1. Temporális változók élettartama.

C++ programban sokszor használunk temporális változókat, pl. :

```
void f(X a1, X a2)
{
   extern void g( const X& );
   X z;
   // ...
   z=a1+a2;  // z = operator=( operator+(a1,a2) );
   g(a1+a2);
}
```

A fenti példában az értékadásnál szükségünk van egy temporális objektumra, amely tartalmazza az a1+a2 összeget. Hasonlóan szükségünk van a függvényhívásnál is egy objektumra, amely már tartalmazza a két változó összegét, hogy referencia szerint átadhassuk a g() függvénynek. Tételezzük fel, hogy az X egy destruktorral rendelkező osztály. A kulcskérdés az, hogy mikor fusson le a temporális objektum destruktora. Kézenfekvő válasz az lenne, hogy a blokk végén, mint minden más normális esetben is az objektumot tartalmazó blokk végén hívódik meg a hozzá tartozó destruktor függvény.

A megoldást két esetre kell tesztelni:

- Az első esetben a g() függvény visszaadhat egy pointert, amely a referenciaként átvett temporális változóra mutat, amelyre az f() függvényben még hivatkozhatunk. Ilyenkor azt szeretnénk, hogy a temporális változó hosszú élettartamú legyen.
- A második esetben képzeljük el, hogy 1000×1000 mátrixokkal képzünk műveleteket, és tucatnyi temporális mátrixunk keletkezik. Ilyenkor minél előbb szeretnénk megszabadulni a temporális objektumoktól, miután már nem hivatkozunk rájuk.

Egy lehetséges megoldást jelentett volna, ha a blokkokkal szabályozzuk a temporális objektumok élettartamát:

```
void f(X a1, X a2)
{
   extern void g( const X& );
   X z;
   // ...
  {  z=a1+a2; }
  {  g(a1+a2); }
}
```

Az első CFront implementáció a temporlális objektumok destruktorait a blokk végén hívta meg. Sok programozó azonban ennél jobb megoldást követelt. Ezért a következő Annotated C++ Reference Manualba (ARM) az került be, hogy a destruktor az objektum keletkezése után a blokk végéig bárhol meghívható, ami egy rossz döntés volt, mert különböző fordító implementációk között a forráskód hordozhatatlanná vált abban az esetben, ha a programozók más-más temporális objektum élettartalommal számoltak. Ha közvetlenül a temporális objektum felhasználása után hívjuk meg a destruktort, akkor a következő kód működésképtelen:

```
class String {
   // ...
public:
   friend String operatot+( const String& s1, const String& s2 );
   // ...
   operator const char*(); // c string
};
```

```
void f( String s1, String s2 )
{
   printf("%s\n", (const char*)(s1+s2) );
}
```

Hasonló programrészek miatt felmerülhet az az ötlet. hogy a temporális objektumokat az őket használó utasítás végen szüntessük meg. Persze erre az esetre is lehet ellenpéldát találni.

```
void g( String s1, String s2 )
{
   const char* p = s1+s2;
   printf("%s\n", p );
}
```

A C++ szabványosítási bizottság két évig szöszmötölt, míg rávették magukat, hogy lezárják a dolgot. Abban mindenki megegyezett, hogy tökéletes megoldás nem létezik.

A lehetséges megoldások a temporális objektumok megszüntetésére:

1. az első használat után rögtön
2. az utasítás végén
3. a következő elágazási pont után
4. a blokk végén
5. a függvény végén
6. az utolsó használat után
7. legyen nem definiált, mint amit ARM-ben is szabályként meghatároztak

Az utolsó használat után lévő temporális objektum megszüntetése jól hangzik, de a fordítóprogramnak ebben az esetben képesnek kell lenni a program vezérlésszerkezetének elemzésére (control flow analysis), amely implementációs problémákat vet fel. A szabványosítási bizottság sokáig az utasítás végén akarta a temporális objektumokat megszüntetni, de egy újabb probléma merült fel:

```
void h(String s1, String s2)
{
   const char* p;
   if( p = s1+s2 ) {
    // ...
   }
}
```

A fordítógyártók meggyőzték a szabványosítási bizottságot, hogy a feltétel rész után a temporális objektum szűnjön meg, és az elfogadott szabály ezután az lett, hogy a *teljes kifejezés* vége után kell a temporális objektum destruktorát meghívni.

```
void f( String s1, String s2 )
{
   const char* p;
   if( p = s1+s2 ) printf("%s\n", p);       // Hibás
   if( ( p=s1+s2 ) && p[0] ) { ... }        // OK

   printf("%s\n", (const char*)(s1+s2) );   // OK
   String s3 = s1+s2;
   printf("%s\n", (const char*)s3  );       // OK
   cout << s1+s2;                           // OK
}
```

Irodalom: Bjarne Stroustrup, The Design and Evalution of C++ 6.3.2 Lifetime of Temporalis