Politechnika Wrocławska

Bardzo szybkie podsumowanie: wykład 5

wer. 7 z drobnymi modyfikacjami!

Wojciech Myszka

2020-05-12 08:51:42 +0200

HR EXCELLENCE IN RESEARCH

Uwagi

1. Obowiązuje cały materiał!
2. Tu tylko podsumowanie.

Politechnika Wrocławska

Formatowane (tekstowe) wejście/wyjście. Binarne wejście/wyjście.

wer. 10 z drobnymi modyfikacjami!

Wojciech Myszka

2019-05-21 21:01:05 +0200

HR EXCELLENCE IN RESEARCH

Część I

Formatowane (tekstowe) wejście/wyjście

Otwarcie pliku I

1. Przed skorzystaniem z pliku należy go „otworzyć" (*open*).
2. **stdin**, **stdout**, **stderr** otwarte są w trybie tekstowym.
3. Standardowa biblioteka (stdio.h) zawiera trzy funkcje fopen, freopen i fclose .
4. Pierwsze dwie służą do „skojarzenia" (powiązania) pliku z odpowiednią strukturą danych.
5. Ostatnia — powiązanie to likwiduje.
6. Wszystkie czynności związane z otwieraniem i zamykaniem plików realizowane są przez System Operacyjny.

Operację otwarcia pliku (ponownego otwarcia) realizują funkcje **fopen** i **freopen**:

Otwarcie pliku II

```
#include <stdio.h>
FILE *fopen(const char *filename, const char *mode);
FILE *freopen(const char *filename, const char *mode,
              FILE *stream);
```

Otwarcie pliku III

Użycie:

```
FILE *fp;
...
fp = fopen("plik.txt", "w");
...
fp = freopen("plik.txt", "r", fp);
```

- Pierwszy parametr (filename) to wskaźnik do tablicy tekstowej (na przykład stałej) zawierającej nawę pliku.
- Parametr drugi to ciąg znaków (lub zmienna) zawierająca w sposób symboliczny określenie trybu dostępu:
  - **r** otwórz plik **tekstowy** do odczytu (plik musi istnieć)
  - **w** otwórz plik **tekstowy** do zapisu (niszcząc jego zawartość jeżeli już istnieje)

Politechnika Wrocławska

Otwarcie pliku IV

- **a** (*append*); otwórz plik **tekstowy** do dopisywania, jeżeli plik nie istnieje — zostanie utworzony.
- **r+** otwarcie pliku **tekstowego** do zapisu i odczytu (plik powinien już istnieć!)
- **w+** otwórz istniejący plik (kasując jego zawartość) lub utwórz nowy plik **tekstowy** w trybie do zapisu i odczytu
- **a+** otwórz plik **tekstowy** na końcu w trybie do zapisu i odczytu

Otwarcie pliku V

Przykład

```
#include <stdio.h>
...
FILE *pp;
...
pp = fopen("Ala.txt", "r");
if (pp == NULL)
    { /* Błąd! */}
...
fscanf(pp, "%d", &i);
```

- Trzeci parametr (stream — tylko w przypadku funkcji reopen) zawiera wskaźnik do struktury danych opisujących strumień danych.

Otwarcie pliku VI

1. Funkcja **fopen()** otwiera plik o podanej nazwie we wskazanym trybie, tworzy strukturę danych opisującą go i zwraca do niej adres. W przypadku błędu — zwraca NULL.
2. Funkcja **freopen()** najpierw zamyka otwarty wcześniej plik i otwiera go ponownie we wskazanym trybie, zwracając wskaźnik do struktury danych opisujących strumień. W przypadku błędu — zwraca NULL.
3. Funkcja **fclose()** zamyka wskazany plik. W przypadku błędu — zwraca NULL.

```
#include <stdio.h>
...
int fclose(FILE *stream);
```

stream to wskaźnik do struktury danych opisujących strumień danych. Przykład:

Otwarcie pliku VII

```
...
fclose( fp );
...
```

Odczyt formatowany I

1. Funkcje typu **scanf**

```
#include <stdio.h>
int fscanf(FILE *stream, const char *format, ...)
    ;

int scanf(const char *format, ...);

int sscanf(const char *s, const char *format,
    ...);
```

Dane czytane są ze wskazanego strumienia (stream) i interpretowane zgodnie z użytym formatem. W formacie powinna wystąpić wystarczająca liczba specyfikacji aby dostarczyć dane dla wszystkich argumentów.

Politechnika Wrocławska

Odczyt formatowany II

- fscanf zwraca liczbę przeczytanych wartości lub wartość EOF
- scanf jest odpowiednikiem fscanf, ale dotyczy strumienia stdin
- sscanf jest odpowiednikiem fscanf, z tym, że dane „czytane" są z tablicy znakowej; dotarcie do końca tabeli jest równoważne z dotarciem do końca pliku

Wejście: Specyfikacja formatu I

Na specyfikację formatu składają się:

- odstępy; Wystąpienie w formacie „białego znaku" (*white space*) powoduje, że funkcje z rodziny scanf będą odczytywać i odrzucać znaki, aż do napotkania pierwszego znaku nie będącego białym znakiem.
- znaki (różne od % i odstępów). Jeżeli taki znak wystąpi w specyfikacji musi się pojawić w strumieniu wejściowym na odpowiednim miejscu!
- specyfikacje konwersji (rozpoczynające się znakiem %)

Po znaku % wystąpić może:

- nieobowiązkowy znak $*$ (oznaczający, że zinterpretowana wartość ma być zignorowana)
- nieobowiązkowa specyfikacja długości pola (określa maksymalną liczbę znaków pola)

Wejście: Specyfikacja formatu II

- jeden z nieobowiązkowych znaków **h**, **l** (mała litera „el") lub **L** mówiących jak ma być interpretowana czytana wielkość (h — **short**, l — **long** albo **double** w przypadku **float**, ll – **long long**, L — **long double**),
- znak określający typ konwersji.

