alokace paměti (no)

- typově orientovaná (dynamická) práce s pamětí
- klíčová slova **new** a **delete**
- jednotlivé proměnné nebo pole proměnných
- alternativa k **xxxalloc** resp. **xxxfree** v jazyce C, zůstává možnost jejich použití (nevhodné)
- new resp delete volá konstruktory resp. destruktory (což malloc a free nedělají !!!)
- lze ve třídách přetížit (volání ”globálních” ::new, ::delete)

dynamická alokace pro jednu proměnnou (objekt). Lze předepsat konstruktor

```
void* :: operator new    (size_t)
void  :: operator delete (void *)

char*  pch = (char*) new char;
// alokace paměti pro jeden prvek typu char
delete pch; // vrácení paměti pro jeden char

Komplex *kk;
kk = new Komplex(10,20); // alokace paměti
// pro jeden prvek Komplex s inicializací
// zde předepsán konstruktor se dvěm parametry
```

dynamická alokace pro pole hodnot – implicitní konstruktory

```
void* :: operator new[ ] (size_t)
void  :: delete[]        (void *)

Komplex *pck = (Komplex*) new Komplex [5*i];
// alokace pole objektů typu Komplex, volá se
// implicitní konstruktor pro každý prvek pole
```

Vysvětlete rozdíl mezi delete a delete[]

Obě dvě verze zajistí odalokování paměti.
První verze zavolá destruktory na každý prvek v poli.
Druhá verze volá destruktor pouze na jeden (první) prvek.
Každý z destruktorů se tedy chová odlišně. Zavoláme-li obyčejné delete na pole, nemusí dojít k volání destruktorů na prky (a odalokování jejich interní paměti, vrácení zdrojů…). V případě zavolání „polního“ delete na jeden prvek může dojít k neočekávaným výsledkům (například může očekávat před polem hlavičku s informacemi o počtu prvků pole. Pokud zde hlavička nebude, může dojít k chybné interpretaci dat, které zde budou).

```
delete[]  pck; // vrácení paměti pole
// destruktory na všechny prvky
delete pck;//destruktor pouze na jeden prvek!!
```

new T = new(sizeof(T)) = T::operator new (size_t)
new T[u] = new(sizeof(T)*u+hlavička) = T::operator new[ ](size_t)
new (2) T = new(sizeof(T),2) - první parametr size_t
new T(v) + volání konstruktoru

```
void T::operator delete(void *ptr)
 { // přetížený operátor delete pro třítu T
   // ošetření ukončení ”života” proměnné
  if (změna) Save(”xxx”);
  if (ptr!=nullptr)
   ::delete ptr; //”globální” delete,
// jinak možné zacyklení
…}
```

konstruktory (v novějších verzích) při nenaalokování paměti podle požadavků používají systém výjimek, konkrétně bad_alloc.

“původní” vracení nullptr (ve starších překladačích NULL) je možné ve verzi s potlačením výjimek (nutno přidat knihovnu - #include <new>)

```
char *uk = (char *)new(std::nothrow) char[10]
```