
LAST
BUT
NOT
LEAST

HIGH LEVEL AGENDA

1. Engineering overview
2. Real world problems

Engineering overview

Learn from success stories

Engineering overview

Learn from success stories

- Project history (C,C++, Linux)

Engineering overview

Learn from success stories

- Project history (C,C++, Linux)
- Browse open source code

Engineering overview

Learn from success stories

- Project history (C,C++, Linux)
- Browse open source code
- Follow development news of commercial products

Engineering overview

Get the best from university

Engineering overview

Get the best from university

➜ Researching alone

Engineering overview

Get the best from university

- ➜ Researching alone
- ➜ Search for areas where the theory could apply

Engineering overview

Get the best from university

- ➜ Researching alone
- ➜ Search for areas where the theory could apply
- ➜ Dive deep for every problem

Engineering overview

Cook vs Chef [1]

Chef

- Invent recipies

Cook

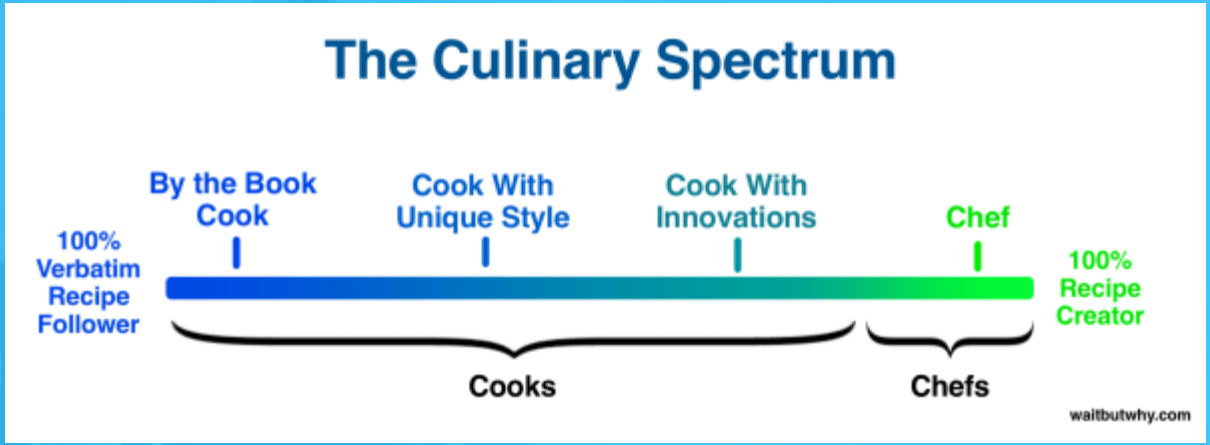
- Follow recipies

Engineering overview

Cook vs Chef [1]

Chef
- Invent recipies

Cook
- Follow recipies



1. waitbutwhy.com/2015/11/the-cook-and-the-chef-musks-secret-sauce.html

“C++ has indeed become too expert friendly”

Bjarne Stroustrup

Real world problems

Who uses C/C++?

- Web servers (apache, nginx)

Who uses C/C++?

- Web servers (apache, nginx)
- Databases (mysql)

Who uses C/C++?

- Web servers (apache, nginx)
- Databases (mysql)
- Operating systems (Linux)

Who uses C/C++?

- Web servers (apache, nginx)
- Databases (mysql)
- Operating systems (Linux)
- Antivirus software

Trivia - 01

What is the difference between

```
char*a = "abc";
char b[]="abc";
```

Packing C/C++ code

1.Executables

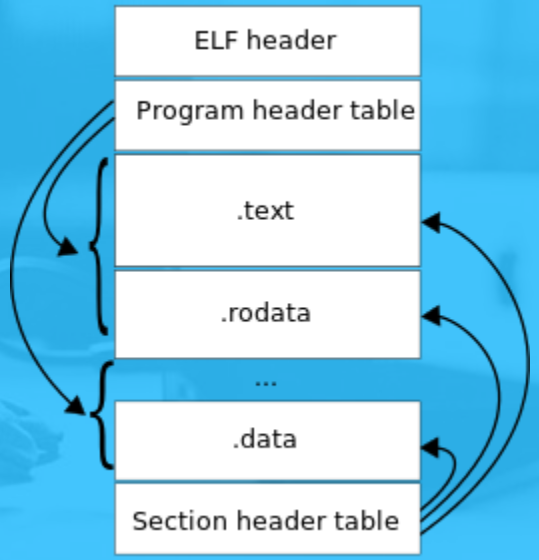


Image source: https://en.wikipedia.org/wiki/Executable_and_Linkable_Format

Packing C/C++ code

2.Static libraries (.lib,.a)

