

RI301

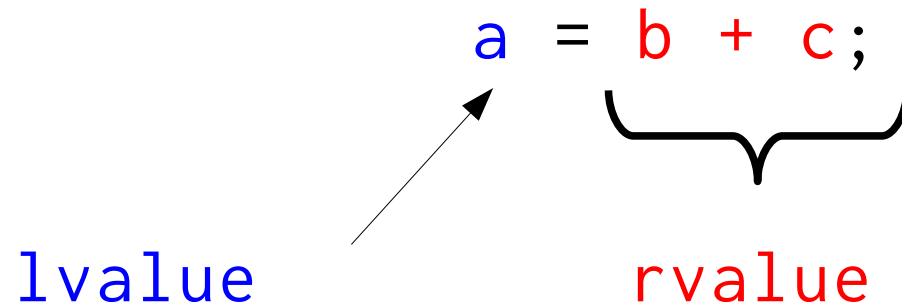
Strukture podataka

dr.sc. Edin Pjanić

Pregled predavanja

- lvalue i rvalue reference
- Move semantika
- Forward (“univerzalna”) referenca (T&&)
- Savršeno prosljeđivanje

Lvalue i rvalue



```
int a = 42;  
int b = 43;
```

```
a = b;  
b = a;  
a = a + b;
```

```
int c = a * b;  
a * b = 42;  
a * (b = 42);
```

Šta je od ovih iskaza ispravno a šta nije?

Zašto?

Lvalue i rvalue

- Šta je lvalue?
- Bilo šta od narednog može poslužiti za odgovor:
 - Sve ono što može stajati sa lijeve strane operatora = je lvalue.
 - Ako neki objekat ima ime to je lvalue.
 - Ako se može dobiti adresa izraza onda je taj izraz lvalue.
 - Ako je tip izraza lvalue referenca, npr. T&, const T& i sl. onda je taj izraz lvalue.
- Šta je rvalue?
 - Sve ono što nije lvalue

Tipičan primjer za rvalue je privremeni objekat koji vraća funkcija ili međurezultat nekog izraza, literal i sl., kao naprimjer:

`neka_funkcija()`, `a+b`, `56`, `32-6`, ...

Lvalue i rvalue reference

- Lvalue referenca je referenca na lvalue. (Tip &)
- Lvalue referenca je drugo ime za objekat koji je iza lvalue izraza i može se koristiti za pristup članovima tog objekta.
- Rvalue referenca je **referenca** na rvalue. (Tip &&)
- Rvalue referenca se tipično koristi radi implementacije move semantike.
 - U tom kontekstu, nastoji se iskoristiti objekat koji bi se ionako uništio.

```
void f(Tip &);  ← Overload za lvalue ref. kao argument
void f(Tip &&); ← Overload za rvalue ref. kao argument
Tip g();
```

```
Tip x;
f(x);      ← Poziva se f(Tip &)
f(g());    ← Poziva se f(Tip &&)
x = g();   ← Šta se ovdje dešava?
```

Lvalue i rvalue reference

- Šta možemo prosljeđivati u funkcije, u zavisnosti od toga kako je deklarisan parametar?

`void f(Tip &);` ← Može primiti samo lvalue

`void f(const Tip &);` ← Može primiti lvalue i rvalue, ali u funkciji ne možemo razlikovati da li je lvalue ili rvalue.

`void f(Tip &&);` ← Može primiti samo rvalue kao argument

`void f(Tip);` ← Prenos parametra po vrijednosti (pravi se kopija prosljeđenog parametra).

std::move()

- Najprije da raščistimo, ova funkcija ne premješta ništa.
- Ova funkcija je samo jedna vrsta cast operatora koji vraća rvalue referencu proslijeđenog joj objekta.
- Koristi se da bi se forsirao poziv odgovarajuće funkcije koja ima više overload varijanti, kako bi se implementirala move semantika (najčešće to na kraju rezultira pozivom move konstruktora ili move operatora =).

```
void f(Tip &);  
void f(Tip &&);
```

```
Tip x;
```

```
f(x);
```

Poziva se varijanta f(Tip &).

```
f(std::move(x));
```

Poziva se varijanta f(Tip &&).

Primjer 1

```
class Tip{
public:
    Tip();
    Tip(const Tip&);
    Tip(Tip&&);
    ...
};

void h(Tip && x){
    Tip a(x);
    ...
}

int main(){
    ...
    h( Tip() );
    ...
}
```

Koja je razlika između:

```
new Tip()
i
Tip()
```

Koji konstruktor će biti pozvan i zašto?

Sjetimo se, sve što ima ime i za šta možemo dobiti adresu je lvalue.

U ovom slučaju x je lvalue unutar funkcije h(), mada je objekat proslijeđen u h() kao rvalue.

Šta da uradimo ako u ovom i ovakvim slučajevima želimo da se pozove move konstruktor?

Rješenje ovog “problema”:

```
void h(Tip && x){
    Tip a(std::move(x));
    ...
}
```


Primjer 2

Šta ako želimo da funkcija `h()` radi isto, bilo da se proslijedi lvalue ili rvalue, s razlikom što treba da pozove odgovarajući konstruktor, zavisno da li je proslijeđen lvalue ili rvalue?

```
class Tip{  
    public:  
        Tip(const Tip&);  
        Tip(Tip&&);  
};
```

```
void h(const Tip & x){  
    Tip a(x);  
    ...  
}
```

```
void h(Tip && x){  
    Tip a(std::move(x));  
    ...  
}
```

Moramo imati dvije varijante funkcije koje imaju istu implementaciju svega osim konstrukcije obje

Postoji li neko elegantnije rješenje, bez dupliranja koda, gdje bi se parametar `x` proslijedio sa nepromijenjenom prirodom, tj. ako je došao kao lvalue da se takav i proslijedi u odgovarajući konstruktor, a ako je došao kao rvalue, da se opet pozove konstruktor koji implementira move semantiku?

