Table of Contents

ATLAS-Japan C++ Tutorial in 2016

Introduction

ん が り 、今 時代ググ&

がっ そ 不安 方

- 東大坂 先生 C++トレーニン&
- http://atlas.kek.jp/comp/index.html
    も（ちょっ 古 が）色&

り 勉強 をお勧め

クラ オブジェ&#1246

C++ クラ も が使え っ れ&#124
っ クラ を知ら 人向け が&#

、 ログラミングを 君 ち &#12
ロセ 内 新 も を創造 き、&#123

具体 ど やっ クラ を定義 &#123
最後 Exercise 黒板を使っ 触れ &#1
一度クラ を定義 えば（ &#1242
オブジェクトを作成 、 &#1

```
int var;
```

やっ

```
MyClass obj;
```

み 感じ り ポインタ 作成

クラ メンバー `public`, `private`, `protected`
属性を付け が可能
む ろ明示 が推奨 れ
objectから指定 き も が `public`,
そ も が `private` (or `protected`)
`public`
メンバー メンバアクセ 演&#
(. も `->`)を使っ アクセ き

```
MyClass obj;
obj.Set("hoge");

MyClass* obj = new MyClass();
obj->Set("hoge2");
```

オブジェクト指向 カ セル&#2
良 せん
、それら アクセ メンバ&#125

public, メンバー**変数** private を使
protected
属性 自分 クラ 、も 継承&#20

そ 他知っ お め

オブジェクトを作 &#6 - memory allocation, ポインタ, 参照

Will Buttinger - offline software tutorial
2015 ライド り拝借

オブジェクト 作成・削除・memory allocation

オブジェクトをC++ 作 、そ&#1242
れ 2種類 (**stack heap**)が り

```
MyClass obj;   // creating instance of MyClass on the stack
int myInt;       // creating instance of int on the stack
MyClass* obj2 = new MyClass();     // creating instance of MyClass on the heap
int* myInt2 = new int();              // creating instance of int on the heap
```

stack

- read/writeが速
- heap り サイズが小
- 自動 削除 れ （自分 **delete** 必要が ）

一方 heap

- stack り 遅
- stack り サイズが大き
- 自分 **delete** どを け （ROOT上 多少や&#1238

一つ例を上げ

```
int main()
{
  int a;
  int* b = new int;      // note: no brackets means using default constructor
  {
    int c;
    int* d = new int;
  }       // At this point c is deleted. The int that d points to hasn't though... MEMORY LEAK!!
  delete b;       // good, we're cleaning up our heap allocation
}     // a gets deleted from the stack here.
```

ポインタ 一言 言えば、メ&#

```
MyClass obj;     // creating instance of MyClass on the stack
int myInt;         // creating instance of int on the stack
MyClass* obj2 = new MyClass();      // creating instance of MyClass on the heap
int* myInt2 = new int();              // creating instance of int on the heap
```

クラ オブジェクト指&#21512

```
MyClass* obj3 = &obj;      // obj3 points at obj
int* myInt3 = &myInt;      // myInt3 points at myInt
```

アドレ っ メモリ も小
(64bitマシン 8 bytes (=64 bits))
  関数 受け渡 き ど オブジ&#12

- basic type (`bool`, `int`, `float`, ...) 差が り せんが
- 関数 引数 イン タン を与&#
  そ そ 関数が終わ き 自動
  オリジナル オブジェク&#1248
- 引数 ポインタを与え ア&#1
  コピー れ アドレ アクセ

```
void myBadFunction(MyClass obj) { obj.myMethod(); }
void myGoodFunction(MyClass* obj) { obj->myMethod(); }

int main()
{
  MyClass obj;
  myGoodFunction(&obj);
}
```

んか らオブジェクトを作&#1
つ 」「 **ど**
」消え （消 ） かを考え 癖&
stack 場合、自動 削除 れ もscope
(括弧 `{ }`)
  外側 アクセ け せん

```
MyClass* b;      // created a pointer, but not made it point at anything
{
    MyClass c;
    b = &c;     // b now points at c.
}    // c is deleted, but we've dangerously kept it's address (held in b)
b->myMethod();     // Can't do this, c doesn't live at the address held by b anymore
```

heap 場合、必ず自分 deleteを け &#1
ダメ 例を示

```
void myBadFunction() {
     MyClass* obj = new MyClass();
     if(obj->someMethod()) return;
     delete obj;      // too late if someMethod() returned true...
}
```

参照

最後 参照 つ 復習 ょ

- MyClassタイ イン タン **ポイン&**
  `MyClass*`
- MyClassタイ イン タン **参照** 型
  `MyClass&`
- 参照 アドレ 代わり オブ&

```
MyClass obj
MyClass* objPtr = &obj;
```

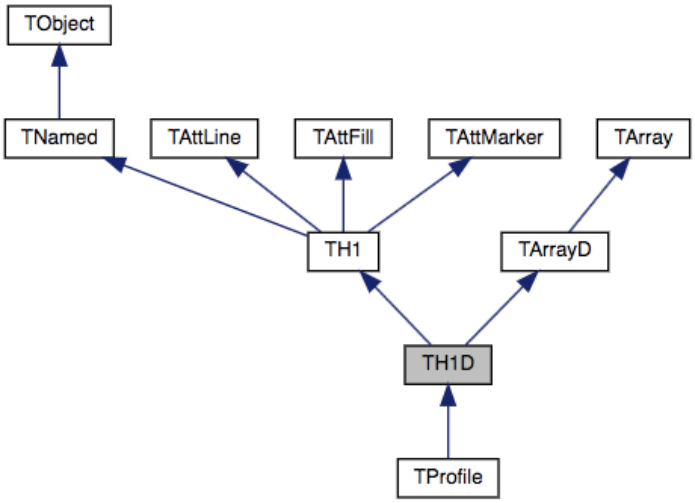
オブジェクト 作成・削&#38500

```
MyClass& objRef = obj;
```