Po znaku procent (%) wystąpić może jeden ze znaków

- d liczba całkowita,
- i liczba całkowita (można wprowadzać wartości szesnastkowe — jeżeli poprzedzone znakami 0x lub ósemkowe jeżeli poprzedzone 0; 031 czytane z formatem %d to 31 a z formatem %i to 25),
- o liczba całkowita kodowana ósemkowo,
- u liczba całkowita bez znaku (**unsigned int**),
- x liczba całkowita kodowana szesnastkowo,

Wejście: Specyfikacja formatu III

- a, e, f, g liczba typu float; można również wczytać w ten sposób nieskończoność lub wartość Not a Number (NaN),
- s ciąg znaków (na końcu zostanie dodany znak zero),
- c znak (lub ciąg znaków gdy wyspecyfikowano szerokość pola), nie jest dodawane zero na końcu,
- [odczytuje niepusty ciąg znaków, z których każdy musi należeć do określonego zbioru, argument powinien być wskaźnikiem na **char**,
- n do zmiennej odpowiadającej tej specyfikacji konwersji wpisana zostanie liczba znaków przetworzonych przez fscanf,
- p w zależności od implementacji — służy do wprowadzania wartość wskaźników,
- % znak %.

Wejście: Specyfikacja formatu IV

Format „[" — Po otwierającym nawiasie następuje ciąg określający znaki jakie mogą występować w odczytanym napisie i kończy się on nawiasem zamykającym tj.]. Znaki pomiędzy nawiasami (tzw. *scanlist*) określają możliwe znaki, chyba że pierwszym znakiem jest ^ — wówczas w odczytanym ciągu znaków mogą występować znaki nie występujące w scanlist. Jeżeli sekwencja zaczyna się od [] lub [^] to ten pierwszy nawias zamykający nie jest traktowany jako koniec sekwencji tylko jak zwykły znak. Jeżeli wewnątrz sekwencji występuje znak − (minus), który nie jest pierwszym lub drugim jeżeli pierwszym jest ^ ani ostatnim znakiem zachowanie jest zależne od implementacji.

Wejście: Specyfikacja formatu V

Przykład

```
#include <stdio.h>
int main(void)
{
    int x, y, n;
    x = scanf("%*d␣%d␣%n", &y, &n);
    printf("y=␣%d,␣n=␣%d\n", y, n);
    printf("x=␣%d\n", x);
    return 0;
}
```

Wejście: Specyfikacja formatu VI

Dane i wyniki

```
10 20 30 40
y= 20, n= 6
x= 1

1␣␣␣␣␣2222␣ala␣ma␣kota
y=␣2222,␣n=␣11
x=␣1
```

Wejście: Specyfikacja formatu VII

Kolejny przykład

```
#include <stdio.h>
int main(void)
{
    int x, y, z, n;
    x = scanf("%4d␣%5d␣%n", &y, &z, &n);
    printf("y=␣%d,␣z=␣%d,␣n=␣%d\n", y, z, n);
    printf("x=␣%d\n", x);
    return 0;
}
```

Wejście: Specyfikacja formatu VIII

Dane i wyniki

```
1234567890
y= 1234, z= 56789, n= 9
x= 2
```

Wejście: Specyfikacja formatu IX

I jeszcze jeden przykład

```
#include <stdio.h>
int main(void)
{
    int x, y, z, n;
    x = scanf("%4d␣a␣%5d␣%n", &y, &z, &n);
    printf("y=␣%d,␣z=␣%d,␣n=␣%d\n", y, z, n);
    printf("x=␣%d\n", x);
    return 0;
}
```

Wejście: Specyfikacja formatu X

Dane i wyniki

```
123b123b
y=␣123,␣z=␣32767,␣n=␣2023244192
x=␣1

1a2␣3␣4␣5␣6␣7␣8␣9
y=␣1,␣z=␣2,␣n=␣4
x=␣2
```

Wejście: Specyfikacja formatu XI

...

```
#include <stdio.h>
int main(void)
{
    int x, y, z, n;
    x = scanf("%4da%5d%n", &y, &z, &n);
    printf("y= %d, z= %d, n= %d\n", y, z, n);
    printf("x= %d\n", x);
    return 0;
}
```

Wejście: Specyfikacja formatu XII

```
123a123
y= 123, z= 123, n= 7
x= 2
```

Podchwytliwe pytania I

P: Jak powinna wyglądać specyfikacja wprowadzania pozwalająca poprawnie zinterpretować dane w postaci:

`123ala456`

O: Oczywiście tak: %iala%i lub lepiej %dala%d
P: A jak powinna wyglądać specyfikacja pozwalająca przeczytać dane postaci

`123ala456`

oraz

`123ola456`

Odpowiedź będzie dłuższa...

