Objektovo orientované programovanie

Niektoré princípy tvorby programov

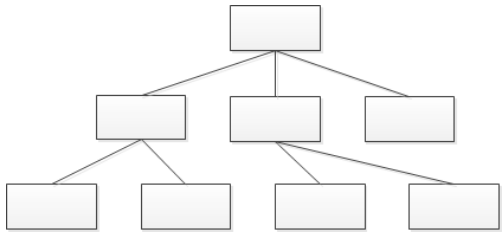
Vladislav Novák

FEI STU v Bratislave

23.9.2014

Ako zvládnuť veľké projekty

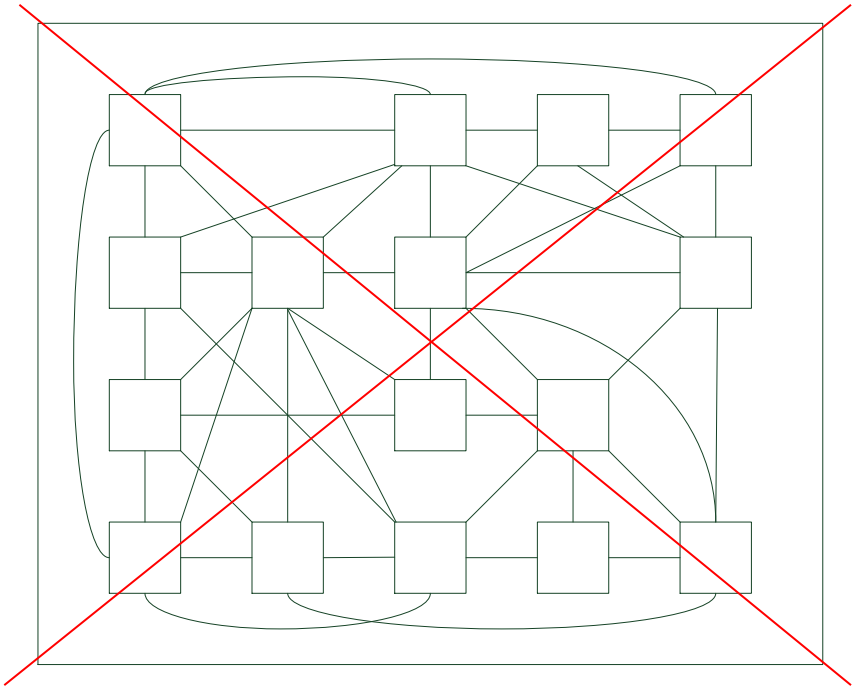
- rozdeliť veľkú úloha na menšie podúlohy (veľký systém na menšie časti)

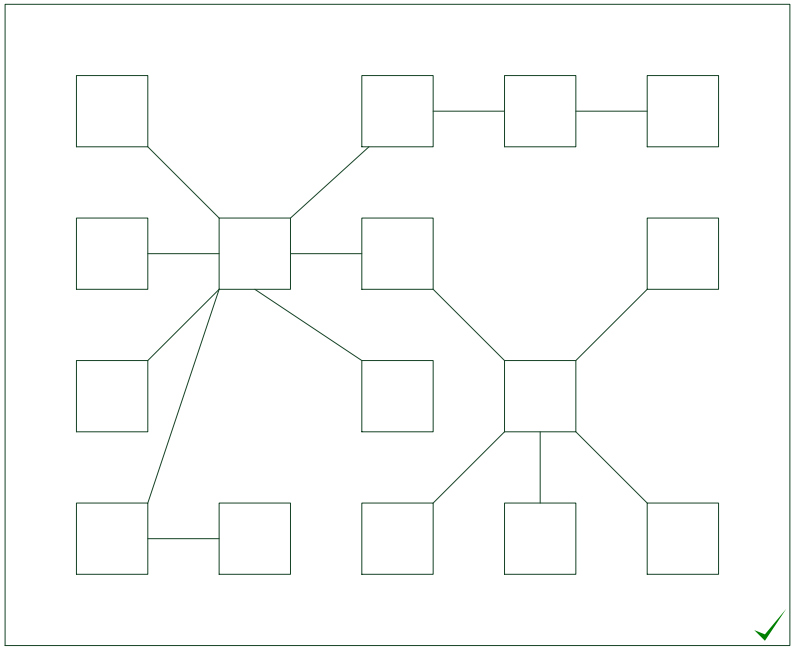


  - minimalizovať vzájomné previazanie jednotlivých častí (ideálne každú časť riešiť nezávisle na inej časti)
  - maximalizovať súdržnosť časti systému (vždy sa zaoberať práve jednou časťou systému)

