Жив ли Си++?
Прошлое, настоящее, будущее.

Карпов Андрей Николаевич
к.ф.-м.н., MVP (Visual C++),
технический директор
ООО «СиПроВер»
Сайт: www.viva64.com
E-Mail: karpov@viva64.com

Умер ли Си/Си++?
Нет. Он одна из важных технологий.

Просто технологий стало больше!

- Кода на Си/Си++ в старых и новых системах не стало меньше.
- Выросло количество разрабатываемых систем, где рационально использовать другие языки (web, мобильные телефоны, клиентские программы).
- **Аналогия с бумагой**. Кажется, компьютерные технологии повсеместно могут заменить бумагу.

Говорить «стало меньше Си++» как говорить «меньше стала использоваться бумага»

- Государственные учреждения. Куда ставить печать и подпись?
- Надежность (архивы, военное дело).
- Бумажные книги не исчезли.
- Комиксы, раскраски для детей, оригами.
- Как поможет iPhone в туалете?

Закрывать бумажную фабрику смысла нет.

А стоит ли продолжать заниматься Си/Си++?

Да. Это приносит деньги.

Поговорим, почему это приносит деньги.

Почему язык Си/Си++ ещё очень долго будет популярен

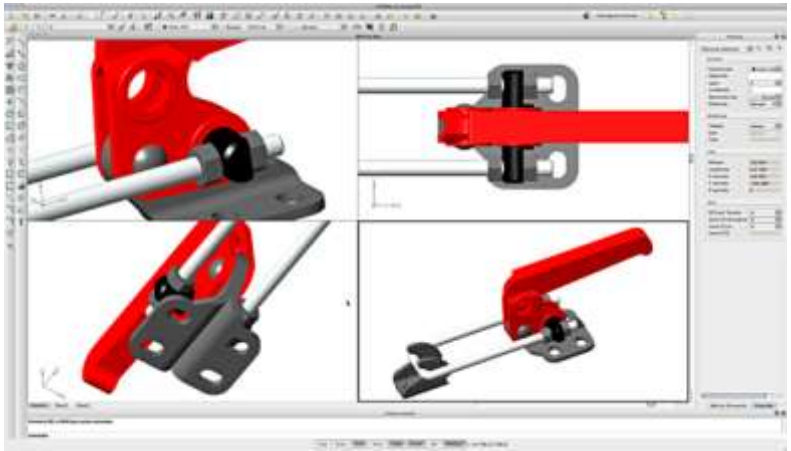
- Инерция;
- Ресурсоемкие вычисления;
- Экономия энергии;
- Ограничения встраиваемых и мобильных систем;
- Новые языки приходят и уходят, а Си/Си++ остаётся.

Инерция

- Существуют такие продукты, как «CHARON-PDP11 for Windows» для поддержки PDP11. Компания Stromasys - www.stromasys.ch

- Неудача Intel с 64-битной архитектурой Itanium.

- Огромное количество БОЛЬШИХ библиотек. Пример: библиотека ACIS.



Ресурсоемкие вычисления

- Активно используется Фортран. Си/Си++ рассматриваются для таких задач в перспективе. Intel активно работает над включением матриц в стандарт языка Си++.
- Си/Си++ позволяют максимально полно использовать доступную память.
- Си самый быстрый (после Фортрана). Не верьте рекламным измерениям скорости.

Экономия энергии

- Макро. Для датацентра с тысячами обращений в секунду важно, работает алгоритм 100 или 150 микросекунд.

- Микро. Важно для встраиваемых систем, работающих долгое время.

Ограничения встраиваемых и мобильных систем

- Нужно работать быстро, чтобы уменьшить потребление энергии.
- Ограничения объема памяти.
- Новое направление – робототехника на дому.
- Лучше потратить деньги на оптимизацию ПО, чем увеличить стоимость аппаратной части миллионов устройств.

Новые языки приходят и уходят, а Си/Си++ остаётся

C++ Applications: http://www.stroustrup.com/applications.html

- Adobe Systems: Photoshop, Acrobat, Illustrator, ...
- Apple: OS X (разные языки, но в основном Си++)
- Autodesk: различные CAD системы
- CERN: Data analysis
- Facebook: Several high-performance and high-reliability components.
- Google: Chromium, MapReduce, Google file system.
- Microsoft: Windows, Microsoft Office, ...
- Mozilla: Firefox, Thunderbird
- ...

Си/Си++ не только жив, но и постоянно развивается

О новом в языке
(будем говорить только про Си++)

Делегирующие конструкторы

```
class File
{
public:
  File(char const * filename, char const * mode)
      : file_(fopen(filename, mode))
  {
  }

  ~File()
  {
    fclose(file_);
  }
```

Теперь нам хочется сделать что-то сложное в конструкторе, что может бросить **исключение**.

Делегирующие конструкторы

```
class File
{
  File(FILE * file) : file_(file)
  {}

public:
  File(char const * filename, char const * mode)
      : File(fopen(filename, mode))
  {
    Foo();
  }
```

Другое применение –

можно избавиться от функции Init().

