ustawianie bieżącej pozycji w pliku

- Funkcja **`rewind`** ustawia wskaźnik bieżącej pozycji na początek pliku
  - **`void rewind (FILE *fp)`**
  - **`rewind(fp)`**
    jest równoważne
  - **`fseek(fp,0,SEEK_SET)`**

Pliki binarne

- Sformatowane pliki tekstowe
  - zawierają rekordy zmiennej długości
  - muszą być przetwarzane sekwencyjnie, od początku pliku, w celu dostępu do określonego rekordu

- Pliki binarne
  - zawierają rekordy o stałej długości
  - potrzebne rekordy mogą być przetwarzane bezpośrednio

- Pliki binarne są odpowiednie do interaktywnego przetwarzania transakcji
  - np. rezerwacja miejsc w samolotach, przetwarzanie zamówień, systemy bankowe

Pliki binarne

- Podstawowe cechy plików binarnych:
  - Pliki binarne są łatwe do odczytu przez komputer, ale trudne do odczytu przez człowieka w przeciwieństwie do plików tekstowych
  - Pliki binarne mogą być dostępne bezpośrednio (dostęp swobodny)
  - Rekord w pliku binarnym jest tworzony z wykorzystaniem struktury
  - Są bardziej wydajne niż pliki tekstowe, gdyż konwersja z/do postaci ASCII nie jest potrzebna
  - Nie mogą być łatwo odczytywane przez inne programy nie napisane w języku C

Zapis do pliku binarnego

- Do zapisu do pliku binarnego służy funkcja **`fwrite`**
  - **`size_t fwrite (void *ptr, size_t size, size_t n, FILE *fptr);`**
  - **`fwrite`** zapisuje z tablicy **`ptr`**, **`n`** obiektów o rozmiarze **`size`** do pliku wskazywanego przez **`fptr`**
  - **`fwrite`** zwraca liczbę zapisanych obiektów
    - która może być mniejsza niż **`n`** w przypadku błędu

Odczyt z pliku binarnego

- Do odczytu z pliku binarnego służy funkcja **`fread`**
  - **`size_t fread (void *ptr, size_t size, size_t n, FILE *fptr);`**
  - **`fread`** odczytuje **`n`** obiektów rozmiaru **`size`** z pliku wskazywanego przez **`fptr`** i umieszcza je w tablicy **`ptr`**
  - **`fread`** zwraca liczbę przeczytanych obiektów
    - która może być mniejsza niż żądana
    - w celu odróżnienia końca pliku i błędu odczytu konieczne jest wywołanie **`feof()`**

Problem zarządzania kontami

- Rozważmy prosty system księgowy do rozliczeń z klientami firmy
- Dla każdego klienta przechowujemy następujące informacje
  - numer konta klienta
  - nazwisko klienta
  - saldo konta klienta
- Te informacje tworzą rekord klienta, który musi być przechowywany przez program

Przetwarzanie pliku binarnego

- Tym razem będziemy przechowywać informacje w pliku binarnym
- Operacje na plikach które należy zaimplementować obejmują
  - wyszukiwanie i odczyt poszczególnych rekordów z pliku
  - dodawanie informacji o nowym kliencie do pliku
  - uaktualnianie rekordów w pliku
  - sekwencyjny odczyt rekordów z pliku