Vzájomná previazanosť (coupling)

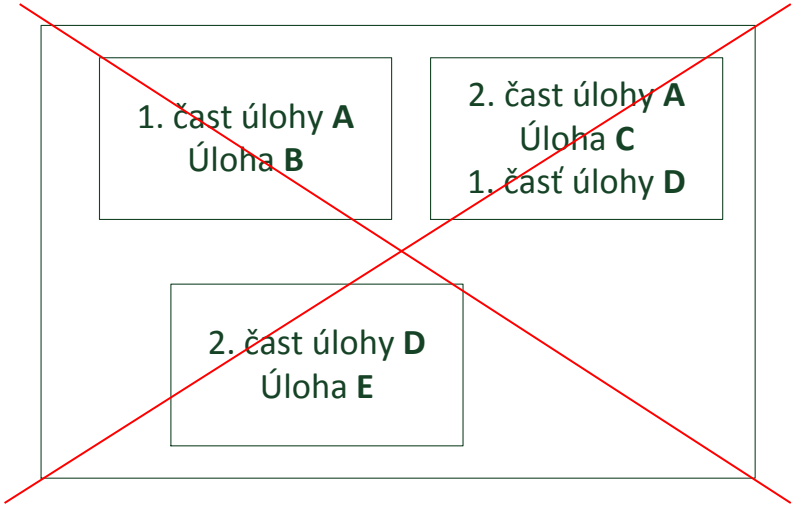
- Previazanosť je určená množstvom väzieb medzi softvérovými entitami (funkcie, triedy, metódy, balíky,.....). Previazanosť by mala byť minimalizovaná. Minimalizácia množstva väzieb medzi entitami znižuje ich vzájomnú závislosť.