Podchwytliwe pytania II

Popatrzmy na program

```
#include <stdio.h>
int main(void){
    int x, y, z, n;
    char tekst[100];
    x = scanf("%d%3[a-z]%d%n", &y, tekst, &z, &n);
    printf("y=␣%d␣", y);
    printf("tekst=␣%s␣", tekst);
    printf("z=␣%d␣", z);
    printf("n=␣%d\n", n);
    printf("x=␣%d\n", x);
    return 0;
}
```

Podchwytliwe pytania III

```
123ala20
y= 123 tekst= ala z= 20 n= 8
x= 3

12ooo3456
y= 12 tekst= ooo z= 3456 n= 9
x= 3

123ALA34
y= 123 tekst=  z= 1250361488 n= 0
x= 1
```

Podchwytliwe pytania IV

Jeżeli specyfikację zmienimy na: "%d%*3[a−z]%d%n" to dane tekstowe będą ignorowane, odpowiednie polecenie będzie wyglądać tak:

```
x = scanf("%d%*3[a−z]%d%n", &y, &z, &n);
```

Politechnika Wrocławska

Podchwytliwe pytania V

Co jeszcze może pojawić się wewnątrz nawiasów kwadratowych?

- ciąg znaków — dotyczy wymienionych znaków: [abc]
- znak ^ na pierwszym miejscu — wszystkie znaki za wyjątkiem wymienionych: [^ABC]
- *początek*−*koniec* — znaki z podanego zakresu: [a−z]

Warto też poczytać o wyrażeniach regularnych czasami nazywanych „regułowymi”...

Podchwytliwe pytania VI

P: W jaki sposób przeczytać dowolny napis?

- Jak wiadomo specyfikacja %s czyta znaki do pierwszego odstępu. Zatem nie można w ten sposób przeczytać napisau „Ala ma kota" (ze względu na odstępy.
- Można wspomóc się wyrażeniami regularnymi: specyfikacja %[aA]s czytać będzie dane tekstowe **zgodne z wzorcem**. Czyli tylko litery a lub A.
- Użycie specyfikacji %[^]s powoduje przeczytanie dowolnych znaków aż do znaku odstępu (w istocie znaczy to samo co poprzednio: wzorcem jest dowolny znak różny od spacji!).
- Co spowoduje zatem specyfikacja %[^\n]s?

Rodzina poleceń fprintf I

Wszystkie funkcje realizują formatowane wyprowadzanie informacji.

```
#include <stdarg.h>
#include <stdio.h>
int fprintf(FILE *stream, const char *format, \
            ...);
int printf(const char *format, ...);
int sprintf(char *s, const char *format, ...);
int snprintf(char *str, size_t size, const char *format, ...);
int vfprintf(FILE *stream, const char *format, \
             va_list arg);
int vprintf(const char *format, va_list arg);
int vsprintf(char *s, const char *format, \
             va_list arg);
```

- **fprintf** to podstawowa wersja funkcji

Rodzina poleceń fprintf II

- **printf** używa stdout jako strumienia wyjściowego.
- **sprintf** przekazuje sformatowane informacje do tablicy znakowej (odpowiedniej długości — musi zadbać programista)
- **snprintf** dodatkowy parametr mówi o długości tablicy znakowej, do której mają być zapisywane dane; dłuższe zostaną przycięte!
- warianty o nazwie rozpoczynającej się na v (vfprintf, fprintf, vsprintf) używają specyficznej metody przekazywania listy parametrów zmiennej długości — nie będziemy się nimi zajmować.

Wyjście: Specyfikacje formatu I

Podobnie jak w przypadku formatowanego wprowadzania danych, zmienna lub stała tekstowa zawiera informacje o sposobie wyprowadzania danych. Zasadą jest, że znak % rozpoczyna specyfikację konwersji, a pozostałe znaki są wyprowadzane w postaci takiej jak w formacie.
Specjalne wskaźniki modyfikujące sposób konwersji to:

- `-` wynik będzie justowany do lewej strony (standardowo do prawej)
- `+` Liczby będą zapisywane zawsze ze znakiem
- `odstęp` odstęp będzie zawsze dodawany przed liczbą (chyba, że liczba poprzedzona jest znakiem)
- `#` Wynik jest konwertowany do postaci „alternatywnej" (zależnej od typu konwersji)

Wyjście: Specyfikacje formatu II

- dla formatu **o** powoduje to zwiększenie precyzji, jeżeli jest to konieczne, aby na początku wyniku było zero;
- dla formatów **x** i **X** niezerowa liczba poprzedzona jest ciągiem 0x lub 0X;
- dla formatów **a**, **A**, **e**, **E**, **f**, **F**, **g** i **G** wynik zawsze zawiera kropkę nawet jeżeli nie ma za nią żadnych cyfr;
- dla formatów **g** i **G** końcowe zera nie są usuwane.

0 Liczby będą poprzedzone wiodącymi zerami (dla formatów **d**, **i**, **o**, **u**, **x**, **X**, **a**, **A**, **e**, **E**, **f**, **F**, **g** i **G**)

Szerokość pola i precyzja I

Minimalna szerokość pola oznacza ile najmniej znaków ma zająć dane pole. Jeżeli wartość po formatowaniu zajmuje mniej miejsca jest ona wyrównywana spacjami z lewej strony (chyba, że podano flagi, które modyfikują to zachowanie). Domyślna wartość tego pola to 0.
Precyzja dla formatów:

- **d**, **i**, **o**, **u**, **x** i **X** określa minimalną liczbę cyfr, które mają być wyświetlone i ma domyślną wartość 1;
- **a**, **A**, **e**, **E**, **f** i **F** — liczbę cyfr, które mają być wyświetlone po kropce i ma domyślną wartość 6;
- **g** i **G** określa liczbę cyfr znaczących i ma domyślną wartość 1;
- dla formatu **s** — maksymalną liczbę znaków, które mają być wypisane.