➜ Share code at compile time

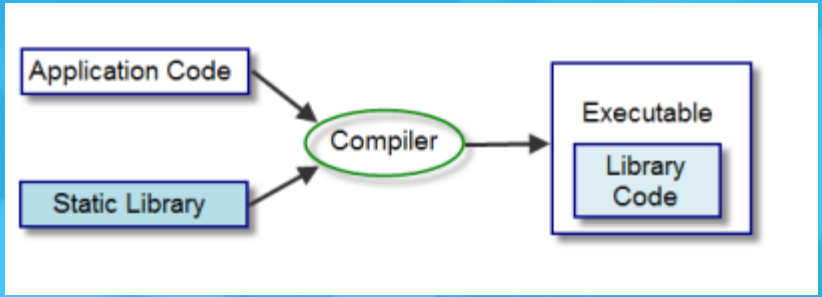


Image source: http://www.bogotobogo.com/cplusplus/images/libraries/StaticLibraryLink.png

Packing C/C++ code

3.Dynamic libraries (.dll/.so/.dylib)

- ➜ Share code at runtime
- ➜ Space efficient

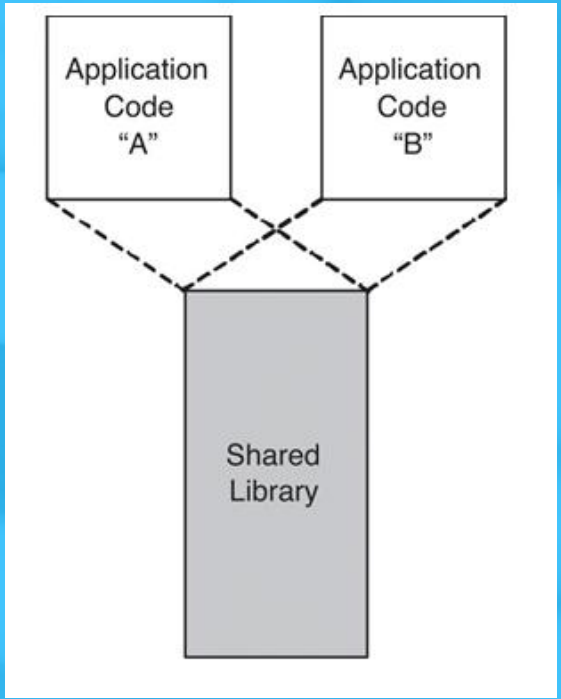


Image source: http://www.bogotobogo.com/cplusplus/images/libraries/static_vs_dynamic_1.png

Trivia - 02

Am I allowed to execute
free(0x53A3CF11)

C++ Idioms

Apache source code

```
AP_CORE_DECLARE(void)
ap_process_connection(conn_rec *c, void *csd)
{
    int rc;
    ap_update_vhost_given_ip(c);

    rc = ap_run_pre_connection(c, csd);
    if (rc != OK && rc != DONE) {
        c->aborted = 1;
    }

    if (!c->aborted) {
        ap_run_process_connection(c);
    }
}
```

Apache source code

Always check for errors

```
AP_CORE_DECLARE(void)
ap_process_connection(conn_rec *c, void *csd)
{
    int rc;
    ap_update_vhost_given_ip(c);

    rc = ap_run_pre_connection(c, csd);
    if (rc != OK && rc != DONE) {
        c->aborted = 1;
    }

    if (!c->aborted) {
        ap_run_process_connection(c);
    }
}
```

Trivia - 03

```
class A{
int* v1,*v2;
public:
   A(){
      v1= new int[n];
      v2= new int[m];
   }
   ~A(){
      delete(v1);delete(v2);
   }
}
…
A* a = new A();
```

Trivia - 03

```
class A{
int* v1,*v2;
public:
   A(){
      v1= new int[n];
      v2= new int[m];
   }
   ~A(){
      delete(v1);delete(v2);
   }
}
...
A* a = new A();
```

```
int* a = new(std::nothrow) int[10];
if(a!=NULL){}
```

OR

```
try{
   a = new int[10];
}
 catch (std::bad_alloc& ba)
  {
    std::cerr << "bad_alloc caught: "
<< ba.what() << '\n';
  }
```