Struktura rekordu

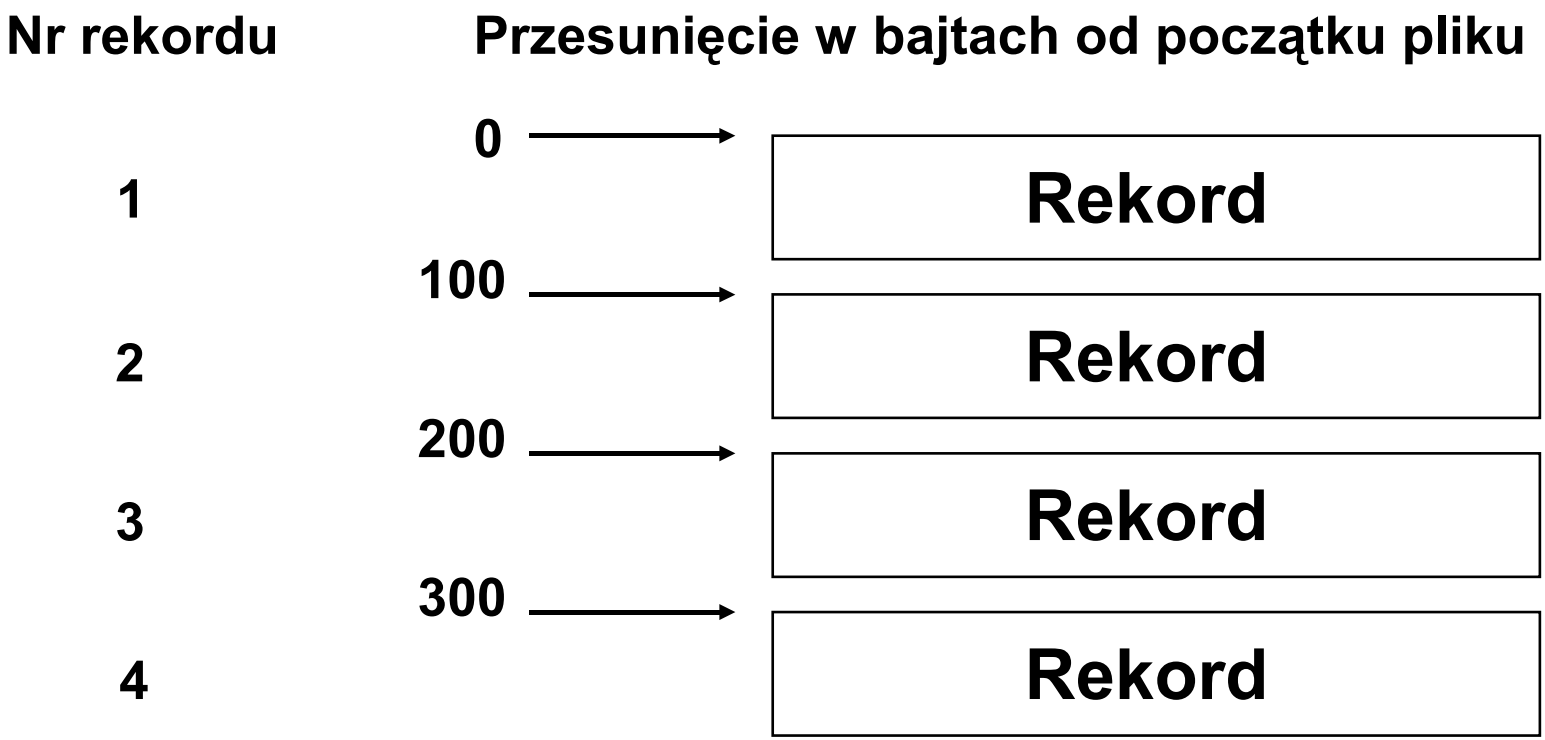
- Zdefiniujmy typ reprezentujący rekord w pliku:

```
typedef struct {
char  name[21];
float balance;
} CustAccount;
```

- Zauważmy, że numer konta jest określony przez pozycję w pliku i nie musimy przechowywać go w rekordzie

Struktura pliku z kontami

- Dostęp do poszczególnych rekordów w pliku binarnym polega na dostępie do ustalonej liczby bajtów w konkretnym miejscu pliku. Przy założeniu, że pojedynczy rekord zajmuje 100 bajtów, otrzymujemy poniższą ilustrację:



Struktura pliku z kontami

- W przypadku plików binarnych:
- Rekordy muszą być przechowywane w kolejności rosnących kluczy

- Pozycja rekordu jest określona przez klucz danych w tym rekordzie
  - np. rekord o numerze konta **`accNo`** jest przechowywany w poczynając od bajtu

```
(accNo-1) * sizeof(CustAccount)
```

