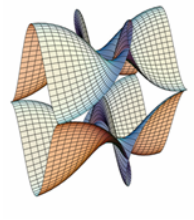

Prirodoslovno-matematički fakultet
Matematički odsjek
Sveučilište u Zagrebu

RAČUNARSKI PRAKTIKUM I

Vježbe 03 - Implementacija strukture string

v2019/2020.

Sastavio: Zvonimir Bujanović

Stringovi u C-u

String u C-u je polje **char**-ova koje završava znakom `'\0'`.

```
char s1[50], s2[50];

strcpy( s1, "bla bla" );
strcpy( s2, s1 );
strcat( s1, "nesto" );
int dulj = strlen( s1 );
```

Operacije sa stringovima u C-u su globalne funkcije.

Stringovi u C++-u

String u C++-u je struktura koja se “sama brine za sebe”.

```
#include <string>

int main( void )
{
    std::string A( "bla bla" ), B;

    B = A;
    A.append( B );
    int dulj = A.size();
    ...
}
```

U ovim vježbama ćemo:

1. Upoznati funkcije iz strukture `string`.
2. Sami implementirati strukturu koja ima (skoro) istu funkcionalnost.

String: Konstruktori

- Defaultni konstruktor (bez parametara).

```
string S;
// sad je S prazan string
```

- Konstruktor koji prima C-ovski string.

```
string S( "nesto" );
// sad je S = "nesto"
```

- Konstruktor koji stvara string od nekoliko kopija istog znaka.

```
string S( 5, 'X' );
// sad je S = "XXXXX"
```

String: Konstruktori

Alternativni način pozivanja konstruktora.
(Ovo radi i za bilo koje druge strukture.)

```
string S;
S = string( "nesto" );
// sad je S = "nesto"
```

```
string S;
S = string( 5, 'X' );
// sad je S = "XXXXX"
```

Sa desne strane smo stvorili privremenu, anonimnu varijablu koja se zatim prekopirala u varijablu `S`.

String: Ispisivanje i učitavanje

Stringove možemo učitavati sa `cin` i ispisivati sa `cout`.

```
string S;

cin >> S;
cout << S;
```

Stringove (kao ni druge strukture koje sami napišemo!) ne možemo učitavati sa `scanf` ni ispisivati sa `printf`!

Vidjet ćemo kasnije kako "naučiti" `cin` i `cout` ispisivati naše strukture (npr. varijable tipa `tocka`).

String: Veličina i konverzija u C-ovski string

`size()` - vraća duljinu stringa

```
string S( "nesto" );
int a = S.size(); // sad je a=5
```

`c_str()` - vraća pokazivač na C-ovski string.

```
string Scpp( "nesto" );
char Sc[20];

// Kopiramo string iz Scpp u Sc.
// Za Sc treba postojati dovoljno memorije.
strcpy( Sc, Scpp.c_str() );

printf( "%s", Scpp.c_str() ); // OK
printf( "%s", Scpp ); // NIJE OK!

// Kopiranje u obratnom smjeru NE RADI -> konstruktor.
strcpy( Scpp.c_str(), Sc ); // NIJE OK!
```

String: Kopiranje

Kopiranje jednog stringa u drugi.

```
string A( "nesto" ), B;

B = A; // sada je B = "nesto"
```

Kopiranje stringova u C-u nismo mogli raditi ovako!

- U C-u su stringovi polja (tj. pokazivači), pa bi se `B=A` interpretiralo kao kopiranje pokazivača (memorijskih adresa).

S druge strane, kopiranje struktura u C/C++ oblika `B=A` funkcionira:

- sva memorija ("niz nula i jedinica") koju zauzima varijabla `A` se prekopira u memoriju koju zauzima varijabla `B`.

