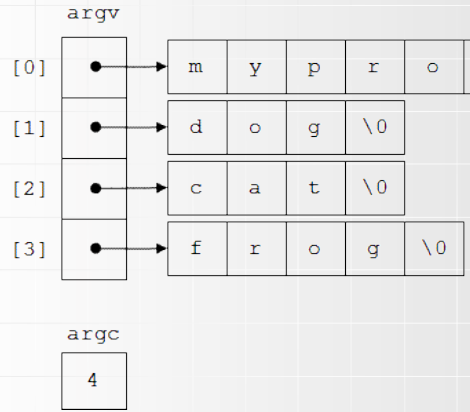
Politechnika Wrocławska

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



Operacje na łańcuchach znaków

wer. 7

Wojciech Myszka

Katedra Mechaniki, Inżynierii Materiałowej i Biomed

2020-05-20 08:40:08 +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!

Politechnika Wrocławska

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);
```

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.

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`

Politechnika Wrocławska

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);
```

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)

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
```

Politechnika Wrocławska

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
```

Politechnika Wrocławska

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
1
$ LANG=pl_PL.utf8 ./stringi
-278
```