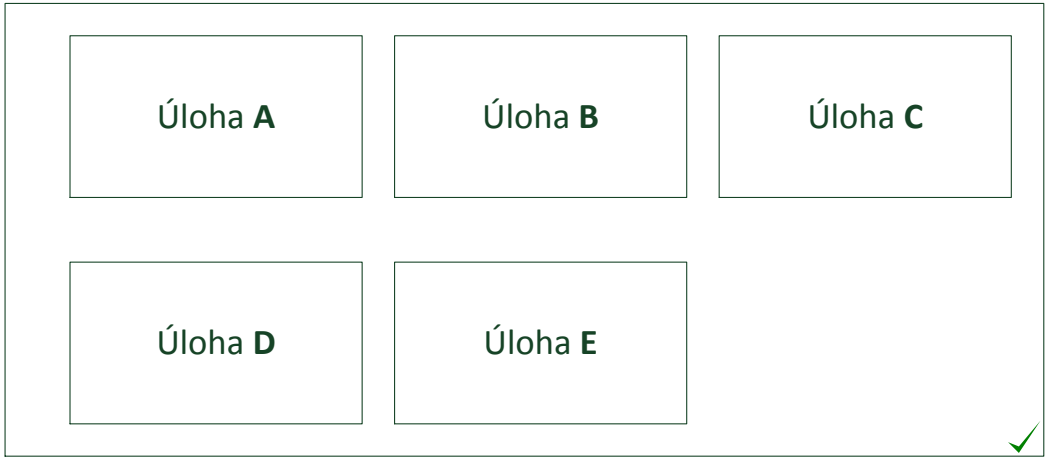




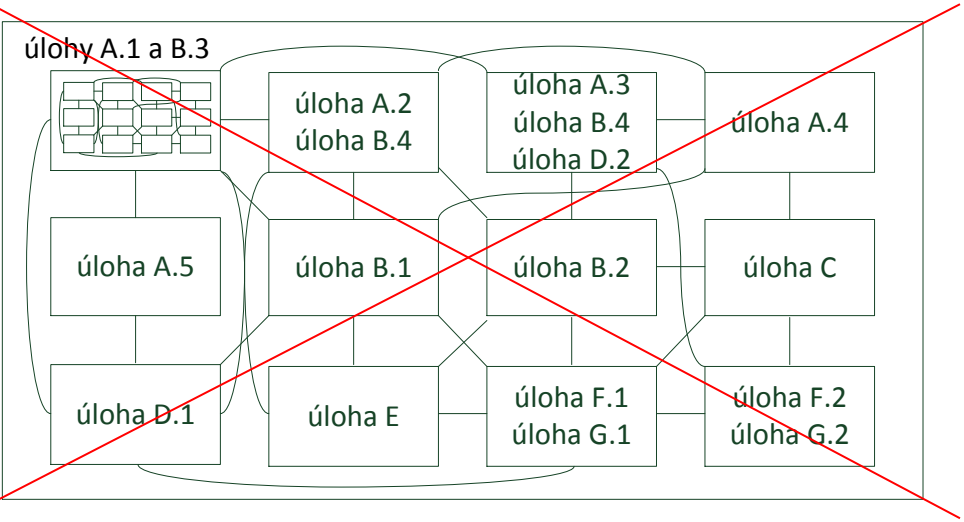
Súdržnosť (cohesion)

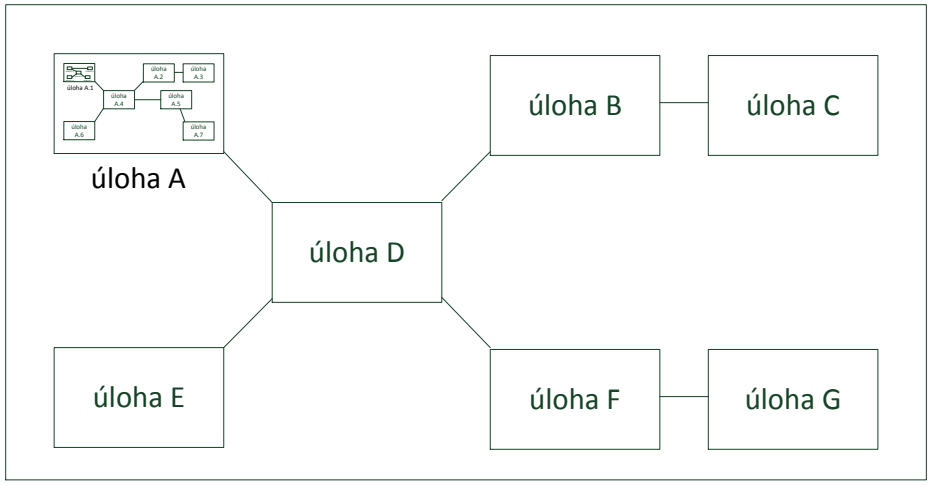
- To čo spolu súvisí je vhodné implementovať práve v jednej softvérovej entite (funkcia, trieda, metóda, balík). Nie je vhodné implementovať niekoľko vecí v jednej entite. Súdržnosť vyjadruje dodržanie tejto zásady.
- Je vhodné maximalizovať súdržnosť. Ak je úloha zložitejšia, treba rozdeliť jednotlivé podúlohy jednotlivým podsystémom (balíkom, triedam, metódam, funkciám).





Provnanie návrhu

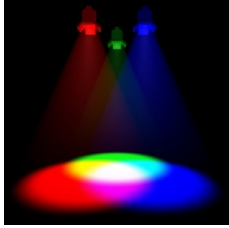


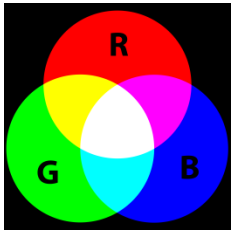


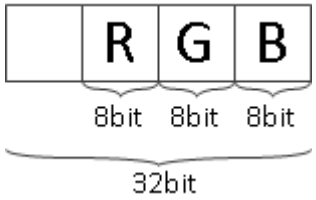
1. príklad

- Reprezenácia farby pomocou farebných zložiek

```
typedef unsigned int Color;
```

