When should we use emplace instead of push in STL containers?

1. emplace_back

Inserts a new element at the end of the vector, right after its current last element. This new element is constructed in place using args as the arguments for its constructor.

2. push_back

Adds a new element at the end of the vector, after its current last element. The content of val is copied (or moved) to the new element.

3. An example

```
class silkMadCow
{
public:
    string name;
    int strong;
    silkMadCow(string _str, int _s)
    {
        name = _str;
        strong = _s;
    }
};
```

```
vector<silkMadCow> OS;
silkMadCow zzz("Zhou ZZ", INT32_MAX);
OS.push_back(zzz);
OS.emplace_back("WJY", 0);
for (auto &i : OS)
{
  cout << i.name << " is Lv.";
  cout << i.strong << " strong";
  cout << endl;
}
```

The output is

```
Zhou ZZ is Lv.2147483647 strong
WJY is Lv.0 strong
```

4. Conclusion

- **"emplace_back"** is used when implicit conversions are involved, or the element type is expsensive to construct, in such condition it's faster than **"push_back"**.
- **push_back** is preferred in normal condition, since the code will tend to be more readable, and will be safer (you can spot problems before real running)

What is parameter pack in C++11?

- A template parameter pack is a template parameter that accepts zero or more template arguments (non-types, types, or templates).

```
template<class ... T>
struct Sombra {};

Sombra<int>;
Sombra<int,char>;
```

- A function parameter pack is a function parameter that accepts zero or more function arguments.

```
void admire()
{
    return;
}
template <class T, class... args>
void admire(T head, args... rest)
{
    cout << "I really admire " << head << "." << endl;
    admire(rest...);
}
```

```
admire("Zhou ZZ");
admire("FS", "XD", 114514);
```

Ouput:

```
I really admire Zhou ZZ.
I really admire FS.
I really admire XD.
I really admire 114514.
```

How to use std::forward to apply this trick?

```
class horseCow
{
public:
    vector<string> _v;

    template <class... Args>
    horseCow(Args &&... args) : _v(std::forward<Args>(args)...) {}
};
```

```
const char *project[3] = {"Mumsh", "Lemondb", "WoBuZhiDao"};
horseCow ve477(5, "Turing Machine");
horseCow ve482(project, project + 3);
for (auto &i : ve477._v)
{
  cout << i << endl;
}
for (auto &i : ve482._v)
{
  cout << i << endl;
}
```

Output:

```
Turing Machine
Turing Machine
Turing Machine
Turing Machine
Turing Machine
Mumsh
Lemondb
WoBuZhiDao
```