Politechnika Wrocławska

Szerokość pola i precyzja II

Szerokość pola może być albo dodatnią liczbą zaczynającą się od cyfry różnej od zera albo gwiazdką. Podobnie precyzja z tą różnicą, że jest jeszcze poprzedzona kropką. Gwiazdka oznacza, że brany jest kolejny z argumentów, który musi być typu **int**. Wartość ujemna przy określeniu szerokości jest traktowana tak jakby podano flagę – (minus).

Rozmiar argumentu I

1. Dla formatów **d** i **i** można użyć jednego ze modyfikator rozmiaru:
   - **hh** — oznacza, że format odnosi się do argumentu typu **signed char**,
   - **h** — oznacza, że format odnosi się do argumentu typu **short**,
   - **l** (*el*) — oznacza, że format odnosi się do argumentu typu **long**,
   - **ll** (**el el**) — oznacza, że format odnosi się do argumentu typu **long long**,
   - **j** — oznacza, że format odnosi się do argumentu typu intmax_t,
   - **z** — oznacza, że że format odnosi się do argumentu typu będącego odpowiednikiem typu size_t ze znakiem,
   - **t** — oznacza, że że format odnosi się do argumentu typu ptrdiff_t.
2. Dla formatów **o**, **u**, **x** i **X** można użyć takich samych modyfikatorów rozmiaru jak dla formatu **d** i oznaczają one, że format odnosi się do argumentu odpowiedniego typu bez znaku.

Rozmiar argumentu II

3. Dla formatu **n** można użyć takich samych modyfikatorów rozmiaru jak dla formatu **d** i oznaczają one, że format odnosi się do argumentu będącego wskaźnikiem na dany typ.
4. Dla formatów **a**, **A**, **e**, **E**, **f**, **F**, **g** i **G** można użyć modyfikatorów rozmiaru **L**, który oznacza, że format odnosi się do argumentu typu **long double**.
5. Dodatkowo, modyfikator **l** (*el*) dla formatu **c** oznacza, że odnosi się on do argumentu typu wint_t, a dla formatu **s**, że odnosi się on do argumenty typu wskaźnik na wchar_t.

Format I

Funkcje z rodziny printf obsługują następujące formaty:

d, i — argument typu **int** jest przedstawiany jako liczba całkowita ze znakiem w postaci [−]ddd.

o, u, x, X — argument typu **unsigned int** jest przedstawiany jako nieujemna liczba całkowita zapisana w systemie oktalnym (**o**), dziesiętnym (**u**) lub heksadecymalnym (**x** i **X**).

f, F — argument typu **double** jest przedstawiany w postaci [−]ddd.ddd.

e, E — argument typu **double** jest reprezentowany w postaci [−]d.ddde+dd, gdzie liczba przed kropką dziesiętną jest różna od zera, jeżeli liczba jest różna od zera, a + oznacza znak wykładnika. Format **E** używa wielkiej litery E zamiast małej.

Format II

g, G — argument typu **double** jest reprezentowany w formacie takim jak **f** lub **e** (odpowiednio **F** lub **E**) zależnie od liczby znaczących cyfr w liczbie oraz określonej precyzji.

a, A — argument typu **double** przedstawiany jest w formacie [−]0xh.hhhp+d czyli analogicznie jak dla **e** i **E**, tyle że liczba zapisana jest w systemie heksadecymalnym.

c — argument typu **int** jest konwertowany do **unsigned char** i wynikowy znak jest wypisywany. Jeżeli podano modyfikator rozmiaru **l** argument typu wint_t konwertowany jest do wielobajtowej sekwencji i wypisywany.

s — argument powinien być typu wskaźnik na **char** (lub wchar_t). Wszystkie znaki z podanej tablicy, aż do, i z wyłączeniem znaku null są wypisywane.

Politechnika Wrocławska

Format III

- p — argument powinien być typu wskaźnik na **void**. Jest to konwertowany na serię drukowalnych znaków w sposób zależny od implementacji.
- n — argument powinien być wskaźnikiem na liczbę całkowitą ze znakiem, do którego zapisana jest liczba zapisanych znaków.

Część II

Binarne wejście/wyjście

„Otwarcie" pliku I

```
#include <stdio.h>
FILE *fopen(const char *filename,
            const char *mode);
FILE *freopen(const char *filename,
              const char *mode, FILE *stream);
...
FILE *fp;
```

- Pierwszy parametr (filename) to wskaźnik do tablicy tekstowej (na przykład stałej) zawierającej nawę pliku.
- Parametr drugi to ciąg znaków (lub zmienna) zawierająca w sposób symboliczny określenie trybu dostępu:
  - **rb** otwórz plik binarny do czytania
  - **wb** utwórz plik do zapisu w trybie binarnym, jeżeli plik istnieje, jego zawartość zostanie skasowana

Politechnika Wrocławska

„Otwarcie" pliku II

- **ab** otwórz plik binarny w trybie do dopisywania (jeżeli plik nie istnieje — zostanie utworzony).
- **r+b** lub **rb+** zapis i odczyt dla plików binarnych (plik musi istnieć)
- **w+b** lub **wb+** otwórz w trybie binarnym plik istniejący (kasując jego zawartość) lub utwórz plik nowy w trybie do zapisu i odczytu
- **a+b** lub **ab+** otwórz plik w trybie binarnym na końcu w trybie do zapisu i odczytu

- Trzeci parametr (stream — tylko w przypadku funkcji reopen) zawiera wskaźnik do struktury danych opisujących strumień danych.