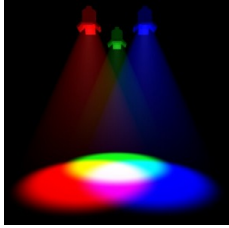


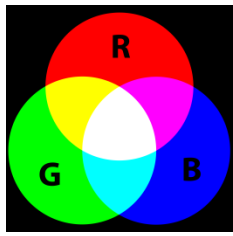


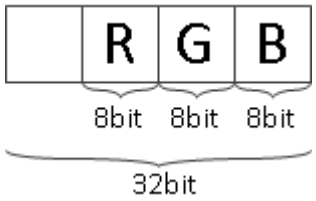



```
void nevhodnePouzitie () {
    Color color = (255<<16 | 255); //0xFF00FF //vysoka previazanost
    unsigned char red = (color>>16 & 0xFF); //vysoka previazanost
}
```

```
typedef unsigned int Color;
```







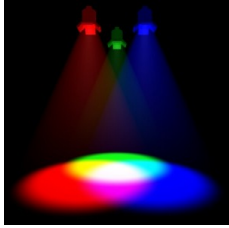
vysoká previazanosť na reprezentáciu

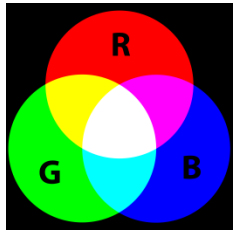
```
void nevhodnePouzitie () {
    Color color = (255<<16 | 255); //0xFF00FF //vysoka previazanost
    unsigned char red = (color>>16 & 0xFF); //vysoka previazanost
}
```

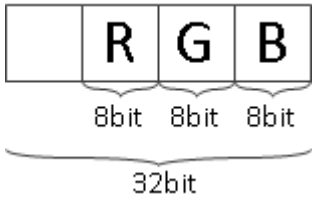
```
typedef struct {
    unsigned char red;
    unsigned char green;
    unsigned char blue;
} Color;
```

obidve reprezentácie majú svoje výhody aj nevýhody

```
typedef unsigned int Color;
```



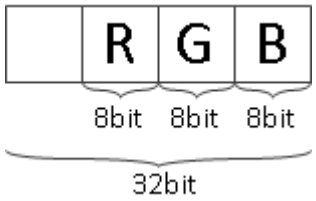


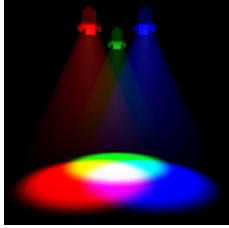


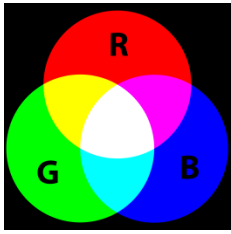

```
void nevhodnePouzitie () {
    Color color = (255<<16 | 255); //0xFF00FF //vysoka previazanost
    unsigned char red = (color>>16 & 0xFF); //vysoka previazanost
}
```

```
typedef struct {
    unsigned char red;
    unsigned char green;
    unsigned char blue;
} Color;
```

```
typedef unsigned int Color;

Color CreateColor(unsigned char red, unsigned char green, unsigned char blue) {
    return red<<16 | green<<8 | blue;
}
unsigned char GetRed(Color color) {
    return (color >> 16) & 0xFF;
}
unsigned char GetGreen(Color color) {
    return (color >> 8) & 0xFF;
}
unsigned char GetBlue(Color color) {
    return (color >> 0) & 0xFF;
}
```





