C++11: Syntax and Feature

第1回

（1.概要、2.字句規約、3.基本事項、4.標準型変換）

まずはじめに

- C++11: Syntax and Feature(http://ezoeryou.github.io/cpp-book/C++11-Syntax-and-Feature.xhtml)を参考に進める

- gccでは**--std=c++11**オプションをつける

1章 概要

C++11とは

- 2011年にメジャーアップデートされたC++の標準規格

- 1998年に初の標準規格（C++98）、2003年にマイナーアップデート（C++03）

- 2017年にメジャーアップデート予定（C++1y）

用語

- 仮引数…関数宣言/定義、catch、マクロ、テンプレートの引数
- 実引数…実際に渡される引数

```
int f(int x) { // xは仮引数
    return x * x + 2 * x + 1;
}

int main() {
    try {
        f(3); // 実引数
    } catch(Exception &e) { // 仮引数
        cerr << "Error!" << endl;
    }
    return 0;
}
```

用語

- 動的な型…基底クラスのポインタに派生クラスを入れると、apの動的な型はBになる（⇔静的な型）

```
class A { /* hoge */ };
class B : public A { /* fuga */ };

B bv;
// apの型は実行時はBになっている
A *ap = &bv;
```

用語

- シグネチャ…関数を一意に特定する情報

```
void func(int a, char b) {
    /* hoge */
}
```

2章 字句規約

ソースファイルの変換

- プリプロセッサ→**ソースの変換**→コンパイル

ソースファイルの変換

- 基本ソース文字セット

a b c d e f g h i j k l m n o p q r s t u v w x y z

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

0 1 2 3 4 5 6 7 8 9

_ { } [ ] # ( ) < > % : ; . ? * + - / ^ & | ~ ! = , \ " '

ソースファイルの変換

- 基本ソース文字セット

a b c d e f g h i j k l m n o p q r s t u v w x y z

A B C D W X Y Z

0 1 2 3 4 5 6 7 8 9

_ { } [ ] # ( ) < > % : ; . ? * + - / ^ & | ~ ! = , \ " '

たとえASCIIでも@や`（グレイヴ・アクセント）は使えない！

ソースファイルの変換

- 基本ソース文字セット外の文字はこの変換によってUCN(=Universal Character Name)に変換

(例)　　あ→\u3042

い→\u3042

ソースファイルの変換

- 行末の\（back slash）+\n（new line）は除去

ソースファイルの変換

- 行末の\（back slash）+\n（new line）は除去

```
1 #include <iostream>
2 using namespace std;
3
4 int main() {
5 /\
6 /\
7 comment
8 }
```

これも一応コンパイルできる

ソースファイルの変換

- 連続する文字列リテラルの連結

（例）”aaa” “bbb” → “aaabbb”

UCN

- \uXXXX
- 基本的にUnicodeを16進数で指定
- エスケープシーケンスっぽいけど、ソースコードの任意の箇所で仕様でき、コンパイル前に置換

UCN

```
levelfour@lemon program$ cat a.cpp
#include <iostream>
using namespace std;

int main() {
        cout << "\u304a\u306f\u3088\u3046\u65e5\u672c" << endl;
        return 0;
}
levelfour@lemon program$ g++ -o a a.cpp
levelfour@lemon program$ ./a
おはよう日本
levelfour@lemon program$
```

UCN

直接文字通りに解釈する

```
levelfour@lemon program$ cat a.cpp
#include <iostream>
using namespace std;

int main() {
        cout << R"(\u304a\u306f\u3088\u3046\u65e5\u672c)" << endl;
        return 0;
}
levelfour@lemon program$ g++ -std=c++11 -o a a.cpp
levelfour@lemon program$ ./a
\u304a\u306f\u3088\u3046\u65e5\u672c
levelfour@lemon program$
```

UCN

- 「int \u3042 = 0;」 みたいなのもOK

トークン

- 識別子
- キーワード
- リテラル
- 演算子
- delimiter

コメント

- /* */ と // の二種類

- /* */はネストできないことだけ注意

予約語

- C++の実装や標準ライブラリ用に予約された識別子

（例）_ + 大文字から始まる名前

連続するアンダースコア2つを含む名前

予約語

- C++の実装や標準ライブラリ用に予約された識別子

（例）_

アンダースコアをむやみに使うのは
避けよう！！！

連続するアンダースコア2つを含む名前

予約語

```cpp
#include <iostream>
using namespace std;

int main() {
    int _ = 42;
    cout << _ << endl;
    return 0;
}
```

これはOK

キーワード

alignas	continue	friend	register	true
alignof	decltype	goto	reinterpret_cast	try
asm	default	if	return	typedef
auto	delete	inline	short	typeid
bool	do	int	signed	typename
break	double	long	sizeof	union
case	dynamic_cast	mutable	static	unsigned
catch	else	namespace	static_assert	using
char	enum	new	static_cast	virtual
char16_t	explicit	noexcept	struct	void
char32_t	export	nullptr	switch	volatile
class	extern	operator	template	wchar_t
const	false	private	this	while
constexpr	float	protected	thread_local	
const_cast	for	public	throw	

