„chytré“ ukazatele – unique_ptr, shared_ptr, weak_ptr

```
void fce(void)
{
int *pu = new int; // dynamicky alokovaná paměť
TTyp aa; // objekt třídy obsahující dynamická data

if ( … )
   throw “Err12“; // vyvolání výjimky ukončí funkci.
                  // volají se destruktory objektů -> paměť
                 // v aa je odalokována destruktorem
                 // paměť alokovaná do pu se ztratí

// nedojde-li k výjimce
delete pu; // o odalokování se musí starat programátor
} // objekt odalokuje paměť automaticky v destruktoru
```

- potřebujeme, aby se ukazatele chovaly jako by byly v objektu -> dáme je do objektu a vytvoříme mu rozhraní, aby se s nim pracovalo jako s ukazatelem
-

„chytré“ ukazatele – unique_ptr, shared_ptr, weak_ptr

```
void fce(void)
{
//shared_ptr<int> dp = new int; *dp = 3;// nefunguje s auto
shared_ptr<int> dp = make_shared<int>(3);//lepší.Lze auto dp=
   // konstruktor uloží ukazatel do proměnné dp
TTyp aa; // objekt třídy obsahující dynamická data

*dp = 3; // přetížený operátor * umožní práci s hodnotou
         // odkazovanou ukazatelem uloženým v dp

if ( … )
   throw “Err12“; // vyvolání výjimky ukončí funkci
                  // volají se destruktory objektů -> paměť
                 // v aa je odalokována destruktorem.
                 // objekt je i dp => odalokuje se i jeho paměť

// nedojde-li k výjimce
} // objekty dp i aa odalokují paměť automaticky v destruktorech
```

„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
- prototypy ve standardní knihovně <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

- přístup k hodnotám pomocí operátoru * (vrací typ pointeru podle typu xxx_ptr) a get() (vrací ukazatel na "skutečný" uložený typ).
- metoda release vrátí objekt, a xxx_ptr "vlastní" nullptr

```
shared_ptr<double> apd (new double);
// ukazatel se vloží, apd je jediným vlastníkem objektu
 *apd.get() = *apd; // dva způsoby přístupu
 double *up = &*apd ; // zjištění adresy, operátor * vrátí
// dereferencovaný vnitřní prvek,
// operátor & zjistí jeho adresu
```

„chytré“ ukazatele – unique_ptr, shared_ptr, weak_ptr

unique_ptr

- unique_ptr pro unikátní vlastnictví paměti – je to „handle“ na ukazatel,
- stávají se (jedinými) vlastníky objektu vloženého pomocí ukazatele
- při přiřazování mezi unique_ptr se objekt přesouvá „move“
- při svém zániku odalokuje odkazovaný objekt
- má specializaci pro pole
- 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]
```

„chytré“ ukazatele – unique_ptr, shared_ptr, weak_ptr

```
{
int *pu = new int;
//unique_ptr<double> dp = new double; *dp = 3.14;
unique_ptr<double> dp = make_unique<double>(3.14);
if ( ) return 1; // neodalokuje pu, dp se odalokuje
if ( ) throw “chyba“; // neodalokuje pu, dp se odalokuje

unique_ptr<double> dp2 = std::move(dp);
// dp je „prázdný“
// využívá move operator=, proto musí být na pravé straně
// rhodnota, která se vytvoří (z lhodnoty dp) pomocí move
// „normální“ operátor= by vytvořil kopii(už by nebyl unikátní)
// a proto je zakázán (nepřeloží se)

delete pu; // odalokuje pu
} // zanikne dp2 i dp (to neodalokuje nic, protože je prázdné)
```

„chytré“ ukazatele – unique_ptr, shared_ptr, weak_ptr

shared_ptr

- shared_ptr slouží pro společné vícenásobné sdílení paměti – společně s kopií shared_ptr se o ní vytváří záznam. 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
- Do funkcí předávat pomocí reference; kopie vytvářet přes kopykonstruktor nebo = (ne přes vnitřní ukazatel)

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

weak_ptr

- realizuje dočasné vlastnictví
- nevlastní vložený ukazatel, odkazuje na objekt zprostředkovaně přes vlastníka (share_ptr, unique_ptr), umí sdílet ukazatele s shared_ptr, aniž by je vlastnil
- pro přístup/práci nutno zkonvertovat na shared_ptr pomocí lock (tím se "zablokuje" případné zrušení původního shared_ptr v této době). Použijeme unique_uk.lock(), který vrátí shared_ptr
- Dá se zjistit, zda originál stále existuje (weak_ptr != nullptr, nebo metoda expired)

```
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 je nullptr => bb již zaniklo
// jednodušeji
if (xx.expired())
  // bb je zrušeno
else
// bb je ještě validní
```