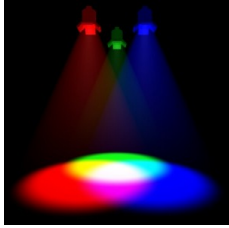


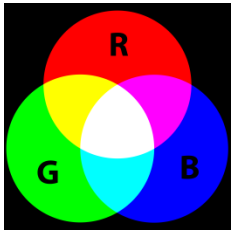
~~void **nevhodnePouzitie** () {~~
~~Color color = (255<<16 | 255); //0xFF00FF //vysoka previazanost~~
~~unsigned char red = (color>>16 & 0xFF); //vysoka previazanost~~
~~}~~

```
void vhodnePouzitie () {
    Color color = CreateColor(255,0,255);
    unsigned char red = GetRed(color);
    printf("farba = %d\n", red);
}
```

```
typedef struct {
    unsigned char red;
    unsigned char green;
    unsigned char blue;
} Color;
```

```
typedef struct {
    unsigned char red;
    unsigned char green;
    unsigned char blue;
} Color;

Color CreateColor(unsigned char red, unsigned char green, unsigned char blue) {
    Color newColor = {.red=red,.green=green,.blue=blue};
    //Color newColor = {red, green, blue}; //previazanost na poradie
    return newColor;
}
unsigned char GetRed(Color color) {
    return color.red;
}
unsigned char GetGreen(Color color) {
    return color.green;
}
unsigned char GetBlue(Color color) {
    return color.blue;
}
```





rovnaký kód ako na predchádzajúcom slajde

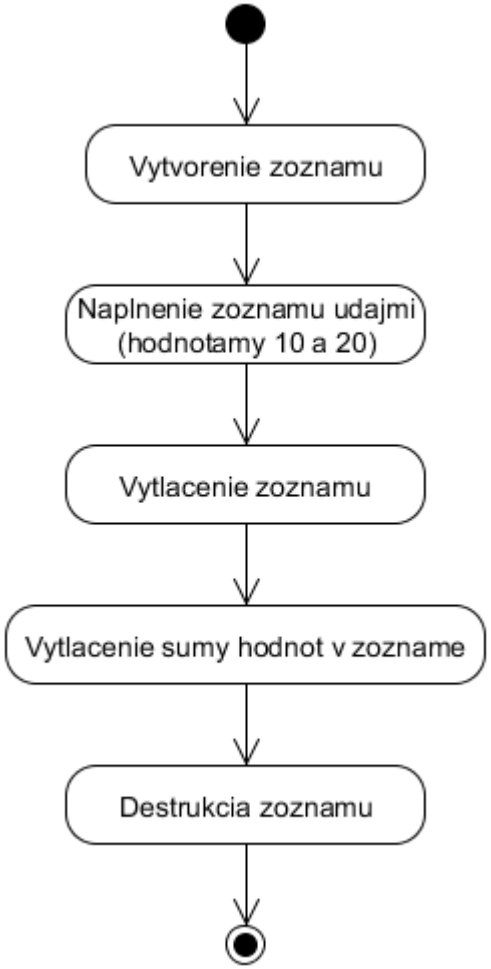
```
void vhodnePouzitie () {
    Color color = CreateColor(255,0,255);
    unsigned char red = GetRed(color);
    printf("farba = %d\n", red);
}
```

```
void pouzitie (){
    Color color = {.red = 255, .green=0, .blue=255};
    unsigned char red = color.red;
}
```

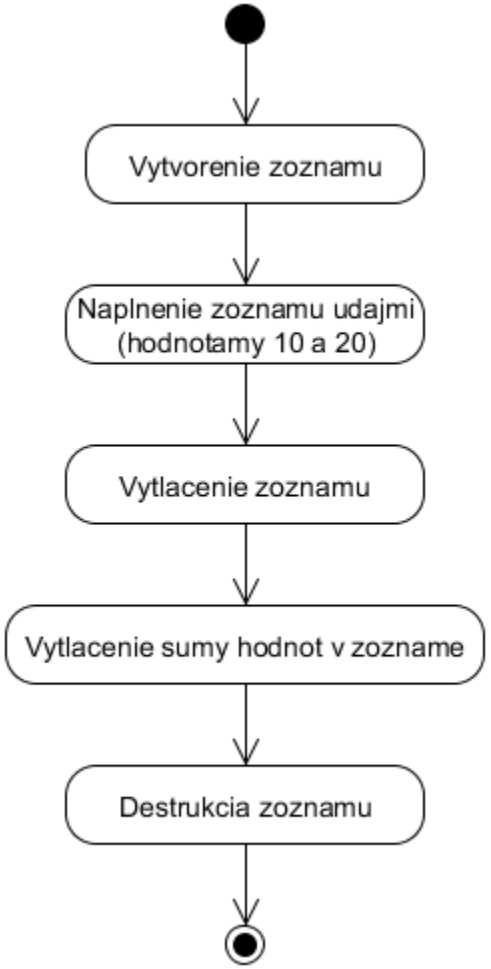
ako zabrániť takémuto použitiu?