キーワード

alignas	continue	friend	register	true
alignof	decltype	goto	reinterpret_cast	try
asm	default	if	return	typedef
auto	delete	inline	short	typeid
bool	do	int	signed	typename
break	double	long	sizeof	union
case	dynamic_cast	mutable	static	unsigned
catch	else	namespace	static_assert	using
char	enum	new	static_cast	virtual
char16_t	explicit	noexcept	struct	void
char32_t	export	nullptr	switch	volatile
class	extern	operator	template	wchar_t
const	false	private	this	while
constexpr	float	protected	thread_local	
const_cast	for	public	throw	

→キーワードは識別子に使えない

コンテクスト依存キーワード

- final（継承不可クラス）とoverride（明示的オーバーライド）

- 特定の箇所にしか登場しないので、これらは識別子として使える

リテラル

- 整数リテラル(10, 0x10, 010, 10u, 10l, 10ll, 10lu)
- 浮動小数点数リテラル(1.0, 1.0e-12, 1.0E5, 1.0f)
- 文字リテラル（'x', u'x', U'x', L'x'）
- 文字列リテラル（''abc'', u8''abc'', U''abc''）

リテラル

- 一文字= | 要素ではないことに要注意

```
#include <iostream>
using namespace std;

int main() {
        char str[] = u8"あ";
        cout << "sizeof str = " << sizeof str << endl;
        return 0;
}
levelfour@lemon program$ ./a
sizeof str = 4
levelfour@lemon program$
```

リテラル

- 生文字列リテラル

```
int main() {
        char str[] = R"(1
abc
        hoge
             )";
        cout << str << endl;

        return 0;
}
levelfour@lemon program$ ./a
1
abc
        hoge

levelfour@lemon program$
```

リテラル

- ポインターリテラル(nullptr)
- ユーザ定義リテラル

以上のようなリテラルをユーザ自身が関数を用いて定義できる（詳しくは13章）

3章 基本事項

宣言と定義

- 宣言…名前の意味内容を明示
- 定義…名前の指し示す具体的なものの記述

$$定義 \in 宣言$$

宣言と定義

- 変数

宣言	定義
`double pi;`	`double pi = 3.14159;`

宣言と定義

- 関数

宣言	定義

`int func(int x);`	`int func(int x) {`
	`    return x*x + 2*x + 1;`
	`}`

宣言と定義

- クラス

宣言	定義

class A;	class A {
	private:
	    int m_var;
	public:
	    A();
	    virtual ~A();
	};

ODR

- ODR = One Defined Rule （定義は一つしか書けない）

```
1 int x = 10;
2 int x = 20; // compile-error
3 printf("%d", x);
~
~
~
```

ODR

- 例外：クラス

a.cpp	b.cpp

class A {
private:
int m_var;
public:
A() : m_var(10) {};
virtual ~A() {};
do() {
cout << “hoge\n”;
};
};
	class A;
	
	A a_ins;
	
	// A::do()が参照できない
	a_ins.do();

ODR

- クラス、enum、クラステンプレート、externなinline関数、externな関数テンプレート etc.はODRの例外

スコープ

- スコープの上書き

```
1 int x = 10;
2 {
3     int x = 20;
4     x; // -> 20
5 }
6 x; // -> 10
```

スコープ

- ラベルのスコープは関数を抜けるまで

```
1 void f() {
2     {
3         label:
4     }
5     goto label;
6 }
```

名前探索

- スコープにおいてある名前の意味するものを決定すること
- ::（スコープ解決演算子）を使う
- 頭に::をつけるとグローバル空間

ADL

- Argument Dependent name Lookup

```
namespace NS
{
    class C {} ;
    void f(C) {}
}

int main()
{
    NS::C c ;
    f(c) ; // -> errorにならない
}
```

演算子オーバーロード
に便利

プログラムの開始

- main関数から開始

- 非ローカル変数(スタック変数以外)の初期化

4章 標準型変換(暗黙の変換)

ポインタへの変換

配列→ポインタ	関数→ポインタ

`int a[10] ;`
`int * p = a ;`
	`void f() {}`
	
	`int main()`
	`{`
	`  using T = void(*)() ;`
	
	`  T p1 = f ;`
	`  T p2 = &f ;`
	`}`

整数の型変換、昇格

- 昇格…intより低い型→int

- 型変換…それ以外

整数の型変換、昇格

- 昇格…intより低い型→int
- 型変換…それ以外

演算子オーバーロードで重要な違いになる！

真偽値への型変換

- 0, nullptr → false

- otherwise → true