C/C++ runtime

- Internal compiler functions
- Implementations differ

```
DLL 1
…
int* someFunction(){
  int* a = new int[10];
  return a;
}

EXECUTABLE

HINSTANCE hGetProcIDDLL = LoadLibrary(“DLL1");
…
int* a = someFunction();
..
delete(a);//fails if exe and dll were build
using different runtimes
```

C/C++ runtime

Solution:

- The DLL is responsible for cleanup
- The DLL provides its version of alloc/free: eg: DLL_Alloc, DLL_Free

C/C++ runtime

Solution:

- The DLL is responsible for cleanup
- The DLL provides its version of alloc/free: eg: DLL_Alloc, DLL_Free

Do not throw exceptions across DLLs. Use return codes instead

Cross platform issues

- `strdup vs _strdup`
- `char vs wchar_t`

Cross platform issues

- `strdup vs _strdup`
- `char vs wchar_t`

```
#ifdef __unix__
    #define CCHAR char
    #define CSTRDUP strdup

#elif defined(_WIN32) || defined(WIN32)
   #define CCHAR wchar_t
   #define CSTRDUP _strdup

#endif
```

Cross architecture issues

Little endian vs Big endian

Cross architecture issues

Little endian vs Big endian

Detect endianess

```
typedef union{
  unsigned long word;
  unsigned char bytes[sizeof(unsigned
long)];
} detect_endian;
```

Cross architecture issues

Little endian vs Big endian

Detect endianess

```
typedef union{
  unsigned long word;
  unsigned char bytes[sizeof(unsigned long)];
} detect_endian;
```

```
detect_endian d;
d.word=1;
if(d.bytes[0] ==1){
//little endian
}
else if(d.bytes[sizeof(unsigned long)-1]==1){
//big endian
}
else{
//unknown
}
```

Cross architecture issues

Little endian vs Big endian

Detect endianess

```
typedef union{
  unsigned long word;
  unsigned char bytes[sizeof(unsigned long)];
} detect_endian;
```

```
detect_endian d;
d.word=1;
if(d.bytes[0] ==1){
//little endian
}
else if(d.bytes[sizeof(unsigned long)-1]==1){
//big endian
}
else{
//unknown
}
```

Example: LZSS Archiving

Trivia - 04

Can virtual pure functions have a body?

Trivia - 04

Can virtual pure functions have a body?

```
class A{
public:
 virtual ~A(){}
 virtual void print()=0{
    std::cout<<"Predefined message";
 }
};

class B: A{
public:
virtual void print(){
  A::print();
  std::cout<<"Concrete message";
}
virtual ~B(){
}
};
```

Virtual_Constructor

1. Create a copy of an object without knowing its concrete type?

Virtual_Constructor

1. Create a copy of an object without knowing its concrete type?
2. Virtual assignment operator complicated with dynamic casts? How about NO?

Virtual_Constructor

1. Create a copy of an object without knowing its concrete type?
2. Virtual assignment operator complicated with dynamic casts? How about NO?

```
class Employee
{
  public:
    virtual ~Employee () {}
    virtual Employee * create () const = 0; // Virtual constructor (creation)
    virtual Employee * clone () const = 0;  // Virtual constructor (copying)
};
```

Virtual_Constructor

```
class Manager : public Employee
{
  public:
    Manager ();
    Manager (Manager const &);
    virtual ~Manager () {}
    Manager * create () const
    {
      return new Manager();
    }
    Manager * clone () const{
      return new Manager (*this);
    }
};
```

```
class Employee
{
  public:
    virtual ~Employee () {}
    virtual Employee * create () const = 0; // Virtual constructor (creation)
    virtual Employee * clone () const = 0;  // Virtual constructor (copying)
};
```

Virtual_Constructor

```
class Office{
Employee* e1;

};
```

Virtual_Constructor

```
class Office{
Employee* e1;
Office& Office::operator=(const Office & that){
    if(this!=&that){

    }
    return *this;
}
};
```

Virtual_Constructor

```
class Office{
Employee* e1;
Office& Office::operator=(const Office & that){
    if(this!=&that){
        Employee* tmp= e1;
        e1 = that.e1->clone();
        delete e1;
    }
    return *this;
}
};
```

Virtual_Constructor

```
class Office{
Employee* e1;
Office& Office::operator=(const Office & that){
    if(this!=&that){
        Employee* tmp= e1;
        e1 = that.e1->clone();
        delete e1;
    }
    return *this;
}
};
```

```
Employee * duplicate (Employee const & e)
{
  return e.clone();
}
```

Trivia - 05

Can we prohibit
stack allocations?