Zapis/Odczyt binarny I

1. **fread()**

```
#include <stdio.h>
size_t fread(void *ptr, size_t size,
             size_t nmemb, FILE *stream);
```

Funkcja realizująca dostęp bezpośredni do pliku, pozwala odczytać **nmemb** elementów o wielkości **size** każdy do tablicy wskazywanej przez **ptr** ze strumienia **stream**.
Funkcja zwraca liczbę przeczytanych elementów, która może być mniejsza od **nmemb** gdy wystąpi błąd lub funkcja dojdzie do końca pliku.

Zapis/Odczyt binarny II

2. **fwrite()**

```
#include <stdio.h>
size_t fwrite(const void *ptr, size_t size,
              size_t nmemb, FILE *stream);
```

Funkcja zapisuje binarnie elementy tablicy wskazanej przez **ptr** do strumienia stream. **size** określa wielkość jednego obiektu, a **nmemb** liczbę obiektów.

Zapis tablicy I

```
#include <stdio.h>
int main()
{
    FILE *pp;
    double t[10] = {
        0, 1, 2, 3, 4, 5, 6, 7, 8, 9
    };
    pp = fopen("Ala.txt", "wb");
    if ( pp == NULL )
    {
        printf("Blad!\n");
        return 1;
    }
    fwrite(t, sizeof( double ), 10, pp);
```

Zapis tablicy II

```
    fclose(pp);
    return 0;
}
```

Pliki binarne I

Prosty przykład

```
/*
 * Odczyt jednej liczby w trybie binarnym
 */
#include <stdio.h>
int main(void)
{
   FILE *fp;
   unsigned int e;
   fp = fopen("foo.txt", "rb");
   if ( fp == NULL )
   {
      printf("blad1\n");
      return 1;
```

Pliki binarne II

Prosty przykład

```
    }
    if ( fread(&e, sizeof( e ), 1, fp) == 0 )
    {
        printf("blad2\n");
        return 1;
    }
    printf("e = %u\n", e);
    return 0;
}
```

Pliki binarne III

Prosty przykład

1. Plik foo.txt nie istnieje

```
$ ls
b.c  Debug
$ Debug/binarne
blad1
```

Pliki binarne IV

Prosty przykład

2. Plik foo.txt istnieje, ale jest pusty

```
$ touch foo.txt
$ ls -l
razem 8
-rw-r--r-- 1 myszka myszka  319 2008-05-05 12:05 b.c
drwxr-xr-x 2 myszka myszka 4096 2008-05-05 12:05 Debug
-rw-r--r-- 1 myszka myszka    0 2008-05-05 12:12 foo.txt
$ Debug/binarne
blad2
```

Pliki binarne V

Prosty przykład

3. Plik istnieje, ma długość trzech bajtów:

```
$ ls -l
razem 12
-rw-r--r-- 1 myszka myszka  319 2008-05-05 12:05 b.c
drwxr-xr-x 2 myszka myszka 4096 2008-05-05 12:05 Debug
-rw-r--r-- 1 myszka myszka    3 2008-05-05 12:15 foo.txt
$ Debug/binarne
blad2
```

Pliki binarne VI

Prosty przykład

4. Plik istnieje, ma długość czterech bajtów:

```
$ cat foo.txt
abc
$ ls -l
razem 12
-rw-r--r-- 1 myszka myszka  319 2008-05-05 12:05 b.c
drwxr-xr-x 2 myszka myszka 4096 2008-05-05 12:05 Debug
-rw-r--r-- 1 myszka myszka    4 2008-05-05 12:18 foo.txt
$ Debug/binarne
e = 174285409
```

Zapis i odczyt I

```c
#include <stdio.h>
int main(void)
{
    FILE *fp;
    unsigned int i, k;
    fp = fopen("foo.txt", "wb");
    if ( fp == NULL )
    {
        printf("blad1\n");
        return 1;
    }
    for ( i = 0; i < 10; i++ )
    {
        fseek(fp, i * sizeof( i ), SEEK_SET);
```

Zapis i odczyt II

```
        fwrite(&i, sizeof( i ), 1, fp);
    }
    fclose(fp);
    fp = fopen("foo.txt", "rb");
    if ( fp == NULL )
    {
        printf("blad1\n");
        return 1;
    }
    for ( i = 0; i < 10; i++ )
    {
        fseek(fp, ( 9 - i ) * sizeof( i ), SEEK_SET);
        fread(&k, sizeof( i ), 1, fp);
        printf("k= %d\n", k);
```

Zapis i odczyt III

```
    }
    fclose(fp);
    return 0;
}
```

Zapis i odczyt (wariant drugi) I

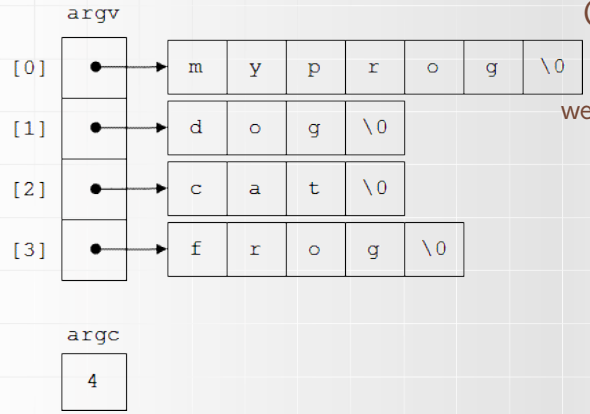
```c
#include <stdio.h>
int main(void)
{
    FILE *fp;
    unsigned int i, k;
    fp = fopen("foo.txt", "w+b");
    if ( fp == NULL )
    {
        printf("blad1\n");
        return 1;
    }
    for ( i = 0; i < 10; i++ )
    {
        fseek(fp, i * sizeof( i ), SEEK_SET);
```

