Шаблони са променљивим бројем параметара

- Шаблони са променљивим бројем параметара омогућавају да се направи један шаблон који покрива случајеве са произвољно много параметара различитог типа.

... означава да је у питању произвољан број параметара и то се назива „пакет параметара шаблона“. Args је само назив за касније идентификовање (и може бити било шта).

```
template<typename... Args>
void foo(Args... args) {
    bar(args...);
}
```

Ова употреба је врло специфична и означава „распакивање“ пакета параметара шаблона у листу параметара функције, која се зове „пакет параметара функције“. args је такође произвољан назив.

Ово је тзв. „распакивање“ параметара функције. Означава да на том месту треба да се нађе листа свих функцијских параметара одвојених зарезима.

Шаблони са променљивим бројем параметара

```
template<typename... Args>
void foo(Args... args) {
    bar(args...);
}
```

- Илустрација употребе оваквог шаблона

```
foo(5, 0.5, "hik");
// је исто што и ово
foo<int, double, const char*>(5, 0.5, "hik");
// што чини да компајлер генерише овакву функцију:
void foo(int p1, double p2, const char* p3) {
    bar(p1, p2, p3);
}
```

- Наравно, p1, p2 и p3 су измишљена имена.

- Подсећање, ово је јако користан сајт: cppinsights.io

Шаблони са променљивим бројем параметара

```
template<typename... Args>
void foo(Args... args) {
    bar(args...);
}
```

- Илустрација употребе оваквог шаблона

```
foo(5, 0.5, "hik");
// је исто што и ово
foo<int, double, const char*>(5, 0.5, "hik");
// што чини да компајлер генерише овакву функцију:
void foo(int p1, double p2, const char* p3) {
    bar(p1, p2, p3);
}
```

- Наравно, p1, p2 и p3 су измишљена имена.

- Подсећање, ово је јако користан сајт: cppinsights.io

Шаблони са променљивим бројем параметара

- Лако се може одабрати како се преносе параметри.

```
template<typename... Args>
void foo(Args&... args) {
    bar(args...);
}

foo(5, 0.5, "hik");

void foo(int & p1, double & p2, const char* & p3) {
    bar(p1, p2, p3);
}
```

- Наравно, може и овако:

```
template<typename... Args> void foo(const Args&... args) //...
```

- На овај механизам се ослањају функције std::make_shared и std::make_unique.

Шаблони са променљивим бројем параметара

- Важно је имати у виду да се параметри распакују по обрасцу који је задат.
- Овај механизам се назива „fold expression“.

```
template<typename... Args>
void foo(Args... args) {
    bar(2*args...);
}

foo(5, 0.5, 4);

void foo(int p1, double p2, int p3) {
    bar(2*p1, 2*p2, 2*p3);
}
```

```
template<typename... Args>
void foo(Args... args) {
    bar(baz(args)...);
}

foo(5, 4);

void foo(int p1, int p2) {
    bar(baz(p1), baz(p2));
}
```

Шаблони са променљивим бројем параметара

- Ово је облик шаблона где је једини параметар заправо „пакет“ параметара.

```
template<typename... Args>
void foo(Args... args) //...
```

- Мада врло често имамо и овако нешто:

```
template<typename T, typename... Args>
void foo(T x, Args... args) {
  // сада имамо x као први параметар и args као остали параметри
  // сада можемо рекурзивно ићи кроз листу параметара
  // тј. “љуштимо” листу
}
```

Шаблони са променљивим бројем параметара

- Једноставни пример суме:
- Желимо да нам функција враћа суму свих параметара, нпр.:

```
int a = sum(1, 2, 3); // a треба да буде 6
int b = sum(6, 7, 5, 2); // b треба да буде 20
```

- Имплементација би могла овако да изгледа:

```
template<typename T, typename... Args>
T sum(T x, Args... args) {
    return x + sum(args...);
}

sum(3, 4, 5);
int sum(int x, int p1, int p2) { return x + sum(p1, p2); }
int sum(int x, int p1) { return x + sum(p1); }
int sum(int x) { return x + sum(); }
int sum() ??? // по шаблону мора бити бар један параметар
```

- ...али проблем је што sum() функција није дефинисана.

Шаблони са променљивим бројем параметара

- Једно решење може бити да дефинишемо ту недостајућу функцију sum(), али ту би наишли на неколико (мањих) проблема.

```
int sum() {
    return 0;
}
template<typename T, typename... Args>
T sum(T x, Args... args) {
    return x + sum(args...);
}

sum(3, 4, 5);
```

```
int sum(int x, int p1, int p2) { return x + sum(p1, p2); }
int sum(int x, int p1) { return x + sum(p1); }
int sum(int x) { return x + sum(); }
```

Шаблони са променљивим бројем параметара

- Једно решење може бити да дефинишемо ту недостајућу функцију sum(), али ту би наишли на неколико (мањих) проблема.
- Боље решење је да урадимо специјализацију шаблона за један параметар:

```
template<typename T>
T sum(T x) {
    return x;
}
template<typename T, typename... Args>
T sum(T x, Args... args) {
    return x + sum(args...);
}

sum(3, 4, 5);
```

```
int sum(int x, int p1, int p2) { return x + sum(p1, p2); }
int sum(int x, int p1) { return x + sum(p1); }
int sum(int x) { return x; }
```

- Сад је све ОК.

Шаблони са променљивим бројем параметара

- Корисно је знати и за **sizeof...** операцију.
- Та операција се примењује на пакет параметара и враћа број параметара.
- Илустровано на функцији која враћа средњу вредност својих параметара.

```
template<typename... Args>
auto average(Args... args) {
    return sum(args...) / sizeof...(args);
}
```

- Подсећање: шта ће бити тип повратне вредности функције **average**?
- Заправо, и тип повратне вредности sum би требало тако да дефинишемо, ако желимо да ради исправно и са разноврсним типовима.

```
template<typename T> T sum(T x) { return x; }

template<typename T, typename... Args>
auto sum(T x, Args... args) { return x + sum(args...); }

// сад ће радити исправно и за овакав позив:
double x = sum(5, 0.5, 7);
```

- sum на најлакши начин :)

```
template<typename... Args>
auto sum(Args... args) {
    return (args + ...);
}
```

- Заграде око **args + ...** су важне!