- Plik binarny należy na początku zainicjalizować pustymi rekordami

Tworzenie pliku binarnego

- Otwarcie pliku do zapisu

```
fp= fopen ("clients.dat", "wb");
```

- Utworzenie zmiennej CustAccount inicjalizowanej jako pusty rekord

```
CustAccount cust={"",0.0};
```

- Pętla zapisująca n pustych rekordów do pliku

```
for (i=0;i<n;i++)
   fwrite (&cust,sizeof(CustAccount),1, fptr);
```

Ustawianie pozycji w pliku

- w celu ustawienia pozycji w pliku na początek odpowiedniego rekordu przed zapisem lub odczytem użyjemy funkcji **`fseek`**

  - **`int fseek (FILE *fp, long offset,int origin)`**

  - Nowa pozycja w pliku wskazuje na **`offset`** bajtów względem pozycji reprezentowanej przez **`origin`**
  - **`fseek`** zwraca wartość niezerową w przypadku błędu
  - Ustawianie pozycji w pliku w celu dostępu do rekordu dla numeru konta **`accNo`**

```
fseek (fptr,(accNo-1)*sizeof(CustAccount),
  SEEK_SET)
```

Przykład zapisu do pliku binarnego

```
for (i = 0; i < ACCOUNTS; i++)
  {
    if (fwrite (&cust, sizeof (CustAccount), 1, fptr) != 1)
      {
        perror ("Error during initialization of clients.dat");
        exit (2);
      }
  }
srand (time (NULL));
for (i = 0; i < sizeof (names) / sizeof (names[0]); i++)
  {
    cust.balance = (1000.0 * rand () / (RAND_MAX + 1.0)) - 500.0;
   /* random balance in range [-500,500[ */
    strcpy (cust.name, names[i]);
    if (fseek (fptr, (accNo - 1) * sizeof (CustAccount), SEEK_SET))
      {
        perror ("Error seeking in clients.dat");
        exit (3);
      }

    if (fwrite (&cust, sizeof (CustAccount), 1, fptr) != 1)
      {
        perror ("Error during initialization of clients.dat");
        exit (2);
      };

    accNo += 16;
  }
```

Przykład wyszukiwania w pliku binarnym

```
printf ("Accounts with debit balance \n\n");
for (i = 0; i < ACCOUNTS; i++)
    {
      if (fread (&cust, sizeof (CustAccount), 1, fptr) != 1)
          {
            perror ("Error reading from clients.dat");
            exit (2);
          }
      if ((cust.balance < 0) && (cust.name[0]))
        printf ("%6d %30s %6.2f\n", i + 1, cust.name, cust.balance);
    }

rewind (fptr);

printf ("\n\nAccounts with credit balance \n\n");
for (i = 0; i < ACCOUNTS; i++)

  {
    if (fread (&cust, sizeof (CustAccount), 1, fptr) != 1)
        {
          perror ("Error reading from clients.dat");
          exit (2);
        }
    if ((cust.balance >= 0) && (cust.name[0]))
      printf ("%6d %30s %6.2f\n", i + 1, cust.name, cust.balance);
  }
```

Uaktualnianie pliku binarnego

- Aby uaktualnić rekord w pliku binarnym
  - Czytamy rekord z pliku
  - Zmieniamy odpowiednie pola
  - Zapisujemy rekord z powrotem na odpowiedniej pozycji w pliku

```
for (i = 0; i < ACCOUNTS; i++)
  {
    if (fread (&cust, sizeof (CustAccount), 1, fptr) != 1)
      {
        perror ("Error reading from clients.dat");
        exit (2);
      }
    if (!strcmp (cust.name, "Barry"))
      {
        strcpy (cust.name, "Barrett");
        if (fseek (fptr, i * sizeof (CustAccount), SEEK_SET))
          {
            perror ("Error seeking in clients.dat");
            exit (3);
          }

        if (fwrite (&cust, sizeof (CustAccount), 1, fptr) != 1)
          {
            perror ("Error reading from clients.dat");
            exit (4);
          }
        break;
      }
  }
```