Zapis i odczyt (wariant drugi) II

```
        fwrite(&i, sizeof( i ), 1, fp);
    }
    for ( i = 0; i < 10; i++ )
    {
        fseek(fp, ( 9 - i ) * sizeof( i ), SEEK_SET);
        fread(&k, sizeof( i ), 1, fp);
        printf("k= %d\n", k);
    }
    fclose(fp);
    return 0;
}
```

Politechnika Wrocławska

```
z123456@turing:~$ myprog dog cat frog
```



Operacje na łańcuchach znaków

wer. 6 z drobnymi modyfikacjami!

Wojciech Myszka

2018-03-27 08:05:39 +0200

HR EXCELLENCE IN RESEARCH

Politechnika Wrocławska

Łańcuch znaków

1. Z łańcuchów znaków korzystamy powszechnie.
2. Najprostszy przykład:

```
printf("Ala␣ma␣kota");
```

3. Łańcuchem znaków (ang: *string*) jest każdy napis zawarty w cudzysłowach.
4. Łańcuch znaków jest **tablicą** typu **char**, w której na ostatniej pozycji znajduje się znak null (znak o kodzie zero).

```
char *tekst = "Jakis␣tam␣tekst";
printf("%c\n", "przyklad"[0]);
   /* wypisze p - znaki w napisach sa
      numerowane od zera */
printf("%c\n", tekst[2]); /* wypisze k */
```

Łańcuch znaków cd I

1. Język C nie robi jakichś wielkich rozróżnień między znakami a cyframi:

```c
#include <stdio.h>
int main(void)
{
    char test[] = "test";
    int i;
    printf("%ld\n", sizeof(test));
    for (i = 0; i < sizeof(test); i++)
        printf("%d\n", test[i]);
    return 0;
}
```

2. Efekt działania tego programu

Łańcuch znaków cd II

5
116
101
115
116
0

Deklaracje

Poniższe deklaracje są (w zasadzie równoważne)

```
char *tekst  = "Jakis tam tekst";
 /* Umieszcza napis w obszarze danych
    programu i przypisuje adres */
char tekst[] = "Jakis tam tekst";
 /* Umieszcza napis w tablicy */
char tekst[] = {'J','a','k','i','s','\
    ' ','t','a','m',' ','t','e','k','\
    's','t','\0'};
 /* Tekst to taka tablica jak kazda inna */
```

Jednak w pierwszym przypadku tekst jest odsyłaczem do **stałej!** Elementów tablicy nie można zmieniać!

Operacje na łańcuchach

Plik nagłówkowy `string.h` zawiera definicje stałych i funkcji związanych z operacjami na łańcuchach tekstów.

```
void     *memccpy(void *, const void *, int,\
         size_t);
void     *memchr(const void *, int, size_t);
int      memcmp(const void *, const void *,\
         size_t);
void     *memcpy(void *, const void *,\
         size_t);
void     *memmove(void *, const void *,\
         size_t);
void     *memset(void *, int, size_t);
char     *strcat(char *, const char *);
char     *strchr(const char *, int);
int      strcmp(const char *, const char *);
int      strcoll(const char *, const char *);
char     *strcpy(char *, const char *);
size_t   strcspn(const char *, const char *);
char     *strdup(const char *);
```

```
char     *strerror(int);
size_t    strlen(const char *);
char     *strncat(char *, const char *,\
         size_t);
int      strncmp(const char *, const char *,\
         size_t);
char     *strncpy(char *, const char *,\
         size_t);
char     *strpbrk(const char *, const char *);
char     *strrchr(const char *, int);
size_t   strspn(const char *, const char *);
char     *strstr(const char *, const char *);
char     *strtok(char *, const char *);
char     *strtok_r(char *, const char *,\
         char **);
size_t   strxfrm(char *, const char *,\
         size_t);
```

Politechnika Wrocławska

Porównywanie ciągów znaków

1. Porównywanie pojedynczych znaków — na podstawie ich kodów ASCII
2. Dla ciągów znaków "aaa" > "aa" > "a" > ""; "ab" > "aa"; "a" > "A"
3. Funkcja *strcmp* służy do porównywania dwu ciągów znaków; ma dwa parametry i zwraca -1 gdy pierwszy ciąg znaków jest mniejszy od drugiego, 0 gdy są równe i $+1$, gdy pierwszy ciąg jest większy od drugiego.
4. Funkcja *strncmp* (o trzech parametrach, trzeci parametr wskazuje ile znaków ma być porównanych) może być użyta, gdy nie chcemy porównywać całych ciągów znaków.