2. príklad

- Práca so zreťazeným zoznamom



```
int mainDobre() {
    Item * list;

    //vytvorenie noveho zoznamu
    list = CreateList(); //nazov funkcie vyjadruje naco funkcia sluzi

    //pridanie hodnot 10 a 20
    list = IntestToList(list, 10);
    list = IntestToList(list, 20);

    //tlac zoznamu
    PrintList(list);

    //vypocet sumy hodnot v zozname a jej vytlacenie
    int sum = SumOfList(list);
    printf("sucet = %d\n", sum);

    //destrukcia zoznamu
    DestroyList(list);
}
```



Nepotrebujeme poznať všetky detaily implementácie, stačí poznať rozhranie (deklarácie a popis funkcii)

```
typedef struct _item{
      int data;
      struct _item * next;
} Item;

Item* CreateItem(int data) {
      Item * item = (Item *)malloc(sizeof(Item));
      item->data = data;
      item->next = NULL;
}

Item* CreateList() {
   return NULL;
}
```

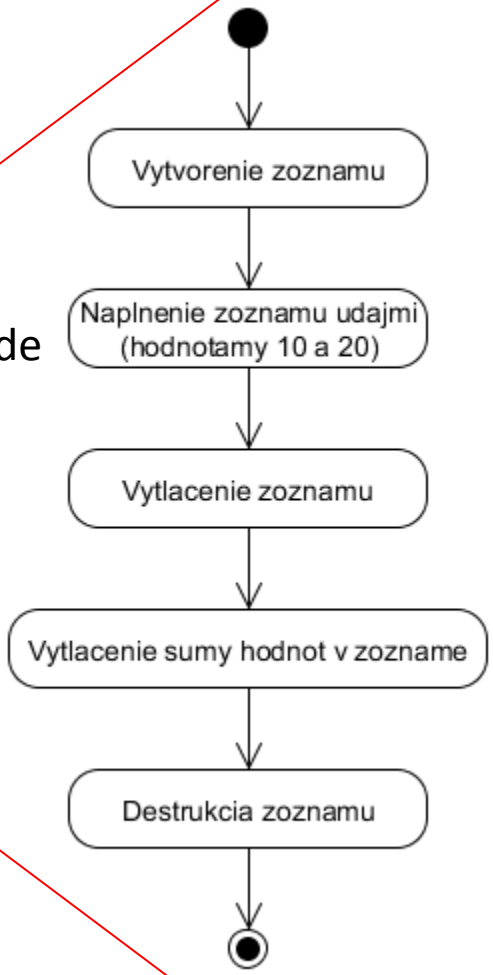
```
Item* IntestToList(Item* list, int value) {
   //uloha sa sklada z dvocho poduloh
   Item* newItem = CreateItem(value); //vytvorenie noveho prvku
   newItem->next = list; //zapojenie noveho provku do zoznamu
   return newItem;
}
```

```
void PrintList(Item* list) {
   while(list != NULL) {
      printf("%d ", list->data);
      list = list->next;
   }
   printf("\n");
}
```

```
int SumOfList(Item* list) {
   int sum = 0;
   while(list != NULL) {
      sum += list->data;
      list = list->next;
   }
   return sum;
}
```

```
void DestroyList(Item* list) {
   Item* next;
   while(list != NULL) {
      next = list->next;
      free(list);
      list = next;
   }
}
```

```
int mainZle() {
    Item* list; //cely zoznam
    Item* newItem;//pomocna premenna pri pridavani noveho prvku do zoznamu

    //vytvorenie zoznamu
    list = NULL; //musime vediet viac o struktre zretazeneho zoznamu

    //pridanie 10 a 20
    newItem = (Item*) malloc(sizeof(Item));
    InsertItemToList(list, newItem, 10); //implementacia na dalsom slajde
    list = newItem;

    newItem = (Item*) malloc(sizeof(Item));
    InsertItemToList(list, newItem, 20);
    list = newItem;

    //tlac zoznamu
    PrintList(list);

    //vypocet sumy hodnot v zozname a jej vytlacenie
    SumOfList(list); //presnejsie PrintSumOfList(list)
    //(co ak v inej casti programu budeme potrebovat iba vypocitat sumu bez tlace?)

    DestroyList(list);
}
```