Trivia - 05

Can we prohibit stack allocations?

```
class A{
public:
 virtual ~A(){}
};

class B: A{
protected:
 ~B(){}
public:
  B(){}
void destroy(){
  delete this;
}
};
```

Trivia - 06

Can we prohibit dynamic allocations?

Trivia - 06

Can we prohibit dynamic allocations?

```
1. class A{
2. private:
3. static void * operator new (size_t s);
4. static void * operator new (size_t,void *);
5. };
```

Static polymorphism
OR
Curiously recurring template pattern

Static polymorphism

```
template <class Derived>
class Base{
public:
void interface_sample1(){
static_cast<Derived*>(this)->implementation_sample1();
}
static void interface_sample2()
{
  Derived::static_implementation_sample2();
}
private:
void implementation_sample1(){}
  static void static_implementation_sample2(){}
};
```

Static polymorphism

```
template <class Derived>
class Base{
public:
void interface_sample1(){
static_cast<Derived*>(this)->implementation_sample1();
}
static void interface_sample2()
{
  Derived::static_implementation_sample2();
}
private:
void implementation_sample1(){}
  static void static_implementation_sample2(){}
};
```

```
class Derived1: public base<Derived1>{
public:
static void static_implementation_sample2(){
  std::cout<<"hello from static method\n";
}
void implementation_sample1(){
  std::cout<<"Hello from non static method\n";
}
};
```

Static polymorphism

```
template <class Derived>
class Base{
public:
void interface_sample1(){
static_cast<Derived*>(this)->implementation_sample1();
}
static void interface_sample2()
{
  Derived::static_implementation_sample2();
}
private:
void implementation_sample1(){}
  static void static_implementation_sample2(){}
};
```

```
class Derived1: public base<Derived1>{
public:
static void static_implementation_sample2(){
  std::cout<<"hello from static method\n";
}
void implementation_sample1(){
  std::cout<<"Hello from non static method\n";
}
};
```

```
base<Derived1> *b = new Derived1();
b->interface_sample1();
b->interface_sample2();
```

Trivia - 07

What is wrong with this function?

```
1.  void log(const char* msg){
2.    char buf[LARGE_ENOUGH_BUFFER];
3.    sprintf(buf,“Logging %s\n”,msg);
4.  //save it on disk
5.  }
6.  …
7.  void authenticate(const char* username){
8.    bool isAuthenticated=false;
9.    log(username);
10. …

11. }
```

Meta functions

- To encapsulate a complex type computation algorithm
- To generate a type using compile-time type selection techniques

Factorial

```
template <int N>
struct Factorial {
    enum { value = N * Factorial<N - 1>::value };
};
template <>
struct Factorial<0>
{
    enum { value = 1 };
};
void foo()
{
    int x = Factorial<4>::value; // == 24
    int y = Factorial<0>::value; // == 1
}
```

Meta functions

- To encapsulate a complex type computation algorithm
- To generate a type using compile-time type selection techniques

Compile time type selection

```
template <bool, class L, class R>
struct IF
{
  typedef R type;
};
template <class L, class R>
struct IF<true, L, R>
{
  typedef L type;
};
```

Meta functions

- To encapsulate a complex type computation algorithm
- To generate a type using compile-time type selection techniques

Compile time type selection

```
template <bool, class L, class R>
struct IF
{
  typedef R type;
};
template <class L, class R>
struct IF<true, L, R>
{
  typedef L type;
};
…
IF<true,int,long long>::type j;
std::cout<<sizeof(j); //4

IF<false,int,long long>::type j;
std::cout<<sizeof(j); //8
```

Trivia - 08

What is the problem with the following function?

```
1.  void custom_cpy(char* dst,const char* src,int n){
2.        memcpy(dst,src,n);
3.  }
```

Trivia - 08

What is the problem with the following function?

```
void custom_cpy(char* dst,const char* src,int n){
      memcpy(dst,src,n);
}

…
char str1[9] = "aabbccdd";
printf("The string: %s\n", str1);//aabbccdd
custom_cpy(str1 + 2, str1, 6);
printf("New string: %s\n", str1);
//expecting aaaabbcc
```

