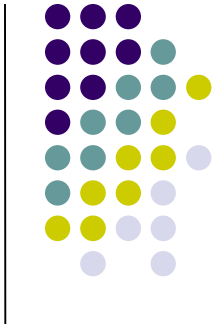


Знаковни низови - стрингови

NIT

Знаковни низови у Цеу

- Знаковни низови нису посебан тип података.
- **Знаковни низ је низ char типа који се завршава са ‘\0’.**
- Неколико синтаксних специфичности и подршка у стандардним библиотекама чине знаковни низ посебним елементом Це језика
- Све остало је на програмеру.
- Зато морамо бити пажљиви, јер рад са знаковним низовима је извор многих проблема.

Пример:

```
char buffer[21];
```

Заузима меморију за знаковни низ у који стаје 20 знакова.

NIT

Синтаксне посебности

- Постојање стринг литерала/константи.

  `"ovo je string literal"`

  - Спајање (или надовезивање, никако ~~*конкатенација*~~) стринг литерала

    `"string" " literal" " sa" " odvojenim" " recima"`

    `"string literal sa odvojenim recima"`

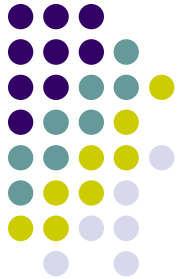
    Корисно, рецимо, када се прелази у нови ред.

- Иницијализација

```
char string[] = {1, 2, 3, 4, 5};
char string[] = {'a', 'b', 'c', 'd', 'e', '\0'};
char string[] = "abcde";
char* string = {1, 2, 3, 4, 5};
char* string = {'a', 'b', 'c', 'd', 'e', '\0'};
char* string = "abcde";
```

NIT

Где се смешта стринг литерал?

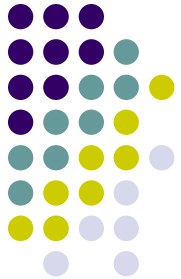


```
char* p = "Zdravo!";
p[3] = 't';
printf("Zdravo!");
scanf("%s", str);
if (strcmp("Zdravo!", str) == 0)
{
   ...
}
```

- Оба ова стринга Zdravo! могу завршити у истој меморији.
- Зато је опасно мењати стринг литерале, тј. то доводи до недефинисаног стања.

NIT

Тип стринг литерала



- Ког типа је стринг литерал?

NIT

Тип стринг литерала

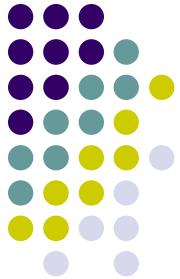
- Ког типа је стринг литерал?

  `char*`

- Зашто није типа `const char*`, кад већ не би требало да се мења?
- Зато што је `const` уведено тек касније...

- У Це++-у тип заиста јесте `const char*`

Знаковни литерал наспрам стринг литерала



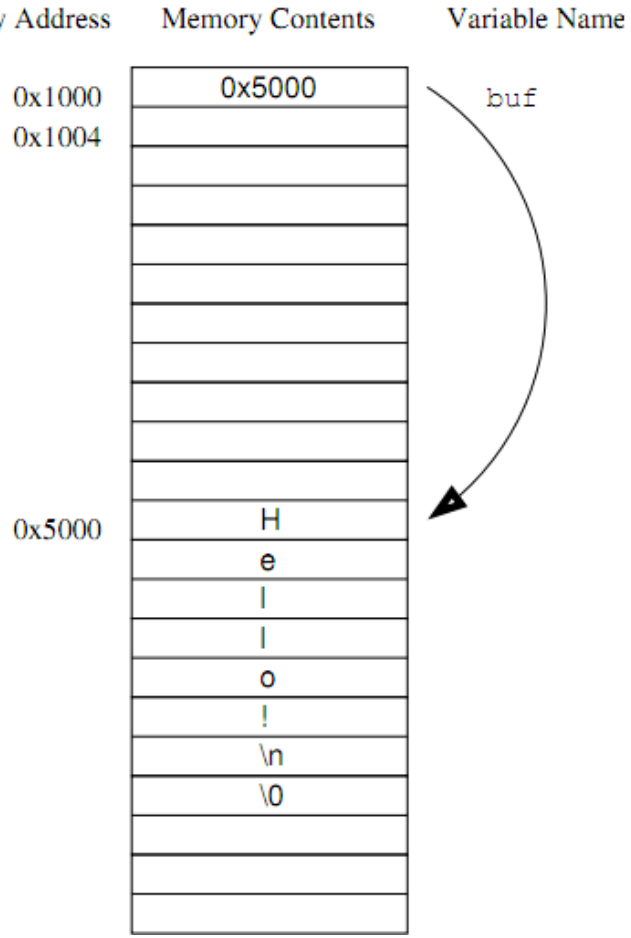
Знаковни литерал је под једноструким наводницима (’, не ”).

```
char buf[10];
buf[0] = ’A’; /* correct */
buf[0] = "A"; /* incorrect */
buf[1] = ’\0’;   /* NULL terminator */
```

Пример

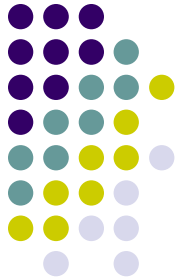
```
const char* buf = "Hello!\n";
```

- Променљива **buf** је показивач на меморију где се стринг налази.
- Приметити NULL (’\0’) знак на крају - аутоматски је додат.