názov funkcie nevyjadruje že funkcia aj tlačí výsledok

slabá súdržnosť
funkcia vykonáva 2 úlohy
(vypočítava sumu a tlačí výsledok)

```
typedef struct _item{
        int data;
        struct _item * next;
} Item;
```

```
void InsertItemToList(Item* list, Item* newItem, int value) {
    newItem->data = value;
    newItem->next = list;
}
```

```
void PrintList(Item* list) {
    while(list != NULL) {
        printf("%d ", list->data);
        list = list->next;
    }
    printf("\n");
}
```

```
void SumOfList(Item* list) {
    int sum = 0;
    while(list != NULL) {
        sum += list->data;
        list = list->next;
    }
    printf("sucet = %d\n", sum);
}
```

```
void DestroyList(Item* list) {
    Item* next;
    while(list != NULL) {
        next = list->next;
        free(list);
        list = next;
    }
}
```

3. príklad

- rozhoduje význam, nie kód

- Zadanie
  - Program pre výpočet predajných cien obchodu
  - Predajná cena bez DPH sa určí z nákupnej ceny tak, že sa ku nákupnej cene pripočíta 20%
  - Predajná cena s DPH sa vypočíta z predajnej ceny bez DPH podľa základnej sadzby

1. verzia

```
int main() {
    double nakupnaCena =  .....
    double predajnaCenaBezDph = nakupnaCena * 1.2;
    double predajnaCenaSDpha = predajnaCenaBezDph * 1.2;
}
```

~~double VypocitajCenu(double vstupnaCena) {~~
~~   return vstupnaCena * 1.2;~~
~~}~~

2. verzia

```
int main() {
    double nakupnaCena =  .....
    double predajnaCenaBezDph = VypocitajCenu(nakupnaCena);
    double predajnaCenaSDpha = VypocitajCenu(predajnaCenaBezDph);
}

double VypocitajCenu(double  vstupnaCena) {
    return vstupnaCena * 1.2;
}
```

3.verzia

```
int main() {
    double nakupnaCena =  .....
    double predajnaCenaBezDph = VypocitajPredajnuCenu(nakupnaCena);
    predajnaCenaSDph = VypocitajCenuSDph(predajnaCenaBezDph);
}

double VypocitajPredajnuCenu(double  nakupnaCena) {
    return nakupnaCena * 1.2;
}

double VypocitajCenuSDph(double  predajnaCenaBezDPH) {
    return predajnaCenaBezDPH * 1.2;
}
```

Softvérová entita

- Softvérová entita (funkcia, metóda, trieda, balík, ...) – jedna úloha – vďaka tomu sa často dá stručne pomenovať. Ale môže sa stať že stručný názov je ťažké nájsť.

Názov softvérovej entity

- príklady softvérovej entity
  - premenná
  - funkcia, metóda
  - trieda
  - balík
- Názov funkcie, ktorá vypočíta obvod kruhu
  -
  - ~~double Funkcia1(double a)~~
  -

Názov softvérovej entity

- príklady softvérovej entity
  - premenná
  - funkcia, metóda
  - trieda
  - balík
- Názov funkcie, ktorá vypočíta obvod kruhu
  -
  - ~~double Funkcia1(double a)~~
  - ~~double NastavFarbu(double datum)~~

Názov softvérovej entity

- príklady softvérovej entity
  - premenná
  - funkcia, metóda
  - trieda
  - balík
- Názov funkcie, ktorá vypočíta obvod kruhu
  - double ObvodKruhu(double polomer)
  - ~~double Funkcia1(double a)~~
  - ~~double NastavFarbu(double datum)~~

podľa názvu funkcie vieme odhadnúť načo funkcia slúži, názov argumentu vystihuje jeho význam

Komentáre

- Majme napríklad komentár ku premennej
  `int nazov;`
- Do komentára nepísať že je to globálna premenná, alebo že je to číslo typu int, ale užitočné informácie (význam premennej (čísla), prípadne ďalšie poznámky)