```cpp
#include <iostream>

using namespace std;

//Klasa student predstavlja studenta.
//Sastoji se od jedne članske varijable (id)
//i dvije metode (SetId i GetId)
class Student {
	int id;
	public:
		//deklaracije metoda
		void SetId(int id);
		int GetId();
};

//Definicija metode SetId klase Student
void Student::SetId(int id){
	this->id=id;
}

//Definicija metode GetId klase Student
int Student::GetId(){
	return this->id;
}

int main(){
	//Stvaranje objekta marko klase Student
	Student marko;
	//Poziv metode SetId objekta student klase Student
	marko.SetId(3);
	//Poziv metode GetId objekta student klase Student
	//te preusmjeravanje vrijednosti koje metoda vraća
	//na standardni izlaz
	cout << marko.GetId();
	//Privatnoj varijabli id objekta student klase Student
	//se ne može pristupiti direktno!
	//student.id = 3; //Greška!
	//cout << student.id; //Greška!
	return 0;
}
```

```cpp
#include <iostream>

using namespace std;

//Klasa student predstavlja studenta.
//Sastoji se od jedne članske varijable (id)
//Deklarirana je i metode GetId
//Definirani su i konstruktori i destruktor
class Student {
	int id;
	public:
		//Podrazumijevani konstruktor
		//Izvršava se za Student damir;
		Student(){
			this->id = 0;
			cout << "Stvaram objekt s id=" << this->id << endl;
		}
		//Konstruktor s parametrom int id
		//Izvršava se za Student vedran(1) i Student *kresimir = new
Student(2);
		Student(int id) {
			this->id = id;
			cout << "Stvaram objekt s id=" << this->id << endl;
		}
		//Destruktor
		//Izvršava se kada objekt ode iz dosega ili
		//kada se nad objektom pozove delete
		~Student(){
			cout << "Unistavam objekt s id=" << this->id << endl;
		}
		int GetId() {return this->id;}
};

int main(){
	//deklaracija objekata na stogu
	Student vedran(1), damir;
	//deklaracija objekta na heapu
	Student *kresimir = new Student(2);
	cout << "U gl. prog.: " << vedran.GetId() << endl;
	cout << "U gl. prog.: " << damir.GetId() << endl;
	cout << "U gl. prog.: " << kresimir->GetId() << endl;
	//objekte na stogu nije potrebno oslobađati, već
	//se samo oslobađaju po izlasku iz dosega
	//objekte na gomili je potrebno eksplicitno osoboditi (delete)
	delete kresimir;
	return 0;
}
```

```cpp
#include <stdio.h>
#include <string.h>
#include <iostream>

using namespace std;

//Klasa student predstavlja studenta.
//Sastoji se od jedne članske varijable (int id)
//i imena (char * name)
//Deklarirana je i metoda GetId
//Definirani su i konstruktori i destruktor
//u kojima se zauzima i oslobađa memorija
//za pohranu imena (znakovnog niza)
class Student {
	int id;
	char* name;
	public:
		Student(int id,char *name) {
			this->id = id;
			//Potrebno je u objektu zauzeti istu količinu memorije
			//kao što ima i ulazni znakovni niz
			this->name =(char*) malloc(strlen(name) + 1);
			//ulazni znakovni niz je potrebno prekopirati
			//u objekt
			strcpy(this->name, name);
			cout << "Stvaram objekt s id=" << this->id << " i imenom:
" << this->name << endl;
		}
		~Student(){
			cout << "Unistavam objekt s id=" << this->id << " i
imenom: " << this->name << endl;
			//Pri uništavanju objekta potrebno je osoboditi
			//memoriju zauzetu u konstruktoru
			free(this->name);
		}
		int getId() {return this->id;}
};

int main(){
	Student gordan(420, "Gogo");
	return 0;
}
```

```
#include <stdio.h>
#include <string.h>
#include <iostream>

using namespace std;

//Klasa student predstavlja studenta.