NIT

Библиотечке посебности



- \0 на крају је битно само зато што то библиотечке функције очекују.

- Посебни симбол у формат стрингу за printf и scanf (и сродне функције).

```
char str[] = “Nesto”;
int i;
printf(“%s”, str); // sta ako nema \0 na kraju?
scanf(“%d%s”, &i, str); // sta ako vise od 5 znakova?
```

- Библиотечке функције за рад са знаковним низовима
  - Пре свега string.h
  - stdlib.h
  - stdio.h

Копирање стрингова

```
char* buf1 = "Hello";
char* buf2 = "olleH";
buf2 = buf1;
buf2[2] = 'M';
printf("%s %s", buf1, buf2);
```

Runtime error!

```
char* buf1 = "Hello";
char buf2[100];
buf2 = buf1;

Compile error!
```

```
#include <string.h>
char* buf1 = "Hello";
char buf2[100];
strcpy(buf2, buf1);
buf2[2] = 'M';
printf("%s %s", buf1, buf2);
Output: Hello HeMlo
```

Копирање стрингова

```
const char* buf1 = "Hello";
const char* buf2 = "olleH";
buf2 = buf1;
buf2[2] = 'M';
printf("%s %s", buf1, buf2);
```

Compiler error!

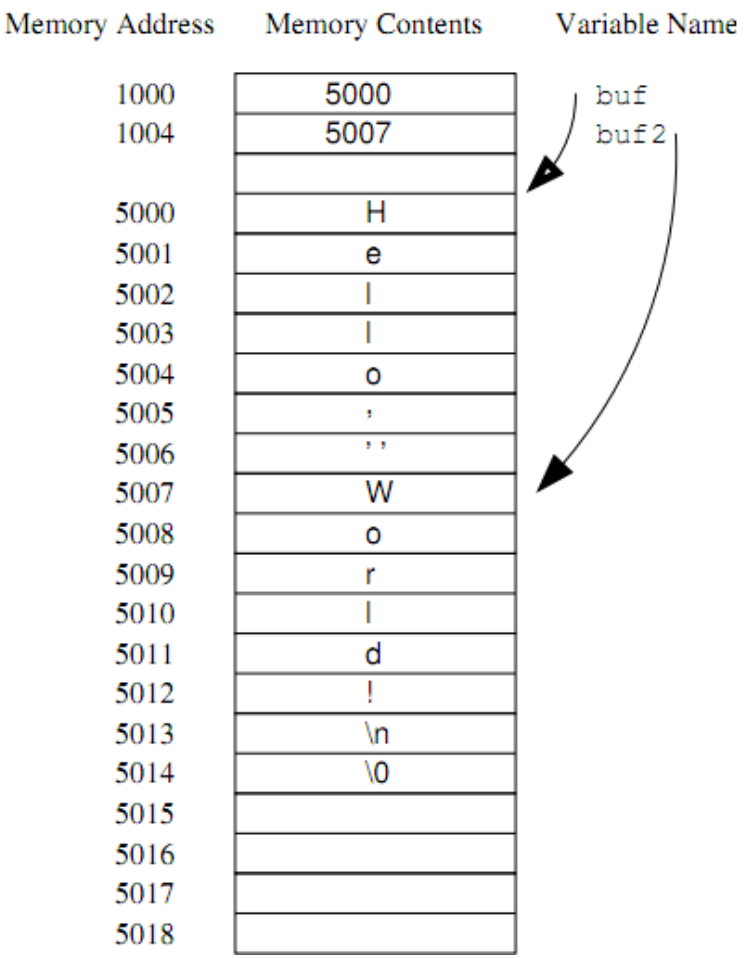
```
const char* buf1 = "Hello";
char buf2[100];
buf2 = buf1;

Compile error!
```

```
#include <string.h>
const char* buf1 = "Hello";
char buf2[100];
strcpy(buf2, buf1);
buf2[2] = 'M';
printf("%s %s", buf1, buf2);
Output: Hello HeMlo
```

Пример

```
char buf[] = "Hello, World!\n";
char* buf2 = buf + 7;
printf("buf: %s\n", buf);
printf("buf2: %s\n", buf2);
buf2[0] = 'M';
printf("buf: %s\n", buf);
```

Шта је излаз?



Премашивање бафера

Стринг не расте сам по потреби. Бафер (парче меморије) који му је додељен се не мења.