Forward referenca /

“Univerzalna referenca” (Scott Meyers)

- Prisjetimo se:
 - Ako želimo deklarirati rvalue referencu, korist ćemo npr. `Tip &&`
- Međutim:
 - Ako u kodu imamo `Tip &&`, to ne znači uvijek da je to rvalue referenca.

`Tip &&` može značiti:

- Rvalue referencu
 - Veže za sebe rvalue, implementira move semantiku.
- Forward referencu, ili tzv. “univerzalnu referencu”
 - Može označavati rvalue ref. ili lvalue ref.
 - Veže lvalue i rvalue.
 - Može implementirati copy, a može i move.

Forward referenca /

“Univerzalna referenca” (Scott Meyers)

- Kad Tip && postaje forward referenca?
- Ako varijabla ili parametar funkcije ima tip deklarisan kao

T &&

za neki zaključeni tip T, to je forward (univerzalna) referenca.

- Forward referenca može biti u kontekstima:

- Parametar generičke funkcije:

```
template<typename T>  
void f(T && x)
```

- auto deklaracija:

```
auto && x = ... ;
```

Savršeno prosljeđivanje parametra - `std::forward<T>`

- Inicijalizacija forward reference:
 - Ako je izraz kojim se forward referenca inicijalizuje **lvalue** onda forward ref. postaje lvalue referenca.
 - Ako je izraz kojim se forward ref. inicijalizuje **rvalue** onda forward ref. postaje rvalue referenca.
- Čemu nam to služi?

```
class Tip{  
    public:  
        Tip(const Tip&);  
        Tip(Tip&&);  
};  
  
template<typename T>  
void h(T&& x){  
    T a(std::forward<T>(x));  
    ...  
}
```

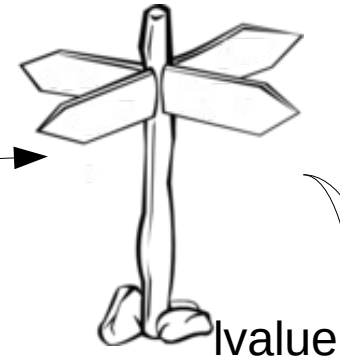
Pri pozivu funkcije `h ()` kompajler zaključuje tip prosljeđenog parametra i generiše funkciju sa odgovarajućim tipom, tj. `T&&` postaje `const T &`, `T&` ili `T&&`, zavisno od toga šta je prosljeđeno.

`forward` vrši odgovarajući cast parametra u njegovu pravu prirodu, tj. onako kako je poslan u funkciju `h ()`. Ako je `x` poslan kao lvalue, `forward` daje lvalue referencu, a ako je `x` prosljeđen kao rvalue, `forward` daje rvalue referencu => move semantika.

Rezime: prenos parametra po referenci

```
void h(const Tip & x){  
    ...  
}
```

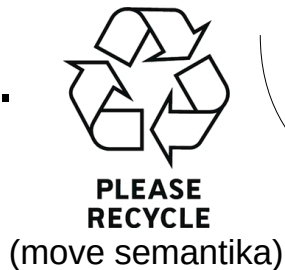
u funkciji:
X ne možemo
mijenjati



```
void h(Tip & x){  
    ...  
}
```

nakon izvršenja funkcije:
X nastavlja da živi

```
void h(Tip && x)  
{  
    ...  
}
```



u funkciji:
X možemo
slobodno
mijenjati

nakon izvršenja funkcije:
X nestaje (tj. poziva se
njegov destruktork), u pravilu


rvalue

Potrebna pažnja

- Treba obratiti pažnju kod parametara metoda generičkih klasa.


- Razmotrimo primjer:

- U primjeru desno tip `T&&` u metodu `metod(T&& x)` nije zaključeni tip u deklaraciji metoda jer je taj tip već poznat pri kreiranju instance ove klase.



```
template<typename T>
class Klasa{
    . . .
    void metod(T&& x);
    . . .
};
```

- Da bi ovaj parametar bio forward referenca, moramo metod definisati kao generički. Tek tada `U&&` postaje forward (univerzalna) referenca:



```
template<typename T>
class Klasa{
    . . .
    template<typename U>
    void metod(U&& x);
    . . .
};
```

Pravila slaganja referenci

- Univerzalna referenca je izraz koji je smislio Scott Meyers, jedan od vodećih C++ eksperata, radi lakšeg pamćenja i shvatanja pravila slaganja referenci.
- Primjer: neka imamo neki objekat koji je deklarisan kao TR, a što je referenca na neki tip T, pri čemu je tip T takođe referenca.

```
typedef int& T;  
typedef T& TR;  
TR var; // šta je sada var?
```

- Pravila:
 - Tip& & postaje Tip&
 - Tip& && postaje Tip&
 - Tip&& & postaje Tip&
 - Tip&& && postaje Tip&&

Za vježbu

- Implementirati copy i move konstruktore za strukture koje smo do sada radili.
- Implementirati copy i move operatore dodjeljivanja za strukture koje smo do sada radili.