//Sastoji se od jedne članske varijable (int id)
//i imena (char * name)
//Deklarirana je i metoda GetId
//Definirani su i konstruktori i destruktor
//u kojima se zauzima i oslobađa memorija
//za pohranu imena (znakovnog niza)
//Definiran je i tzv. kopirajući konstruktor
//Student(const Student &initObj)
class Student {
	int id;
	char* name;
	public:
		Student(int i,char *str) {
			this->id = i;
			this->name =(char*) malloc(strlen(str) + 1);
			strcpy(this->name, str);
			cout << "Stvaranje objekta s id=" << id << endl;
		}
		//Kopirajući konstruktor kojim se definira način kopiranja
		//objekta u trenu kada se obavlja pridruživanje
		//ili kada se objekt prenosi u funkciju
		Student(const Student &sourceObject) {
			this->id = sourceObject.id;
			this->name =(char*) malloc(strlen(sourceObject.name) +
1);
			strcpy(this->name, sourceObject.name);
			cout << "Kopiranje objekta s id=" << this->id << endl;
		}

		~Student(){
			cout << "Unistavanje objekta s id=" << id << "; adresa
name=" << (int)this->name << endl;
			free(name);
		}
		int getId() {return id;}
};

void PohvaliStudenta(Student student){
	cout << "Student s id=" << student.getId() << " je dobio pohvalnicu!"
<< endl;
}

int main(){
	Student vedran(420, "Vedran");
	Student noviVedran = vedran;
	PohvaliStudenta(noviVedran);
	return 0;
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

class Trokut {
  public:
    float a, b, c;
    Trokut () {
            a = b = c = 0;
    }
    Trokut (float a0, float b0, float c0) {
            a = a0;
            b = b0;
            c = c0;
    }
    float O() {
            return a + b + c;
    }
    float P() {
            float s = O() / 2.f;
            return sqrt(s * (s-a) * (s-b) * (s-c));
    }
    ~Trokut () {
            printf ("Nestajem (%f, %f, %f)\n", a, b, c);
    }
};

int main()
{
      Trokut t1;
      Trokut *t2 = new Trokut (3.f, 4.f, 5.f);
      Trokut &t3 = Trokut (4, 5, 6);

      printf("%f %f\n", t1.O(), t1.P());
      printf("%f %f\n", t2->O(), t2->P());
      printf("%f %f\n", t3.O(), t3.P());

      delete t2;
      return 0;
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

class KompleksniBroj {
      private:
            double re;
            double im;
      public:
            KompleksniBroj(double real, double imaginary) {  //constructor
                  re = real;
                  im = imaginary;
            }

    KompleksniBroj(const KompleksniBroj &other) { // copy constructor
      this->re = other.re;
      this->im = other.im;
    }

    KompleksniBroj() { // constructor
      re = 0;
      im = 0;
    }

    //  KompleksniBroj::~KompleksniBroj() { //destructor
    //        printf("\nUnistavam objekt\n");
    //    }

            KompleksniBroj operator +(KompleksniBroj other) {  //overload
operatora +
              KompleksniBroj temp;
              temp.re = re + other.re; // this->re + other.re
              temp.im = im + other.im; // this->im + other.im
                  return temp;
            }

            KompleksniBroj KompleksniBroj::operator *(KompleksniBroj other)
{ //overload operatora *
                  return KompleksniBroj(re*other.re - im*other.im,
re*other.im + im*other.re);
            }

    KompleksniBroj KompleksniBroj::operator -(KompleksniBroj other) {
//overload operatora -
      KompleksniBroj temp;
              temp.re = re - other.re;
              temp.im = im - other.im;
                  return temp;
    }

    KompleksniBroj KompleksniBroj::Konjugiraj() {
      KompleksniBroj tmp;
      tmp.re = this->re;
      tmp.im = this->im * -1.0;
      return tmp;
    }

    double KompleksniBroj::Modul() {
      double z;
      z=sqrt((re*re)+(im*im));
      return z;
```

```
    }

    KompleksniBroj KompleksniBroj::operator /(KompleksniBroj other) {
//overload operatora /
      double div;

      div=(other.re * other.re) + (other.im * other.im);
      if (!div) throw "Dijeljenje s nulom!";
      KompleksniBroj tmp;

      tmp.re =(re * other.re) + (im * other.im);
      tmp.re /= div;
      tmp.im = (im * other.re) - (re * other.im);
      tmp.im /=div;
      return tmp;
    }

    bool KompleksniBroj::operator ==(KompleksniBroj other) { //overload
operatora ==
       return (re == other.re)&&(im==other.im) ? 1 : 0;
    }

    void KompleksniBroj::Ispisi() {
       printf("%.2lf%+.2lfi", re, im);
    }