Пример:

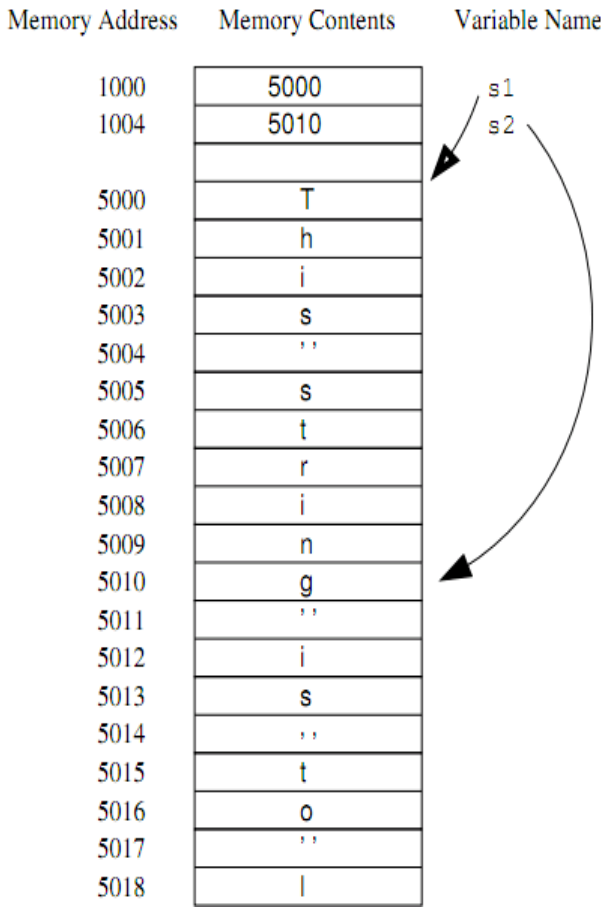
```
char s1[] = "1. string";
char s2[] = "2. string";
strcpy(s1, "This string is too long!\n");
```

Копирамо стринг величине 25 у меморију која је предвиђена за 9 знакова!

Врло је могуће да смо преписали s2!

Шта више, пошто смо премашили и s2 почели смо са писањем по ко зна чему.

Компајлер неће ово приметити, а и често неће бити детектовано ни током извршавања (осим што програм неће радити)!



Стринг литерали

Пример 1

```
char* str;
str = "hello";
printf("%s\n", str);
```

Пример 2

```
char str[100];
strcpy(str, "hello");
printf("%s\n", str);
```

Исти испис на екран, али понашање врло различито.

NIT

Стринг литерали

Пример 1

```
char* str;
str = "hello";
printf("%s\n", str);
strcpy(str, "hello");
```

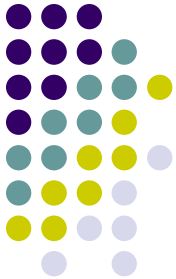
Пример 2

```
char str[100];
strcpy(str, "hello");
printf("%s\n", str);
str = "hello";
```

Пример 1 узрокује упис на недозвољено место.
Пример 2 се неће ни превести.

NIT

Стринг литерали



Пример 1

```
const char* str
  = "hello";
printf("%s\n", str);
strcpy(str, "hello");
```

Пример 2

```
char str[100];
strcpy(str, "hello");
printf("%s\n", str);
str = "hello";
```

Са const обезбеђујемо да и пример 1 пријави грешку током превођења.

Још један поучан пример

```
int main()
{
    char* str;
    str = (char*)malloc(100);
    str = "hello";
    free(str);
    return 0;
}
```

```
int main()
{
    char* str;
    str = (char*)malloc(100);
    strcpy(str, "hello");
    free(str);
    return 0;
}
```

Леви пример се преводи исправно али приликом извршавања пријављује грешку. Зашто?

Стрингови као параметри функције

Као и редовни низови, стрингови се преносе само „преко референце”.

```
void Print1(char* str)
{
    printf("%s", str);
}

void Print2(char* ary, int n)
{
    int i;
    for (i = 0; i < n; i++)
        printf("%c", ary[i]);
}
```

<string.h> неке важније функције

```
char* strcpy(char* s1, const char* s2);
char* strncpy(char* s1, const char* s2, size_t n);

char* strcat(char* s1, const char* s2);
char* strncat(char* s1, const char* s2, size_t n);

int strcmp(const char* s1, const char* s2);
int strncmp(const char* s1, const char* s2, size_t n);

char* strtok(char* str, const char* delim);
```

Погледати у тексту стандарда детаљни опис ових функција.

<string.h> још функција

```
void* memcpy(void* s1, const void* s2, size_t n);
void* memmove(void* s1, const void* s2, size_t n);

int memcmp(const void* s1, const void* s2, size_t n);

void* memset(void* str, int c, size_t n);
```

Функције за конверзију

Из знаковног низа у бројеве. <stdlib.h>

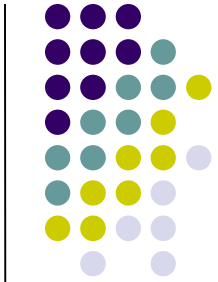
```
int atoi(const char* nptr);
long atol(const char* nptr);
long long atoll(const char* nptr);
double atof(const char* nptr);
```

Обрнуто? <stdio.h>

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

sprintf(s, "%d", 5);
```

NIT

<ctype.h>



```
int isalnum(int ch);

int isalpha(int ch);

int islower(int ch);

int isupper(int ch);
```