



# New Features in C99 (1)

2003.10.11

**KLDP** 작은 세미나

전웅 (mycoboco@hanmail.net)

## <목표>

---

- ◆ C99의 새로운 기술을 이해하기 위한 배경 설명 (기존 표준과의 차이 및 도입 배경)
- ◆ C99에 새로 추가된 기술 일부에 대한 간략한 소개

## <구성>

---

- ◆ C 표준화의 간략한 역사
- ◆ C99 표준화의 원칙 (guideline)
  - ◆ 새 기술 도입에 대한 항목을 중심으로
- ◆ 각 기술에 대한 소개 (21개 - 총 53개)
  - ◆ 표준에 의해 명시된 대표적 기술 중심  
(명시되지 않은 기술도 다수지만 실제 프로그래밍 환경에 주는 영향 없음)
  - ◆ C90과 깊은 연관성 갖는 것을 우선으로

# <C 표준화의 역사>



## <표준화 원칙>

(새 기술 도입에 대한 항목을 중심으로)

---

- ◆ Existing code is important, existing implementations are not
- ◆ Avoid “quiet changes”
  
- ◆ Support international programming
- ◆ Codify existing practice to address evident deficiencies
- ◆ Minimize incompatibilities with C90
- ◆ Minimize incompatibilities with C++

# 1. Restricted character set support via digraph and <iso646.h>

---

- ◆ “표준” C 언어의 기반 character set: ASCII가 아닌 ISO 646 Invariant Set (ASCII 의 subset)
- ◆ C 언어는 동시대 다른 언어에 비해 다양한 그래픽 문자 사용
- ◆ ASCII에는 존재하지만 ISO 646 Invariant Set 에는 존재하지 않는 9개 문자를 위해 trigraph 도입
  - ◆ 기존 프로그램에 대한 영향을 최소화하기 위해 ?? 로 시작하는 9개의 trigraph 정의
  - ◆ 토큰화 이전에 치환이 이루어짐

# 1. Restricted character set support via digraph and <iso646.h> (Cont')

---

## ◇ Trigraph 사용의 예

```
??=include <stdio.h>
```

```
int main(void)
```

```
??<
```

```
printf("Hello, world??-??/n");
```

```
printf("What??!");
```

```
return 0;
```

```
??>
```

# 1. Restricted character set support via digraph and <iso646.h> (Cont')

---

- ◇ trigraph 보다 나은 6개의 digraph 도입 - trigraph와는 달리 하나의 독립된 토큰으로 사용됨

```
%:include <stdio.h>
```

```
int main(void)
```

```
%<
```

```
printf("Hello, world??-??/n");
```

```
return 0;
```

```
%>
```



# 1. Restricted character set support via digraph and <iso646.h> (Cont')

---

- ◆ 프로그램의 가독성 증진을 위한 **<iso646.h>** 매크로 지원
  - ◆ trigraph나 digraph로 쓰여져야 하는 연산자들을 매크로로 제공

```
and (&&)  
and_eq (&=)  
or (||)  
compl (~)
```

## 2. More precise aliasing rule

---

- ◇ C 표준은 모든 종류의 aliasing을 허락하지는 않음

```
void func(int *pi, unsigned int *pui)
{
    *pui = 2;
    *pi = 3;
    another func(*pui);    /* ? */
}
```

```
int i;
func(&i, (unsigned int *)&i);
```

## 2. More precise aliasing rule (Cont')

---

```
void func(int *pi, float *pf)
{
    *pf = 2.0;
    *pi = 3;
    another func(*pf);      /* ? */
}

int i;
func(&i, (float *)&i;)    /* wrong */
```