// napraviti overload: ++, +=, -=, --, *=, /=
// dodati funkciju za čitanje kompl. broja s tipkovnice
// dodati funkcije za ispis samo realnog i samo imaginarnog dijela
};

int main() {
      KompleksniBroj a(-5,5);
      KompleksniBroj b(3,3);

  KompleksniBroj c = b;          // poziva copy constructor
  KompleksniBroj *d = new KompleksniBroj();   //stvara objekt na heapu

  printf("\n\nKompleksni broj a je: ");
  (a+b).Ispisi();
  return 0;

  printf("\n\nKompleksni broj a je: ");
  a.Ispisi();

      printf("\n\nKompleksni broj b je: ");
  b.Ispisi();

  printf("\n\nKompleksni broj d je: ");
  d->Ispisi();

  printf("\n\nKonjugirani kompleksni broj b je: ");
  b.Konjugiraj().Ispisi();

  printf("\n\nModul kompleksnog broja b je %.2lf", b.Modul());

  printf("\n\nKompleksni broj c (c=b) je: ");
  c.Ispisi();

  c = c + b;
```

```
  printf("\n\nKompleksni broj c nakon c=c+b je: ");
  c.Ispisi();

  (c == b) ? printf ("\n\nc = b") : printf("\n\nc != b");

  printf("\n\nZbrajanje: a + b = ");
  *d = a + b;
  d->Ispisi();

  printf("\n\nOduzimanje: a - b = ");
  *d = a - b;
  d->Ispisi();

  printf("\n\nMnozenje: a * b = ");
  *d = a * b;
  d->Ispisi();

  try {
            *d = a / b;
            printf("\n\nDijeljenje: a / b = ");
            d->Ispisi();
            printf("\n\n");

  } catch (const char *poruka) {
    printf("\n\n%s\n\n", poruka);
  }

  delete d; // brisi d s heapa
  system("PAUSE");
  return EXIT_SUCCESS;
}
```

```
#include <iostream>

using namespace std;

class Tester {
	int   integerMember;
	char *stringMember;
	public:
		Tester(){
			printf("Constructor\n");
		}
		~Tester() {
			printf("Dest:%d\n", this->integerMember);
		}
		int GetIntegerMember(){
			return this->integerMember;
		}
		void SetIntegerMember(int integerMember){
			this->integerMember = integerMember;
		}
		char* GetStringMember(){
			return this->stringMember;
		}
		void SetStringMember(char *stringMember){
			this->stringMember = stringMember;
		}
};

void DemoFunction()
{
	float *pFloat = new float;
	int *pInt = new int[10];
	char *pString = new char[20];
	delete pFloat;
	delete []pInt;
	delete []pString;
}

int main(int argc, char* argv[])
{
	//Stvaranje pojedinačnih objekata
	//na stogu
	Tester stackInstance;
	stackInstance.SetIntegerMember(1);
	//Stvaranje pojedinačnih objekata
	//na gomili
	Tester *heapInstance = new Tester();
	heapInstance->SetIntegerMember(2);
	//Brisanje objekta stvorenog na gomili
	delete heapInstance;

	//Stvaranje polja objekata (polje smješteno na stogu)
	//Da bi se ova linija uspješno izvršila podrazumijevani konstruktor
mora biti definiran
	Tester testerArrayStack[10];
	for( int i=0; i<10; i++ ){
		testerArrayStack[i].SetIntegerMember(10+i);
	}
	//Stvaranje polja objekata (polje smješteno na gomili)
	Tester *testerArrayHeap = new Tester[10];
	for( int i=0; i<10; i++ ){
```

```
            testerArrayHeap[i].SetIntegerMember(20+i);
        }
        //Polje stvoreno na gomili je potrebno eksplicitno obrisati
        //obavezno koristiti delete [] (uglate zagrade)
        delete [] testerArrayHeap;
        return 0;
}
```

```
/* RedListom.c */
#include <stdlib.h>
#include <stdio.h>
#include <time.h>

typedef int tip;
//Objektna implementacija reda listom
class Red {
      //Definiranje struktura podataka
      //za pohranu čvorova liste

      struct at {
            tip element;
            struct at *sljed;
      };
      typedef struct at atom;
      //Privatna varijabla klase koja
      //sadrži pokazivač na ulaz i izlaz iz reda
      //(liste od koje se red sastoji)
      atom *ulaz, *izlaz;
      //Privatna metoda
      void obrisiRed();
public:
      //Javne metode: konstruktor i destruktor
      //te metode Dodaj i Skini (iz reda)
      Red();
      ~Red();
    int Dodaj (tip element);
      int Skini (tip *element);
};

//Definicija metode za brisanje svih elemenata iz reda
//Metoda poziva metodu skini dok ne skine i posljednji element iz reda
void Red::obrisiRed(){
      int element;
      while(1){
            if(Skini(&element) == 0){
                  break;
            }
      }
}

//U konstruktoru je potrebno inicijalizirati pokazivače
//na ulaz i izlaz na NULL
Red::Red(){
      this->ulaz = NULL;
      this->izlaz = NULL;
}

//U destruktoru se poziva privatna metoda koja prazni red
Red::~Red(){
      this->obrisiRed();
}

//Definicija metode Dodaj koja sada radi s this->ulaz i this->izlaz.
