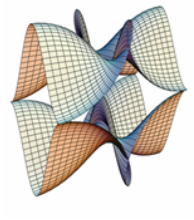

Prirodoslovno-matematički fakultet
Matematički odsjek
Sveučilište u Zagrebu

RAČUNARSKI PRAKTIKUM I

Vježbe 02 - Funkcije kao elementi strukture

v2018/2019.

Sastavio: Zvonimir Bujanović

Strukture i funkcije u C-u

Funkcije koje rade s pojedinom strukturom su globalne:

```
int top( stack S ) { return S.element[S.size-1]; }
void push( stack *S, int x ) { ... }

int main( void )
{
    stack S, T;

    makeNull( &S ); makeNull( &T );
    push( &S, 7 ); push( &T, 9 );
    int a = top( S ); pop( &T );

    return 0;
}
```

Kojem tipu pripada koja funkcija? Npr. `makeNull` $\rightsquigarrow$ `stack` ili `queue`?
U C-u funkcije ne mogu imati isto ime, a različite tipove parametara.

Strukture i funkcije u C++-u

Svaka varijabla (objekt) nekog složenog tipa (strukture) se treba sama brinuti za svoju funkcionalnost, tj. svoj sadržaj.

To ne treba biti uloga globalnih funkcija.

```
int main( void )
{
    stack S, T;

    S.makeNull(); T.makeNull();
    S.push( 7 ); T.push( 9 );
    int a = S.top(); T.pop();

    return 0;
}
```

Funkcija kao element strukture

Da to postignemo, trebaju nam funkcije kao elementi u strukturi!

Deklaracija strukture čiji članovi su i funkcije:

```
struct stack
{
    int size, element[100];

    void makeNull( void );
    void push( int x );
    void pop( void );
    int top( void );
};
```

Funkcija kao element strukture - Implementacija

Implementaciju funkcije koja je član strukture možemo napisati na dva načina.

Prvi način: unutar same deklaracije strukture.

- Ovo je pogodno za male strukture koje pišemo u istoj datoteci u kojoj je `main`.
- Ovako ne možemo razdvojiti sučelje od implementacije!

```
struct stack
{
    int size, element[100];

    void makeNull( void ) { size = 0; }

    int top( void ) { return element[size-1]; }
    ...
};
```

Funkcija kao element strukture - Implementacija

Drugi način: izvan deklaracije strukture.

- Ovo je pogodno za veće strukture i za razdvajanje sučelja od implementacije.

```
struct stack
{
    int size, element[100];

    void makeNull( void );
    int top( void );
    ...
};

void stack::makeNull( void ) { size = 0; }

int stack::top( void ) { return element[size-1]; }
```

Funkcija kao element strukture

```
void stack::makeNull( void )
{
    size = 0;
}

int main( void )
{
    stack S, T;
    S.makeNull(); T.makeNull();

    return 0;
}
```

Postoje `S.size`, `T.size`, `S.element`, `T.element`.

`makeNull` zna koja ga varijabla (objekt) poziva i ovisno o tome mijenja `S.size` ili `T.size`.

Funkcije članice strukture vide podatke koji pripadaju istoj strukturi.

Zadatak 1

Napišite implementaciju sljedeće strukture, te odgovarajuću funkciju main.

```
struct tocka
{
    int x, y;

    void ispisiTocku();
    void unesiTocku();
    tocka simetricnaTocka();
    float udaljenost( tocka Q );
};
```

Pristup članovima strukture

Funkcijama koje su članovi strukture pristupamo jednako kao i podacima koji su članovi.

```
tocka P = {10, 20}, Q, *T = &P;

Q.x = 10; Q.y = 30;
T->x = 15;

P.ispisiTocku();
T->ispisiTocku();
Q.ispisiTocku();

cout << P.udaljenost( Q );
tocka R = T->simetricnaTocka();
```

Razdvajanje sučelja i implementacije

Sučelje i implementaciju možemo razdvojiti u dvije datoteke ako koristimo “drugi način”.

```
// tocka.h
struct tocka
{
    int x, y;
    void ispisiTocku();
};
```

```
// tocka.cpp
#include "tocka.h"

void tocka::ispisiTocku( void )
{
    cout << "(" << x << ", " << y << ")";
}
```

Više varijabli istog imena

Unutar funkcije članice strukture, možemo pristupiti:

- Lokalnim varijablama deklariranim unutar te funkcije.
- Varijablama koje su članovi strukture.
- Globalnim varijablama.

```
int z = 3;
struct test
{
    int y; // postavimo negdje vani y=5;

    void ispis()
    {
        int x = 7;
        cout << x; // ispise 7
        cout << y; // ispise 5
        cout << z; // ispise 3
    }
};
```

Više varijabli istog imena

Ako te varijable imaju isto ime, svejedno im možemo pristupiti.

- Lokalnoj varijabli `x` pristupamo bez promjene.
- Varijabli `x` koja je član strukture pristupamo sa `imestrukture::x`.
- Globalnoj varijabli `x` pristupamo sa `::x`.

```
int x = 3;
struct test
{
    int x; // postavimo negdje vani x=5;

    void ispis()
    {
        int x = 7;
        cout << x; // ispise 7
        cout << test::x; // ispise 5
        cout << ::x; // ispise 3
    }
};
```

Konstruktori

Često elemente svakog objekta odmah na početku treba postaviti na neku vrijednost:

```
void stack::makeNull( void )
{
    size = 0;
}
```

Standardizirani način kako se to radi: konstruktor.
⇝ specijalna funkcija koja ima isto ime kao i sama struktura.

```
struct stack
{
    int size, element[100];

    stack( void )
    {
        size = 0;
    }
};
```

