„chytré“ ukazatele – unique_ptr, shared_ptr, weak_ptr

- „obaly“ pro ukazatele, které mají „vylepšené“ vlastnosti (například při konci života odalokují naalokovanou paměť)
- „bezstarostné“ ukazatele – umožňují volněji programovat a zamezit leakům v paměti
- výhoda při výjimkách – odalokuje se paměť
- ve standardní knihovně <memory>
- unique_ptr pro unikátní vlastnictví paměti – je to „handle“ na ukazatel, při přiřazování mezi těmito _ptr se objekt přesouvá „move“
- shared_ptr pro společné sdílení paměti – obsah se „kopíruje“ – vytváří se odkazy a k odalokování dochází až při rušení posledního prvku, který na paměť odkazuje

```
{
int *pu = new int;
unique_ptr<double> dp = new double;

if ( ) return 1; // neodalokuje pu, dp se odalokuje
if ( ) throw “chyba“; // neodalokuje pu, dp se odalokuje

delete pu; // odalokuje pu
} // zanikne dp
```

ukazatele auto/unique/ shared_ptr

move_ptr **counted/ staré názvy**

„chytré“ ukazatele 34.3.1 strous

- "chytré" ukazatele
- ukazatele jsou v <memory>, -> v prostoru std
- automatické odalokování naalokované paměti přístupné pomocí tohoto objektu
- "garbage collector" - automatické odalokování v místě zániku "nosiče"
- pozor v některých částech (výjimka v konstruktoru …) je nutné ošetřit jinak

auto_ptr (před C++11)

- starý mechanizmus nahrazený unique_ptr (podobná - funkce, bezpečnější, umí pole...).
- stávají se vlastníkem předaného objektu
- "garbage collection" funkce - odalokuje vnitřní ukazatel při svém zániku.
- každý ukazatel může vlastnit pouze jeden auto_ptr (jinak by byl dvakrát odalokován)
- přiřazení dvou auto_ptr je realizováno pomocí přesunu (move přiřazení) - původní auto_ptr má nyní ukazatel nullptr
- přístup k hodnotám pomocí operátoru * (vrací typ pointeru podle typu auto_ptr) a get() (vrací ukazatel na "skutečný" uložený typ). V případě, že je auto_ptr<base> a je do ní vložen ukazatel na potomka, potom přístup přes * vrací base, zatímco get() umožní pracovat s potomkem.
- metoda release vrátí objekt, a auto_ptr "vlastní" nullptr

```
{
auto_ptr<double> // parametr double, ale jedná se o ukazatel
      apd (new double); // vytvoří se nový objekt,
// ukazatel se vloží, apd je jediným vlastníkem objektu
 *apd.get() = *apd; // dva způsoby přístupu (get lepší)
 double *up = &*apd ; // zjištění adresy, operátor * vrátí
// dereferencovaný vnitřní prvek,
// operátor & zjistí jeho adresu


} // zaniká objekt apd a součástí jeho zániku je i zrušení
// vnitřního objektu - pomocí ukazatele na double
```

unique_ptr

- stávají se (jedinými) vlastníky objektu vloženého pomocí ukazatele
- při svém zániku odalokuje odkazovaný objekt
- má specializaci pro pole
- move přiřazení
- podporuje operátory *(vhodné pro základní typy), ->(vhodné pro přístup do struktury), [] (bez pointerové aritmetiky)

pro pole

```
unique_ptr <double []> x(new double[xx]);
```

přístup
x[i] nebo x.get()[i]

shared_ptr

- ukazatel se sdílí (spoluvlastní) mezi několika shared_ptr
- součástí je počítadlo, které počítá, kolik objektů na vložený prvek odkazuje
- "odalokování" nebo zrušení probíhá tak, že se odečítá počítadlo. Poslední prvek zruší i odkazovaný objekt

- make_shared - vytvoří shared pointer tak, že z dodaného objektu si vytvoří kopii

weak_ptr
- realizuje dočasné vlastnictví
- nevlastní vložený ukazatel
- umí sdílet ukazatele s shared_ptr, aniž by je vlastnil
- pro přístup nutno zkonvertovat na shared_ptr pomocí lock (tím se "zablokuje" případné zrušení původního shared_ptr v této době)

```
weak_ptr<double> xx;
shared_ptr<double>bb (new double);
xx = bb;
...
shared_ptr aa = xx.lock();
if (xx)
   // ukazatel je v pořádku=bb ještě existuje
else
   // ukazatel není v pořádku = bb již zaniklo

// jednodušeji
if (xx.expired())
  // bb je zrušeno
else
// bb je ještě validní
```