Trivia - 08

What is the problem with the following function?

```
void custom_cpy(char* dst,const char* src,int n){
      memcpy(dst,src,n);
}

…
char str1[9] = "aabbccdd";
printf("The string: %s\n", str1);//aabbccdd
custom_cpy(str1 + 2, str1, 6);
printf("New string: %s\n", str1);
//expecting aaaabbcc but surprise: aaaabbbb
```

Trivia - 08

What is the problem with the following function?

```
void custom_cpy(char* dst,const char* src,int n){
      memmove(dst,src,n);
}

…
char str1[9] = "aabbccdd";
printf("The string: %s\n", str1);
custom_cpy(str1 + 2, str1, 6);
printf("New string: %s\n", str1);
```

Structure versioning

```
//include/publicheader.h
struct InitInfo{
   int version;
   int type;
   char* path;
};
```

Structure versioning

```
//include/publicheader.h
struct InitInfo{
   int version;
   int type;
   char* path;
};
```

InitInfo

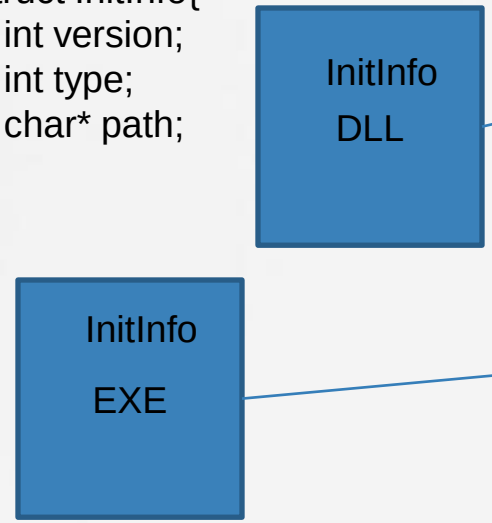
DLL

InitInfo

EXE

Structure versioning

```
//include/publicheader.h
struct InitInfo{
   int version;
   int type;
   char* path;
};
```

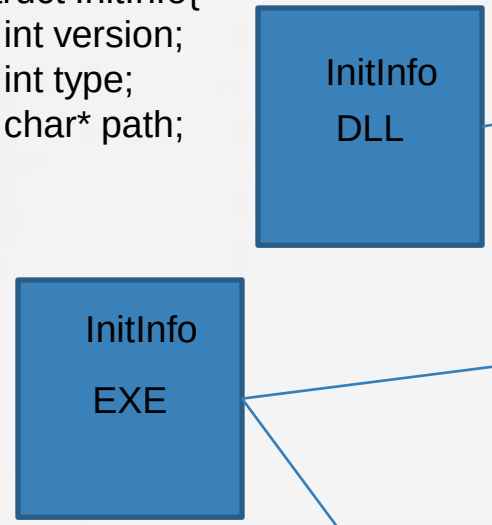



```
struct InitInfo_v1{
   int version;
   int type;
   char* path;
};
```

```
struct InitInfo{
   int version;
   int type;
   char* path;
   char* registry;
};
```

Structure versioning

```
//include/publicheader.h
struct InitInfo{
   int version;
   int type;
   char* path;
};
```




```
struct InitInfo{
   int version;
   int type;
   char* path;
   char* registry;
};
struct InitInfo_v1{
   int version;
   int type;
   char* path;
};
```

DLL_Init(InitInfo,…);

Structure packing

```
struct InitInfo{
   char version;
   int type;
   char flags;
};
```

HW
DEVICE

Structure packing

```
struct InitInfo{
   char version;
   int type;
   char flags;
};
```

1 byte version	4 bytes type	1 byte flag

Structure packing

```
struct InitInfo{
   char version;
   int type;
   char flags;
};
```

```
#pragma pack(push, 1)
struct InitInfo{
   char version;
   int type;
   char flags;
};
```

1 byte version	4 bytes type	1 byte flag

Take away

- Read some standards (C++11 standard)
- Contribute to open source projects
- Good books:
    1. Linkers and loaders
    2. Inside the C++ Object Model (great read for knowing implementation details)

Image source: http://www.bogotobogo.com/cplusplus/images/libraries/StaticLibraryLink.png

Thanks!

Any questions?

CREDITS

Special thanks to all the people who made and released these awesome resources for free:

- Presentation template by SlidesCarnival
- Photographs by Death to the Stock Photo (license)