//this->ulaz i this->izlaz su privatne članske varijable objekta
//klase red
int Red::Dodaj (int element) {
      atom *novi;
      if (novi = (atom *)malloc (sizeof (atom))) {
            novi->element = element;
```

```
            novi->sljed = NULL;
            if (this->izlaz == NULL) {
                  this->izlaz = novi;
            } else {
                  (this->ulaz)->sljed = novi;
            }
            this->ulaz = novi;
            printf("Na adresu %p dodao sam %d\n", novi, element);
            return 1;
      }
      return 0;
}

//Definicija metode skini koja također radi s this->ulaz i this->izlaz.
int Red::Skini (int *element) {
      atom *stari;
      if (this->izlaz) {
            *element = (this->izlaz)->element;
            stari = this->izlaz;
            this->izlaz = (this->izlaz)->sljed;
            printf ("Skidam s adrese %p\n", stari);
            free (stari);
            if (this->izlaz == NULL) this->ulaz = NULL;
            return 1;
  }
      return 0;
}

int main(){
      //Stvaranje objekta klase Red na heapu
      Red *red = new Red();
      red->Dodaj(3);
      red->Dodaj(4);
      int element;
      red->Skini(&element);
      //Brisanje objekta klase Red s heapa
      delete red;
      return 0;
}
```

```
#include <stdio.h>
#include <string.h>
#include <iostream>

using namespace std;

//Objektna implementacija stoga listom
class Stog{
      //Definiranje struktura podataka
      //za pohranu čvorova liste
      typedef int tip;
      struct at {
            tip element;
            struct at *sljed;
      };
      typedef struct at atom;

      //Privatna varijabla klase koja
      //sadrži pokazivač na prvi element stoga
      //(liste od koje se stog sastoji)
      atom *vrh;
      //Privatna metoda
      void obrisiStog();
public:
      //Javne metode: konstruktor i destruktor
      //te metode Dodaj i Skini (sa stoga)
      Stog();
      ~Stog();
    int Dodaj (tip element);
      int Skini (tip *element);
};

//Definicija metode za brisanje svih elemenata sa stoga
//Metoda poziva metodu skini dok ne skine i posljednji element sa stoga
void Stog::obrisiStog(){
      int element;
      while(1){
            if(Skini(&element) == 0){
                  break;
            }
      }
}

//U konstruktoru je potrebno inicijalizirati pokazivač
//na prvi element stoga (liste) na NULL
Stog::Stog(){
      this->vrh = NULL;
}

//U destruktoru se poziva privatna metoda koja prazni stog
Stog::~Stog(){
      this->obrisiStog();
}

//Definicija metode Dodaj koja sada radi s this->vrh.
//this->vrh je privatna članska varijabla objekta
//klase stog
int Stog::Dodaj (tip element) {
      atom *novi;
      if ((novi = (atom *) malloc(sizeof(atom))) != NULL) {
            novi->element = element;
```

```
            novi->sljed = this->vrh;
            printf("Na adresu %p dodao sam %d, a sljedeci je %p\n",
      novi, element, this->vrh);
            this->vrh = novi;
            return 1;
      }
      else
            return 0;
}

//Definicija metode skini koja također radi s this->vrh
int Stog::Skini (tip *element) {
      atom *pom;
    if (this->vrh == NULL) return 0;
      *element = this->vrh->element;
      printf ("Skidam s adrese %p\n", this->vrh);
      pom = this->vrh->sljed;
      free(this->vrh);
      this->vrh = pom;
      return 1;
}

int main(){
      //Stvaranje objekta klase Stog na heapu
      Stog *stog = new Stog();
      stog->Dodaj(3);
      stog->Dodaj(4);
      int element;
      stog->Skini(&element);
      //Brisanje objekta klase stog s heapa
      delete stog;
      return 0;
}
```

```
#include <stdio.h>
#include <string.h>
#include <iostream>

using namespace std;

int main(){
	//Stvaranje objekta klase Stog na heapu
	Stog *stog = new Stog();
	stog->Dodaj(3);
	stog->Dodaj(4);
	int element;
	stog->Skini(&element);
	//Brisanje objekta klase stog s heapa
	delete stog;
}
```