Sa proceduralnog na OO programiranje: klase i objekti

```c
/* File: stack.h */

#define MaxStackSize 256

struct Stack;

void stack_init (Stack* this);
int stack_push (Stack* this, unsigned in);
int stack_pop (Stack* this, unsigned* out);

struct Stack {
  unsigned stack[MaxStackSize];  // Stack
  int sp; // Stack pointer
};


/* File stack.c */
#include “stack.h"

void stack_init (Stack* this) {
  this->sp = 0;
}

int stack_push (Stack* this, unsigned in) {
  if (this->sp==MaxStackSize) return -1;
  this->stack[this->sp++] = in;
  return 0;
}

int stack_pop (Stack* this, unsigned* out) {
  if (this->sp==0) return -1;
  *out = this->stack[--this->sp];
  return 0;
}
```

```c
// File: stack.h

const int MaxStackSize = 256;

class Stack {
public:
  Stack ();
  int push (unsigned in);
  int pop (unsigned* out);

private:
  unsigned stack[MaxStackSize];  // Stack
  int sp; // Stack pointer
};


// File stack.cpp
#include "stack.h"

Stack::Stack () {
  this->sp = 0;
}

int Stack::push (unsigned in) {
  if (this->sp==MaxStackSize) return -1;
  this->stack[this->sp++] = in;
  return 0;
}

int Stack::pop (unsigned* out) {
  if (this->sp==0) return -1;
  *out = this->stack[--this->sp];
  return 0;
}
```

Sa proceduralnog na OO programiranje: klase i objekti

- Na ovaj način postiže se potpuna efikasnost — kod generisan za funkcije članice i njihov poziv, kao i pristup članovima objekata podjednako je efikasan kao i ekvivalentan C kod, jer nema nikakve dodatne režije (*overhead*)
- Sa druge strane, postiže se višestruka dobit u pisanju i strukturiranju programa, kao i u podršci prevodioca koji vrši odgovarajuće automatske provere (samo u vreme prevođenja), umesto da sve to zavisi samo od discipline i pedantnosti programera:
  - Koncept klase kao apstrakcije se jasno ističe i podstiče programera da je na ispravan način definiše, odnosno grupiše strukturu i ponašanje (operacije nad tom strukturom)
  - Struktura i operacije nad njom su "upakovani" u jedinstvenu, jasno istaknutu logičku celinu, i pripadaju *oblasti važenja klase* (*class scope*)
  - Podržana je enkapsulacija, jer prevodilac kontroliše i ne dozvoljava neovlašćen pristup do implementacije klase (privatnih članova)
  - Postoji koncept konstruktora (i destruktora), pa je inicijalizacija objekata lakša, očiglednija i manje podložna greškama