Inicjalizacja i niszczenie obiektów w wybranym języku programowania.

Patryk Lisik

Uniwersytet Łódzki

2023/24

Co to typ?

Typ

Dane i zbiór operacji, które można na nich wykonać [1].

```
class ChildFucntion : public LineFucntion {
    public:
ChildFucntion(double a, double b): LineFunction(a,b){
    puts("Child");}

ChildFucntion(ChildFucntion const & cf{
   puts("Child copy constuctor");
    }
~CholdFucntion(){puts("Child destructor");}
    }
```

Rysunek 1: Przykład własnego typu w C++

Co to obiekt?

Obiekt

Ukonkretnienie typu w postaci zaalokowanej pamięci.

Obiekt

```
class ChildFucntion : public LineFucntion {
    public:
ChildFucntion(double a, double b): LineFunction(a,b){
    puts("Child");}

ChildFucntion(ChildFucntion const & cf{
   puts("Child copy constuctor");
    }
~CholdFucntion(){puts("Child destructor");}
    }
```

Konstruktor


```
class ChildFucntion : public LineFucntion {
    public:
ChildFucntion(double a, double b): LineFunction(a,b){
    puts("Child");}

ChildFucntion(ChildFucntion const & cf{
   puts("Child copy constuctor");
    }
~CholdFucntion(){puts("Child destructor");}
    }
```

Rysunek 2: Jawnie zdefiniowany konstruktor w C++. Zgodnie z zasadą 0/3/5.

Type alokacji


```
class ChildFucntion : public LineFucntion {
    public:
ChildFucntion(double a, double b): LineFunction(a,b){
    puts("Child");}

ChildFucntion(ChildFucntion const & cf{
   puts("Child copy constuctor");
    }
~CholdFucntion(){puts("Child destructor");}
    }
```

Rysunek 3: Alokacja na stosie

```
class ChildFucntion : public LineFucntion {
    public:
ChildFucntion(double a, double b): LineFunction(a,b){
    puts("Child");}

ChildFucntion(ChildFucntion const & cf{
   puts("Child copy constuctor");
    }
~CholdFucntion(){puts("Child destructor");}
    }
```

Rysunek 4: Alokacja na stercie

ABI konstruktorów w C++

Interfejs binarny aplikacji w c++

Kompilatory C++ są zgodne z Itanium C++ ABI [2]. Które definiuje następujące konstruktory i ich nazwy:

```
class ChildFucntion : public LineFucntion {
    public:
ChildFucntion(double a, double b): LineFunction(a,b){
    puts("Child");}

ChildFucntion(ChildFucntion const & cf{
   puts("Child copy constuctor");
    }
~CholdFucntion(){puts("Child destructor");}
    }
```

Użycie konstruktorów z ABI

Konstruktor C1 jest używany gdy obiekt nie jest klasą pochodną [3]. Konstruktor C2 jest używany gdy tworzony jest obiekt jest klasą pochodną. Kompilator zawsze emituje oba konstruktory.

Konstruktor C3 nie jest nigdy generowany przez GCC [4].

W C++ konstruktory domyślnie nie są dziedziczone. CI1 i CI2 są używane gdy w standardzie C++11 je dziedziczymy.

```
class ChildFucntion : public LineFucntion {
    public:
ChildFucntion(double a, double b): LineFunction(a,b){
    puts("Child");}

ChildFucntion(ChildFucntion const & cf{
   puts("Child copy constuctor");
    }
~CholdFucntion(){puts("Child destructor");}
    }
```

Podwójne wywołanie destruktora?

ABI wymaga destruktora D1 i D2, które są tożsame w przypadku braku dziedziczenia.

```
class ChildFucntion : public LineFucntion {
    public:
ChildFucntion(double a, double b): LineFunction(a,b){
    puts("Child");}

ChildFucntion(ChildFucntion const & cf{
   puts("Child copy constuctor");
    }
~CholdFucntion(){puts("Child destructor");}
    }
```

Rysunek 5: Wynik kompilacji kodu z rys. 6 za pomocą GCC 13.3 z flagą -00

Alokacja i dealokacja na stercie


```
class ChildFucntion : public LineFucntion {
    public:
ChildFucntion(double a, double b): LineFunction(a,b){
    puts("Child");}

ChildFucntion(ChildFucntion const & cf{
   puts("Child copy constuctor");
    }
~CholdFucntion(){puts("Child destructor");}
    }
```

Rysunek 6: Wynik kompilacji kodu z rys. 6 za pomocą GCC 13.3 z flagą -00

Alokacja pamięci

Alokacja pamięci

Alokacja pamięci to rezerwacja przez system operacyjny pamięci dla procesu[5, 6].
(Nie rozpatrujmy sytuacji gdy nie ma systemu operacyjnego)

POSIX

W systemach zgodnych z POSIX alokacja pamięci odbywa się poprzez ‘mmap()‘ [5].

Windows

W systemach Windows alokacja pamięci odbywa się poprzez VirtualAlloc[memoryapi.h] [6].