String: Dodavanje na kraj

`append()` - dodavanje jednog stringa na kraj drugog

```
string A( "nesto" ), B( "drugo" );

A.append(B); // sad je A="nestodrugo"
B.append(3,'Z'); // sad je B="drugoZZZ"
```

Alternativno, možemo koristiti i operatore.

```
string A( "nesto" ), B( "drugo" );

A += B;
B = B + "ZZZ";

string C = string("XYZ") + "ABC" // OK
string C = "XYZ" + "ABC" // NIJE OK: bar jedan mora biti C++
```

String: Operatori uspoređivanja

Uspoređivanje jednakosti.

```
string S( "bla" ), T( "nesto" );

if( S == T )
    cout << "isti";
else
    cout << "nisu isti"; // ovo se ispise
```

Uspoređivanje po abecedi.

```
string S( "bla" ), T( "nesto" );

if( S < T )
    cout << "T je veci"; // ovo se ispise
else
    cout << "S je veci";
```

String: Podstringovi

`substr(pocetak, duljina)` - vraća podstring od `duljina` znakova koji počinje na indexu `pocetak`.

```
string A( "Nesto" ), B;

B = A.substr( 3, 2 ); // B="to"
```

`erase(pocetak, duljina)` - briše podstring od `duljina` znakova koji počinje na indexu `pocetak`.

```
string S( "nestodrugo" );

S.erase( 2, 6 ); // sad je S="nego"
```

String: Podstringovi

`find(stoTrazim, gdjePocinjem)` - vraća index na kojem se prvi put nakon indexa `gdjePocinjem` javlja podstring `stoTrazim`.

```
string A( "kokodako" );

int gdje = A.find( "ko", 1 ); // gdje=2
// trazi "ko" pocevsi od indexa 1 (ne 0)
// "ko" se prvi put (iza indexa 1) javi na indexu 2
```

Ako `find` ne uspije naći podstring, vraća `string::npos`.

```
string S( "kokoda" ), T( "kokos" );

int gdje = S.find( T, 0 );
if( gdje == string::npos )
    cout << "nema ga";
```

String: Pristup pojedinim znakovima

Znakovima u stringu možemo pristupiti pomoću uglatih zagrada, kao u C-u.

```
string S( "nesto" );

char znak = S[3]; // znak='t'
S[2] = 'r'; // sad je S="nerto";
```

Postoje i mnoge druge funkcije, vidi:
http://www.cplusplus.com/reference/string/string/

Zadatak 1

Napišite program koji, redom:

1. Učitava dva stringa `A` i `B`.
2. Ako se u stringu `A` pojavljuje `B` kao podstring, onda iz `A` treba obrisati taj podstring. Ovaj postupak treba ponavljati sve dok god se `B` pojavljuje kao podstring.
   Označimo sa `n` ukupan broj tih pojavljivanja.
3. `n` kopija stringa `B` dodaje na kraj stringa `A`.

Na primjer, ako je `A="kokodako"` i `"B=ko"`, onda u koraku 2 pronalazimo `n=3` kopije, a nakon brisanja vrijedi `A="da"`. Nakon koraka 3 je `A="dakokoko"`.

Zadatak 2

Implementirajte svoj tip podatka `MyString` koji ima dio funkcionalnosti stringa iz C++-a:

- sve konstruktore
- funkcije `size()`, `append()`, `substr()`, `find()`
- umjesto `A=B` neka postoji funkcija `A.copy(B)`
- umjesto `A==B` neka postoji `A.isEqual(B)`
- umjesto `A<B` neka postoji `A.isLessThan(B)`

Pretpostavite da je duljina stringa max. 100 znakova. U slučaju da bi neki poziv funkcije rezultirao duljim stringom, prijavite grešku.