Politechnika Wrocławska

Kopiowanie znaków

1. Do kopiowania służy funkcja *strcpy* o dwu parametrach; ciąg znakach zawarty w drugim parametrze kopiowany jest do pierwszego parametru. Do obowiązku programisty należy zapewnienie aby tablica będąca pierwszym parametrem miała wystarczającą ilość miejsca na pomieszczenie całego zapisu!
2. Drugi wariant funkcji *strncpy*; dodatkowy, trzeci parametr mówi ile znaków ma być skopiowanych — (ale zawsze trzeba pamiętać o miejscu na znak null znajdującym się na końcu napisu). Znaki z drugiego parametry kopiowane są do końca łańcucha, chyba że tekst jest dłuższy niż liczba znaków do skopiowania; w tym przypadku programista musi dodać znak NULL „ręcznie" na końcu skopiowanego tekstu.

Politechnika Wrocławska

Łączenie napisów I

1. Do łączenia napisów służy funkcja *strcat* (nazwa pochodzi od STRing conCATenate).
2. Podobnie jak w poprzednich przypadkach jest również drugi wariant *strncat*
3. W wersji standardowej Zawartość drugiego parametru funkcji dopisywana jest na końcu zawartości pierwszego.

```
char tekst[80] = "";
strcat(tekst, "ala");
strcat(tekst, "␣ma␣");
strcat(tekst, "kota");
puts(tekst);
```

4. Pierwszy parametr funkcji musi mieć dostateczną ilość miejsca na pomieszczenie połączonych ciągów znaków!
5. W drugim wariancie występuje trzeci parametr mówiący ile znaków ma być dołączonych.

Długość ciągu znaków

1. Funkcja *strlen* podaje długość ciągu znaków (bez znaku NULL)

Wyszukiwanie

1. Funkcja *strchr* (gdzie pierwszym parametrem jest ciąg znaków a drugim pojedynczy znak) zwraca numer znaku zgodnego z poszukiwanym.
2. Funkcja *strrchr* poszukuje od prawej do lewej.
3. Funkcja *strstr* (o dwu parametrach) wyszukuje pierwszego wystąpienia ciągu znaków będącego drugim parametrem w pierwszym.
4. Funkcja *strtok* może służyć do podziału pierwszego parametru na ciąg żetonów; zakładamy, że żetony (tokeny) oddzielone są zadanym znakiem (drugi parametr).

Konwersje

1. *atol*, *strtol* — zamienia łańcuch na liczbę całkowitą typu **long**
2. *atoi* — zamienia łańcuch na liczbę całkowitą typu **int**
3. *atoll*, *strtoll* — zamienia łańcuch na liczbę całkowitą typu **long long** (64 bity); dodatkowo istnieje przestarzała funkcja *atoq* będąca rozszerzeniem GNU,
4. *atof*, *strtod* — przekształca łańcuch na liczbę typu double

Funkcje ato* nie wykrywają błędów konwersji
Funkcje zdefiniowane są w pliku `stdlib.h`

Różne informacje

Plik nagłówkowy `ctype.h` zawiera definicje funkcji (i różnych stałych) związanych z rozpoznawaniem typów informacji

```
int    isalnum(int);
int    isalpha(int);
int    isascii(int);
int    isblank(int);
int    iscntrl(int);
int    isdigit(int);
int    isgraph(int);
int    islower(int);
int    isprint(int);
```

```
int    ispunct(int);
int    isspace(int);
int    isupper(int);
int    isxdigit(int);
int    toascii(int);
int    tolower(int);
int    toupper(int);
int    _toupper(int);
int    _tolower(int);
```

Politechnika Wrocławska

Konwersje I

```
int atoi( const char * string );
long atol( const char *s );
double atof( const char* string );
```

Parametrem funkcji jest ciąg znaków zawierający napis, który ma być skonwertowany

Konwersje II

Funkcje z rodziny strto*

```
long int strtol(const char *str, char **endptr, int base);
unsigned long int strtoul(const char *str, char **endptr, int base);
double strtod(const char *str, char **endptr);
```

Pierwszym parametrem funkcji jest napis podlegający konwersji, drugim jest znak ograniczający napis (może to być znak NULL). Trzecim parametrem w przypadku funkcji konwersji do postaci całkowitej jest podstawa zapisu liczb (zawarta pomiędzy 2 a 36). Specjalna wartość 0 pozwala automatycznie rozpoznawać liczby postaci 0nnnnn (gdzie n to cyfra od zera do 7) jako ósemkowe a liczby postaci 0xuuuu jako szesnastkowe (u to cyfry od zera do 9 oraz a — f lub A — F)

Politechnika Wrocławska

Unicode I

1. Jeżeli ograniczyć się do kodowania jednobajtowego (255 znaków) nie ma właściwie żadnych problemów, poza tym, że pogram musi być uruchamiany w odpowiednim środowisku. Dostępne kodowania znaków to ISO-8859-n (n od 1 do 9), cp-12xx, i tak dalej, i tak dalej...
2. Standardowe funkcje porównywania ciągów znaków nie będą działały!
3. Uniwersalny system kodowania znaków Unicode został dopuszczony do użytku w standardzie C99
4. Z Unicode też nie jest łatwo, mamy kilka rodzajów. Zestaw znaków Unicode obejmuje ponad 900 tys. symboli. Zapisać to można na 3 bajtach.
5. Ze względów technicznych przyjęto 4 bajty jako podstawowa wielkość znaku (bo łatwo jedną instrukcją) wybrać z pamięci.
6. W praktyce to za dużo, stąd warianty
   6.1 UTF-8 — od 1 do 6 bajtów (dla znaków poniżej 65536 do 3 bajtów) na znak; wszystkie standardowe znaki ASCII na jednym bajcie.