Диапазонный for, auto, списки инициализации

```
auto x = A * B;
```

```
std::vector<string> string_array = { "aa", "bb" };

for (auto s : string_array)
{
  cout << s << endl;
}
```

```
for (const auto x : { 1,2,3,5,8,13,21,34 })
  cout << x << '\n';
```

Управление поведением по умолчанию: default и delete

Сейчас стандартная идиома «запрета копирования» может быть явно выражена следующим образом:

```
class X {
  …
  X& operator=(const X&) = delete;
  X(const X&) = delete;
```

И наоборот, мы можем явно сказать о том, что хотим использовать поведение копирования по умолчанию:

```
class Y {
  …
  Y& operator=(const Y&) = default;
  Y(const Y&) = default;
```

enum class – строго типизированные перечисления

Объявления “enum class Color { red, blue };” решают проблемы:

- Стандартные перечисления (enums) могут неявно преобразовываться к int.
- Стандартные перечисления экспортируют свои значения в окружающую область видимости, что приводит к коллизиям имен.
- Невозможно указать тип, лежащий в основе стандартных перечислений.

constexpr – обобщенные гарантировано константные выражения

```
constexpr int Foo(int a, int b)
{
  constexpr int tmp = a + 10;
  return tmp | b;
}


int x = Foo(1, 2);
```

decltype – тип выражения

```
void f(const vector<int>& a, vector<float>& b)
{
      typedef decltype(a[0]*b[0]) Tmp;
      for (int i=0; i<b.size(); ++i) {
            Tmp* p = new Tmp(a[i]*b[i]);
            // ...
      }
      // ...
}
```

Инициализация членов класса при объявлении

```
class A {
public:
	int a = 7;
};
```

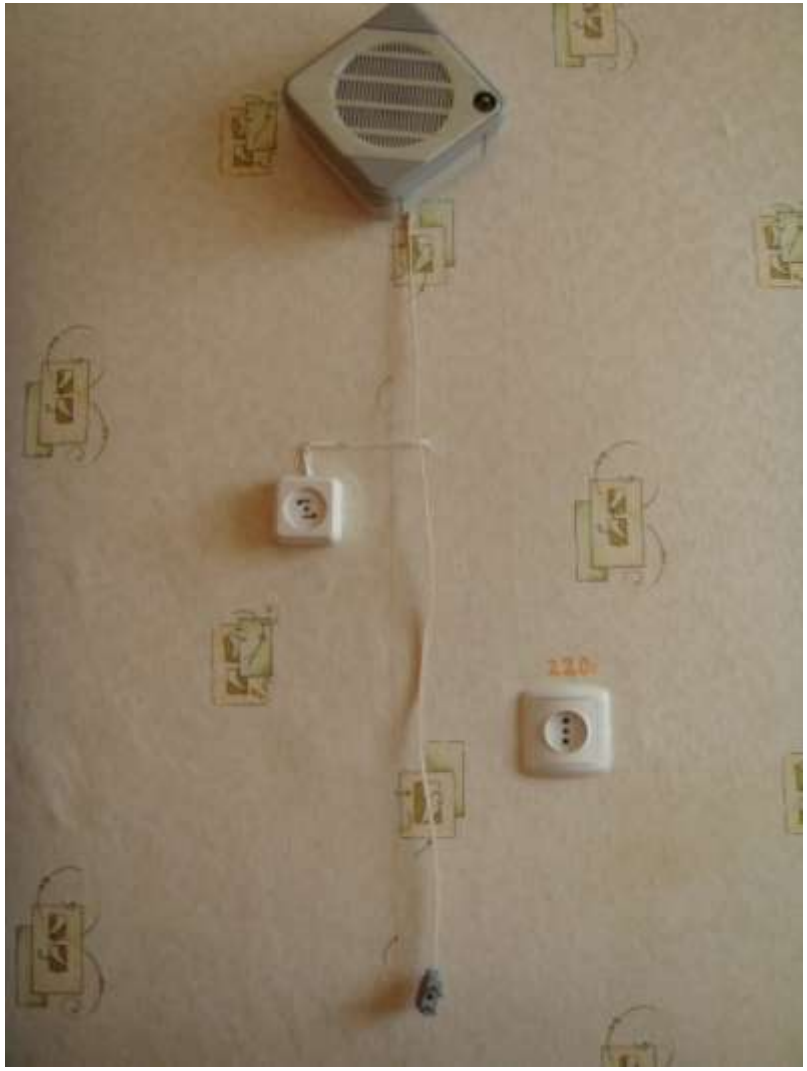
Эквивалентно:

```
class A {
public:
	int a;
	A() : a(7) {}
};
```

Статические утверждения

```
static_assert(
  sizeof(long)>=8,
  "64-bit code generation required for this library."
);
```

nullptr - литерал для задания нулевого указателя



```
void f(int);
void f(char*);

f(0);       // вызов f(int)

f(nullptr); // вызов f(char*)
```

Шаблоны с переменным числом параметров

```
template<typename T, typename... Args>
void printf(const char* s, T value, Args... args)
{
  while (s && *s) {
    if (*s=='%' && *++s!='%') {
      std::cout << value;
      return printf(++s, args...);
    }
    std::cout << *s++;
  }
}
```

Лямбда-выражения

```
std::sort(v.begin(), v.end(),
          [](int a, int b) { return abs(a)<abs(b); });
```

```
vector<int> myList;
int total = 0;
for_each(myList.begin(), myList.end(), [&total](int x)
{
  total += x;
});
std::cout << total;
```

```
auto myFunc = [this]() { this->PrivateMemberFunc (); };
```

Другое

- Rvalue ссылки &&
- Пользовательские литералы
- Атрибуты [[…]]
- Класс array: array<int,6> a = { 1, 2, 3 };
- …

Дополнительные ссылки

- C++ 11 FAQ от Бьярна Страуструпа (перевод) http://sergeyteplyakov.blogspot.com/2012/05/c-11-faq.html

- «Универсальные» ссылки в C++11 или T&& не всегда означает «Rvalue Reference» http://habrahabr.ru/post/157961/