Strukture sa dinamički alociranim poljem

Što ispisuje sljedeći fragment koda?

```
struct MyString
{
    char polje[100];
    ...
};

void test( MyString S )
{
    S.polje[1] = 'A';
    cout << S.polje;
}
```

```
int main( void )
{
    MyString S("XYZ");

    test( S );
    cout << S.polje;



    return 0;
}
```

Strukture sa dinamički alociranim poljem

Što ispisuje sljedeći fragment koda?

```
struct MyString
{
    char *polje;
    ...
};

void test( MyString S )
{
    S.polje[1] = 'A';
    cout << S.polje;
}
```

```
int main( void )
{
    MyString S("XYZ");

    test( S );
    cout << S.polje;



    return 0;
}
```

Ako korisnik strukture ne vidi njezinu implementaciju, očekuje li ovakav ispis ili onaj s prethodne stranice?

Copy-konstruktor

- Trebamo “naučiti” strukturu sa dinamički alociranim poljem da se na ispravan način zna prenijeti kao parametar u funkciju.
- Za prenošenje parametara u funkciju zadužena je jedna specijalna vrsta konstruktora: tzv. copy-konstruktor.
- On služi ispravnom kopiranju dvaju istovrsnih objekata.
- Treba nam samo kod struktura sa dinamički alociranim elementima.

Copy-konstruktor

Svaka struktura odmah po deklaraciji ima implicitni copy-konstruktor: on svu memoriju koju zauzima objekt `P` prekopira u objekt koji se konstruira. Ako ga sami ne napišemo, događa se sljedeće:

```
struct proba
{
    int a, *b, c[100];
    double x;

    proba( const proba &P )
    {
        // kopiraj memoriju koju zauzima P u memoriju
        // koju zauzima objekt koji se konstruira
    }
};
```

Napomena: `proba &P` je tzv. referenca – više o referencama kasnije.

Copy-konstruktor

Dakle, implicitni copy-konstruktor radi sljedeće:

```
struct proba
{
    int a, *b, c[100];
    double x;

    proba( const proba &P )
    {
        a = P.a;
        b = P.b;
        // polje P.c se zatim byte po byte kopira u polje c
        // jer se memorija koju zauzima P.c nalazi
        // "unutar" varijable P
        x = P.x;
    }
};
```

Copy-konstruktor

Kod strukture `MyString`, implicitni copy-konstruktor bi napravio ovo:

```
struct MyString
{
    char polje[100];

    MyString( const MyString &A )
    {
        cout << "copy-konstruktor!";
        strcpy( polje, A.polje );
    }
};


...
MyString A( "nesto" );
MyString B( A ); // poziv copy-kon!!!
test( A ); // poziv copy-kon!!!
```

(Zapravo, svih 100 znakova iz `A.polje` bi se prekopiralo u `polje`.)

Copy-konstruktor

Uz dinamičku alokaciju za polje, unutar copy-konstruktora moramo alocirati novo polje!

```
struct MyString
{
    char *polje;

    MyString( const MyString &A )
    {
        cout << "copy-konstruktor!";
        polje = (char *) malloc( A.size() + 1 );
        strcpy( polje, A.polje );
    }
};


...
MyString A( "nesto" );
MyString B( A ); // poziv copy-kon!!!
test( A ); // poziv copy-kon!!!
```

C++11: ključna riječ `nullptr`

Do sada smo NULL-pokazivač inicijalizirali ovako:

```
#include <stdlib.h>

// NULL je makro-konstanta definirana u stdlib.h
int *p = NULL;
```

C++11 - Uvodi se nova ključna riječ `nullptr`.

```
int *p = nullptr;
```

Zadatak 3

Promijenite program iz Zadatka 2 tako da podržava stringove varijabilne duljine.

- Svaki string treba zauzimati točno onoliko memorije koliko znakova ima (i jedan dodatni za `'\0'`).
- `append` treba napraviti realokaciju memorije.
- Radi jednostavnosti koristite `malloc`, `free` i `realloc`.

Napomena: u C++-u nema ekvivalenta funkciji `realloc` pa bi trebalo raditi nekoliko new+delete-ova da bi se implementirala funkcija `append`.