Unicode II

6.2 UTF-16 — 2 lub 4 bajty (UTF-32) na znak.

6.3 UCS-2 — 2 bajty na znak przez co znaki z numerami powyżej 65 535 nie są uwzględnione

7. Domyślne kodowanie dla C to kodowanie zleżne od systemu; linux: czterobajtowe!.

Unicode III

Co należy zrobić, by zacząć korzystać z kodowania UCS-2

- powinniśmy korzystać z typu **wchar_t** (ang. „wide character"), jednak jeśli chcemy udostępniać kod źródłowy programu do kompilacji na innych platformach, powinniśmy ustawić odpowiednie parametry dla kompilatorów, by rozmiar był identyczny niezależnie od platformy.
- korzystamy z odpowiedników funkcji operujących na typie char pracujących na **wchar_t** (z reguły składnia jest identyczna z tą różnicą, że w nazwach funkcji zastępujemy „str" na „wcs" np. *strcpy* — *wcscpy*; *strcmp* — *wcscmp*)
- jeśli przyzwyczajeni jesteśmy do korzystania z klasy **string**, powinniśmy zamiast niej korzystać z **wstring**, która posiada zbliżoną składnię, ale pracuje na typie **wchar_t**.

Polskie literki

UTF-8!

```c
#include <stdio.h>
#include <string.h>
int main(void)
{
    char napis1[30] = "Ala ma kota";
    char napis2[30] = "Ala ma kotę";
    printf( "napis1: %d\n", (int) strlen(napis1) );
    printf( "napis2: %d\n", (int) strlen(napis2) );
    return 0;
}
```

Wyniki:

```
napis1: 11
napis2: 12
```

Polskie literki

```c
#include <stdio.h>
#include <string.h>
#include <wchar.h>
int main(void)
{
    char napis1[30] = "Ala ma kota";
    char napis2[30] = "Ala ma kotę";
    wchar_t napis3[12] = L"Ala ma kotę";
    printf( "napis1: %d\n", (int) strlen(napis1) );
    printf( "napis2: %d\n", (int) strlen(napis2) );
    printf( "napis3: %d, sizeof napis3 %d\n",
            (int) wcslen(napis3),
            (int) sizeof( napis3 ) );
    return 0;
}
```

Wyniki

```
napis1: 11
napis2: 12
napis3: 11, sizeof napis3 48
```

Polskie literki

```c
#include <stdio.h>
#include <string.h>
#include <wchar.h>
int main(void)
{
    int i;
    wchar_t m[2] = L"A";
    union ccc{
        char n[8];
        wchar_t N[2];
    } c;
    c.N[0] = m[0];
    c.N[1] = m[1];
    for (i=0; i<8; i++)
    printf("%d - %x\n", i, c.n[i]);
    return 0;
}
```

```
0 - 41
1 - 0
2 - 0
3 - 0
4 - 0
5 - 0
6 - 0
7 - 0
```

Polskie literki

```c
#include <stdio.h>
#include <string.h>
#include <wchar.h>
int main(void)
{
    int i;
    wchar_t m[2] = L"Ą";
    union ccc{
        char n[8];
        wchar_t N[2];
    } c;
    c.N[0] = m[0];
    c.N[1] = m[1];
    for (i=0; i<8; i++)
    printf("%d - %x\n", i, c.n[i]);
    return 0;
}
```

```
0 - 4
1 - 1
2 - 0
3 - 0
4 - 0
5 - 0
6 - 0
7 - 0
```

Locale

1. *Locale* to zestaw definicji określających specyficzny dla różnych języków (i krajów) sposób prezentacji różnych informacji.
2. Z jakichś dziwnych powodów problem nie ma zbyt bogatej „literatury".
3. Jeżeli ktoś chce się zapoznać z pewną realizacją uniksową — mogę polecić bardzo stary tekst System Operacyjny HP-UX a sprawa polska.

Locale

Unix

Zachowanie systemu po wyborze określonego języka zmienia się. Za pomocą kilku zmiennych środowiska można regulować zakres w jakim podporządkowujemy się specyficznym regułom. I tak:

LANG	określa używany język,
LC_CTYPE	definiuje charakter (litery, cyfry, znaki przestankowe, znaki drukowalne) poszczególnych znaków,
LC_COLLATE	określa sposób sortowania,
LC_MONETARY	opisuje zaznaczania jednostek monetarnych (symbol waluty, gdzie jest on umieszczany...),
LC_NUMERIC	sposób zapisu liczb, znak oddzielający grypy cyfr, znak oddzielający część całkowitą od ułamkowej,
LC_TIME	postać podawania daty i czasu,
LC_MESSAGES	język używany w wyświetlanych komunikatach,
LC_ALL	wszystkie LC_*.

Locale

Programowanie

1. Sprawa nie jest prosta!
2. Można wszystko zrobić na wiele sposobów.
3. Bardzo często komunikaty wyprowadzanie przez (różne) programy powtarzają się.
4. Powstaje myśl stworzenia „bazy danych” zawierających różne komunikaty oraz ich tłumaczenia na różne języki.
5. Są też gotowe biblioteki zapewniające obsługę takiej bazy danych (oraz jej tworzenie).

Locale

Przykład

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <wchar.h>
#include <locale.h>
int main(void)
{
/*     printf("%s\n", getenv("LANG")); */
    setlocale(LC_ALL, getenv("LANG"));
    printf("%d␣\n", wcscoll(L"żółć", L"żółw"));
    return 0;
}
```

Wyniki

```
$ LANG=C ./stringi
144
$ LANG=pl_PL.utf8 ./stringi
-25
```