Kolejność wywoływania konstruktorów


```
class ChildFucntion : public LineFucntion {
    public:
ChildFucntion(double a, double b): LineFunction(a,b){
    puts("Child");}

ChildFucntion(ChildFucntion const & cf{
   puts("Child copy constuctor");
    }
~CholdFucntion(){puts("Child destructor");}
    }
```

Najpierw wywoływany jest konstruktor rodzica.

Unique_ptr

Własność w programowaniu

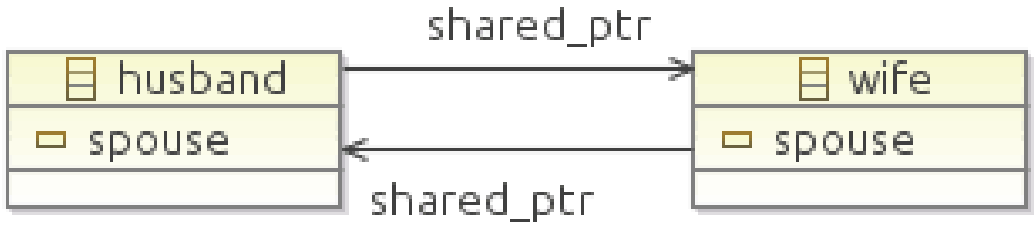
Własność określa kto jest odpowiedzialny za zwolnienie pamięci [7, 8].

std::move informuje kompilator o możliwości przeniesienie zasobów już istniejącego obiektu.

Shared_ptr

std:shared_ptr składa się z licznika referencji i właściwego wskaźnika. Podczas kopi licznik referencji jest zwiększany, a podczas wywołania destruktora zmniejszany. Przy wywołaniu ostatniego destruktora pamięć jest zwalniana [9].

Problem cyklicznych referencji



Rysunek 7: Problem cyklicznych referencji gdzie pamięć nigdy nie jest zwalniana. Źródło: [10]

std::weak_ptr

Rozwiązaniem problemu cyklicznych referencji jest referencja słaba(std::weak_ptr) która nie powoduje inkrementacji licznika referencji. Nie gwarantuje one że obiekt nie został już usunięty [10, 11].

Oznacz i zmieć (ang. *mark and sweep*)

Faza mark

Przeszukujemy "w głąb" graf obiektów i oznaczamy każdy obiekt do którego dotarliśmy [12].

Faza sweep

Usuwamy obiekty nieoznaczone.

Faza compact(Opcjonalna)

Obiekty przesuwane są blisko siebie w pamięci.

Krytyka

Jest to bardzo wolny algorytm, który wymaga zatrzymania przetwarzania do działania (ang. *stop the world GC*).

Bez fazy compact powoduje dużą fragmentację pamięci.

Inne rodzaje odśmiecaczy/garbage collectorów

- Generacyjny
- Trzy kolorowy
- Arenowy

Literatura I

 "Ibm.com – type definitions." https://www.ibm.com/docs/en/epfz/5.3?topic=reference-type-definitions, (dostęp 20-10-2023).

 "Itanium c++ abi." https://itanium-cxx-abi.github.io/cxx-abi/abi.html, (dostęp 31-10-2023).

 "Two identical constructors emitted? that's not a bug!." https://community.ibm.com/community/user/power/blogs/archive-user/2015/08/27/two-identical-constructors-emitted-thats-not-a (dostęp 31-10-2023).

Literatura II

 "Kod źródłowy gcc. generowanie konstrktorów." https://gcc.gnu.org/git/?p=gcc.git;a=blob;f=gcc/cp/mangle.c;h=10c2e2beb0c422e4f56e17e7659fbeb4ab3ee31b;hb=refs\/tags/gcc-4_8_1-release#l1644, (dostęp 31-10-2023).

 "mmap(2) — linux manual page." https://man7.org/linux/man-pages/man2/mmap.2.html, (dostęp 31-10-2023).

 "Virtualalloc function (memoryapi.h)." https://learn.microsoft.com/en-gb/windows/win32/api/memoryapi/nf-memoryapi-virtualalloc?redirectedfrom=MSDN, (dostęp 31-10-2023).

Literatura III

 "The rust programming language sekcja [what is ownership?]." https://doc.rust-lang.org/book/ch04-01-what-is-ownership.html#what-is-ownership, (dostęp 31-10-2023).

 "Cpp reference - dynamic memory managemnt - std::unique_ptr." https://en.cppreference.com/w/cpp/memory/unique_ptr, (dostęp 31-10-2023).

 "Cpp reference - std::move." https://en.cppreference.com/w/cpp/utility/move, (dostęp 31-10-2023).

Literatura IV


"Weak pointers and circular references in c++ 11." https://visualstudiomagazine.com/articles/2012/10/19/circular-references.aspx, (dostęp 31-10-2023).


"Cpp reference - dynamic memory management - std::weak_ptr." https://en.cppreference.com/w/cpp/memory/weak_ptr, (dostęp 31-10-2023).


"Cs330 - mark and sweep." https://www.cs.odu.edu/~zeil/cs330/f13/Public/garbageCollection/garbageCollection-htmlsu5.html, (dostęp 31-10-2023).