継承, Virtual 関数, ャ ト

継承 virtual 関数

継承 身近 例を示



実際 クラ リファレン を見
例えば `Add(const TH1* h1, ...)`
    関数 TH1D 定義 れ せん
TH1クラ 定義 れ れ TH1DやTH1F 同&#1
    オブジェクト指向 ログラ&
親クラ 何 操作が か、も &#20

メンバー関数を一つ持っ &#

```
class MyClassA
{
 public:
  void myMethod() { std::cout << "It's MyClassA" << std::endl; }
};

class MyClassB : public MyClassA
{
 public:
  void myMethod() { std::cout << "It's MyClassB" << std::endl; }
};
```

 れを実際 使用 み ポイン&#12479
/
参照 型 タイ 依存 結果が&#310

```
MyClassB* b = new MyClassB();
MyClassA* b_as_a = b;

b->myMethod();        // prints "It's MyClassB"
b_as_a->myMethod();   // prints "It's MyClassA"
```

    関数をvirtual み ょ

```
class MyClassA
{
 public:
  virtual void myMethod() { std::cout << "It's MyClassA" << std::endl; }
```

```
};

class MyClassB : public MyClassA
{
 public:
  virtual void myMethod() { std::cout << "It's MyClassB" << std::endl; }
};
```

コンパイラが呼ぶ関数 実&

```
MyClassB* b = new MyClassB();
MyClassA* b_as_a = b;

b->myMethod();         // prints "It's MyClassB"
b_as_a->myMethod();    // prints "It's MyClassB"
```

をポリモーフィズム（&#228

Casting

ャ ト つ も学ん おき ょ

- C-style cast
- static_cast
- dynamic_cast
- reinterpret_cast
- const_cast

5つ り が知っ お ほ も static_cast dynamic

dynamic_cast

安全 底クラ から派生クラ &
め 使
（派生クラ から 底クラ へ

static_cast

主 派生クラ から 底クラ へ
危険 ャ トをコンパイルエ&#

使 方一つだけ例を示

```
int ival;
long lval = static_cast<long>(ival);
```

Coding rule

今日言 実 だけ っ り ら &#

- 自分が読ん ぐ 理解 き コ&
- 他 人も使 （使っ 欲 ） ら

か 一定 ルール 従っ コー&#1248
`m_` 始め か）

ATLAS 推奨コーディング法 も

- ATLAS C++ Coding Guidelines - ATL-COM-SOFT-2016-001

一部古 情報も り が日 語版

- ATLAS C++ コーディング タンダー&#1248 2003年版

、一Ƅ ソー コード ファイ きちん オブジェクト志向 （1万行を超え コード ん 読&

Exercise 1 - C++ programming

自分 一つクラ を用 ログラ

- C++コード コンパイル方&#2786
- 一つ ログラム め 複数 コ
- クラ を作っ み

フォーカ

り えずど か作業 場所を作

```
[junpei@login01 ~]$ mkdir cxxtut2015
[junpei@login01 ~]$ cd cxxtut2015
```

Particle 粒子 クラ を作り クラ

- 粒子 名前
- エネルギー
- 電荷

３つ パラメータを保持 粒子 情報を出力 き 機能&#12 クラ 定義 ヘ ダファイル (`Particle.h`) を き ょ

Show Particle.h Hide Particle.h

```
#ifndef Particle_h
#define Particle_h

#include <string>

class Particle
{
 public:
  Particle();
  virtual ~Particle();

  virtual void printInfo();

  inline void setName(std::string name) { m_name = name; }
  inline void setEnergy(double energy) { m_energy = energy; }
  inline void setCharge(const int charge) { m_charge = charge; }
```

```
  inline std::string getName() const { return m_name; }
  inline double getEnergy() const { return m_energy; }
  inline int getCharge() const { return m_charge; }

 protected:
  std::string m_name;            ///< particle name
  double m_energy;             ///< 4-momentum
  int m_charge;                        ///< charge
};

#endif
```

Particleクラ 関数 身 (`Particle.cxx`) 実装 ょ

Show Particle.cxx Hide Particle.cxx

```
#include "Particle.h"

#include <iostream>

Particle::Particle()
{
  m_name = "NOT DEFINED";
  m_energy = 0.0;
  m_charge = 0;
}

Particle::~Particle()
{}

void Particle::printInfo()
{
  std::cout << "Particle Name = " << m_name
                << ", energy = " << m_energy
                << ", charge = " << m_charge << std::endl;
}
```

そ main()を作っ クラ を呼び (`exercise.cxx`)

Show exercise.cxx Hide exercise.cxx

```
#include "Particle.h"

int main(void)
{
  Particle electron;

  electron.setName("electron");
  electron.setEnergy(1000.);   // 1000 MeV
  electron.setCharge(-1);

  electron.printInfo();

  return 0;
}
```

コンパイル み

```
[junpei@login01 cxxtut2015]$ g++ -c Particle.cxx
[junpei@login01 cxxtut2015]$ g++ exercise.cxx Particle.o -o exercise.exe
```

せっか 実行

```
[junpei@login01 cxxtut2015]$ ./exercise.exe
```

This topic: Main > AtlasJapanCppTutorial2016
Topic revision: r7 - 2016-12-25 - JumpeiMaeda