Preklapanje funkcija

- Ponekad postoji potreba da se naprave potprogrami koje rade logički istu stvar, samo sa drugačijim brojem ili tipovima parametara. U tradicionalnim jezicima, za ovakve potprograme morala bi da se osmisle različita imena, jer prevodilac poziv potprograma vezuje sa pozvanim potprogramom samo na osnovu imena
- Na jeziku C++, različite funkcije mogu imati isto ime, ukoliko se dovoljno razlikuju po broju ili tipovima parametara; ovo se naziva *preklapanje* (ili *preopterećenje*) funkcija (*function overloading*)
- Prevodilac vezuje poziv funkcije za pozvanu funkciju ne samo na osnovu imena, nego i u zavisnosti od broja i tipova stvarnih argumenata koje uparuje sa tipovima formalnih parametara, pri čemu se pretražuju oblasti važenja u zavisnosti od toga kojim oblastima važenja pripadaju argumenti (tzv. *argument-dependent lookup*, ADL i *overload resolution*). Na primer:

```
double max (double, double);
const char* max (const char*, const char*);
…
const char* s = max("March", "January");
double d = max(3.6,5);
```

Ove funkcije vraćaju "veći" od dva parametra, šta go da su

Poziva se *max(const char*, const char*)*

Poziva se *max(double, double)*

Preklapanje funkcija

- U poređenje tipova ulaze i cv-kvalifikacije, što znači da se razlikuju tipovi koji jesu ili nisu kvalifikovani kao konstantni, uključujući i objekte i njihove funkcije članice koje jesu ili nisu konstantne. Na primer:

```
Task& TaskQueue::at (int position = 0);

inline const Task& TaskQueue::at (int position = 0) const;
  return const_cast<TaskQueue* const>(this)->at(position);
}
```

Sada se na sledećim mestima pozivaju odgovarajuće funkcije, u zavisnosti od konstantnosti objekta za koji se one pozivaju:

```
const TaskQueue* pcq = ...;
TaskQueue* pq = ...;

const Task& ct1 = pcq->at();

const Task& ct2 = pq->at();

Task& ct3 = pq->at();

Task& ct4 = pcq->at();
```