Algoritamska dekompozicija

- Fundament proceduralnog programiranja jeste *algoritamska* (*proceduralna*) *dekompozicija* (*algorithmic, procedural decomposition*):
  - u centru pažnje jeste zadatak ili posao koji treba uraditi, za koji se definiše *postupak* (*procedura, algoritam*)
  - taj postupak se inicijalno deli na *korake*, najpre visokog nivoa apstrakcije i velike granularnosti ("da bi se to uradilo, potrebno je najpre uraditi *a*, pa onda *b*, potom, ako je ispunjen uslov *c*, treba uraditi *d*, inače *e* itd); ovi koraci definišu se kao *procedure* (potprogrami)
    - dalje se svaki krupniji korak deli istim postupkom na manje korake (rekurzivna primena istog postupka dekompozicije), sve dok se ne dođe do najsitnijeg nivoa granularnosti, onog koji se može izraziti elementarnim konstruktima programskog jezika koji se koristi
- Procedure su takođe proizvod apstrakcije: određeni (krupniji) korak se apstrahuje kao celina, pri čemu se njegova razrada i detalji odlažu za kasnije, jer za korišćenje nisu bitni (bitan je samo interfejs)
- Primer: Program koji održava *n* stek-mašina koje treba da izvršavaju komande (operacije *add*, *sub*, *push*, *pop* itd). Na ulazu se nalaze komande u tekstualnom obliku, svaka u po jednom redu. Svaka komanda je u formatu *i*:*command*, gde je *i* redni broj stek-mašine na koju se odnosi komanda u nastavku. Sa ulaza se učitava jedan po jedan znak operacijom *getChar*. Zapis komande sa ulaza treba prevesti pre nego što se izvrši.

Algoritamska dekompozicija

```
enum OpCode { add, sub, …};

int main () {

  int out;
  string cmdin;
  OpCode cmdout;

  while (readCommand(out,cmdin)) {
    translate(cmdin,cmdout);
    performCmd(out,cmdout);
  }

}
```

```
bool readCommand (int& out, string& cmd) {

  bool ret = readOut(out);
  if (!ret) return false;

  return readCmd(cmd);

}

bool readOut (int& out) {

  out = 0;

  char c = getChar();
  while (isDigit(c)) {
      out = out*10 + (c-'0');
      c = getChar();
  }

  return (c!=EOF);

}

bool readCmd (string& cmd) {
  …
}
```

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

Stack sm[N];
…

void translate (string cmdin, OpCode& cmdout) {
  if (cmdin=="ADD") cmdout = add;
  if (cmdin=="SUB") cmdout = sub;
  …
}
```

```
void performCmd (int out, OpCode cmd) {
  switch (cmd) {

    case add: {
      int op1 = pop(out);
      int op2 = pop(out);
      push(out,op1+op2);
      break;
    }

    case sub: …
    …
  }
}

int pop (int out) {
  if (sm[out].sp==0) return 0;
  else return sm[out].stack[--sm[out].sp];
}

…
```