Konstruktori

- Konstruktoru se nikad ne piše povratni tip!
  (U implementaciji isto ne vraćamo nikakvu vrijednost sa **`return`**.)
- Implementacija izvan strukture (“drugi način”):

```
struct stack
{
    int size, element[100];

    stack();
    ...
};

stack::stack()
{
    size = 0;
}
```

Konstruktori

Konstruktor se sam, automatski pozove čim deklariramo varijablu!

```
int main( void )
{
    stack S, T; // pozovu se konstruktori za S i T

    cout << S.size; // ispise 0
    cout << T.size; // ispise 0

    return 0;
}
```

Još jedan način inicijalizacije (C++11)

C++11 - Dodatni način za inicijalizaciju strukture.
Napomena: konstruktor se pozove nakon inicijalizacije.

```
struct stack
{
    int size = 0, element[100] = {5, 6, 7};

    stack() { cout << "Inicijalizacija..."; }
    ...
};

int main( void )
{
    stack S; // ispise "Inicijalizacija..."

    cout << S.size << " " << S.element[0]; // ispise "0 5"

    return 0;
}
```

Destruktori

Slično, često puta treba nešto napraviti kada objekt završava životni vijek (npr. osloboditi memoriju) $\rightsquigarrow$ destruktor

- Ime destruktora = ime strukture ispred koje ide znak `~` (tilda).
- Također nemaju povratnu vrijednost.
- Automatski se sami pozivaju kada varijabla nestaje.

```
struct stack
{
    int size, element[100];

    ~stack()
    {
        cout << "Stack je gotov!";
    }

    ...
};
```

Zadatak 2

Što ispisuje ovaj program?

```
struct test
{
    test()
    {
        cout << "Rodio se novi test!";
    }
    ~test()
    {
        cout << "Nestao je jedan test!";
    }
};

int main( void )
{
    test A, B;
    cout << "nesto";
    return 0;
}
```

Konstruktori s parametrima

Konstruktor može imati i parametre. Destruktor nikada.

```
struct stack
{
    int size, *element;

    stack( int maxSize )
    {
        element = (int *) malloc( maxSize * sizeof( int ) );
        size = 0;
    }

    ~stack() { free( element ); }
};
```

Konstruktori s parametrima

Tada objekte treba deklarirati s odgovarajućim parametrima koji se onda prosljeđuju konstruktoru.

```
int main( void )
{
    stack S( 50 );  // stog sa 50 elemenata
    stack T( 100 ); // stog sa 100 elemenata

    return 0;
}
```

C++ dozvoljava više funkcija istog imena, pa tako i konstruktora.

- Konstruktor koji ne prima niti jedan parametar zovemo defaultni konstruktor.
- Samo ako postoji defaultni konstruktor možemo deklarirati varijable sa npr. `stack S;`
- Ako ne napišemo niti jedan konstruktor, onda implicitno postoji defaultni konstruktor koji ne radi ništa.

Zadatak 3

1. Nadopunite implementaciju strukture `tocka` konstruktorom koji prima početne koordinate točke kao parametre.

2. Dodajte i destruktor koji ispisuje na ekran koordinate točke prije njezinog uništenja.

3. Uvjerite se da možete deklarirati varijable sa `tocka A, B;` samo ako uz konstruktor iz 1 postoji i defaultni konstruktor.

Dinamička alokacija memorije

Alokacija u C-u:

```
struktura *p;

p = (struktura*) malloc( sizeof(struktura) );
init_struktura( *p );
```

Dealokacija u C-u:

```
destruct_struktura( *p );
free( p );
```

Dinamička alokacija memorije

Problem u C++-u:

- `malloc` ne poziva konstruktor, a `free` ne poziva destruktor!

```
struct test
{
    test() { cout << "Rodio se novi test!"; }
    ~test() { cout << "Nema ga vise!"; }
};

int main( void )
{
    test *A;

    A = (test *)malloc( sizeof( test ) );
    // nista se ne ispise, ne pozovu se konstr/destr!!!
    free( A );

    return 0;
}
```

Dinamička alokacija memorije

C++: nove naredbe `new` i `delete` koje pozivaju konstruktor i destruktor!

```
struct test
{
    test() { cout << "Rodio se novi test!"; }
    ~test() { cout << "Nema ga vise!"; }
};

int main( void )
{
    test *A;

    A = new test; // ispise se "Rodio se novi test"
    delete A;     // ispise se "Nema ga vise"

    return 0;
}
```

Dinamička alokacija memorije

Operator `new`:

1. Alocira memoriju na heapu za strukturu (njene varijable).
2. Pozove konstruktor.
3. Vraća pokazivač na alociranu memoriju (tipa `struktura*`).

```
struktura *p = new struktura;
```

Operator `delete`:

1. Pozove destruktor.
2. Oslobodi zauzetu memoriju s heapa.

```
delete p;
```

Dinamička alokacija memorije

`new` se može koristiti i s konstruktorom koji prima parametre:

```
struct hop
{
    hop( int a ) { cout << a; }
};

int main( void )
{
    hop *A, *B;

    A = new hop( 5 );  // ispise "5"
    B = new hop( 12 ); // ispise "12"

    delete A; delete B;

    return 0;
}
```

Dinamička alokacija polja

`new` može alocirati i polje struktura; za svaku se pozove konstruktor (samo bez parametara)!

```
struct hop
{
    int a;
    hop() { cout << "Novi hop!"; }
};

int main( void )
{
    hop *A;

    A = new hop[5]; // novo polje od 5 hop-ova
    A[2].a = 5; A[1].a = 7;
    delete[] A; // uoci []; za svaki se pozove destr.

    return 0;
}
```

Zadatak 4

1. Učitajte prirodni broj n i koordinate n točaka u ravnini.
2. Odredite par točaka koji je najmanje i par točaka koji je najviše udaljen.