## 2. More precise aliasing rule (Cont')

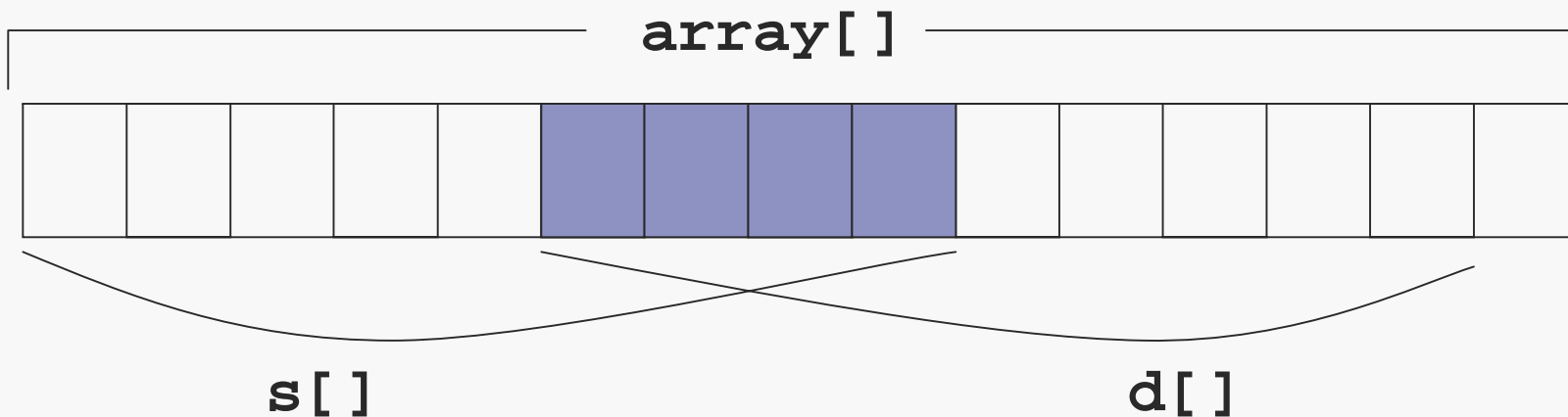
---

- ◆ C90의 aliasing rule은 **malloc()**로 할당된 object 등과 관련하여 불완전한 형태의 규칙이었음
- ◆ C99에서는 effective type이라는 개념을 도입하여 aliasing rule과 관련된 문제의 “거의” 해결 – union과 관련된 문제로 여전히 개정 중

### 3. Restricted pointer

- ◆ array object의 부분이 aliasing되어 overlapped object에 대한 read/write 작업이 있는 경우, vector processing 같은 최적화에 치명적인 영향을 줌

```
void func(double *d, const double *s);
```



### 3. Restricted pointer (Cont')

---

- ◆ C90에서는 이와 같은 행동을 억제하는 권고만을 했을 뿐
- ◆ C99에서 restricted pointer의 도입으로 보다 나은 최적화 가능

```
void func(double *restrict d,  
          const double *restrict s);
```

- ◆ 동시에 overlapped object를 피해야 하는 함수들(**memcpy()** 등)의 기술이 용이해짐

## 4. Flexible array member

```
struct foo {
    int number;
    double bar[100];
} *flexible;

flexible = malloc(
    sizeof(struct foo)
    - sizeof(double)*100
    + sizeof(double)*n);

flexible->number = n;
flexible->a[n-1] = 0;
/* wrong */
```

◇ C99에서 적법한 방법

```
struct foo {
    int number;
    double bar[1];
} *flexible;

flexible = malloc(
    sizeof(struct foo) +
    sizeof(double)*(n-1));

flexible->number = n;
flexible->a[n-1] = 0;
/* wrong */
```

지원

## 5. Increased minimum translation limits

---

- ◆ implementation에게 어느 정도의 물리적인 자원을 요구하기 위한 규정: translation limits
- ◆ 일종의 guideline의 성격 (따라서, rubber teeth라고 불림)
- ◆ 컴퓨팅 환경의 발전에 따라 적절한 수준으로 확대되었음



## 6. Remove implicit `int`

---

- ◆ typeless 언어 (BCPL, B) 에서 발전해온 C 언어의 특성으로 과거 언어의 잔재가 남아 있음

```
main() /* int main() */ { return 0; }  
func(a, b) /* int a, b; */ { return a+b; }  
void foo(const i); /* const int i; */
```

- ◆ C90에서는 backward compatibility를 위해 그대로 유지
- ◆ C99에서는 과감하게 제거하였음

## 7. `return` without expression

- ◆ implicit `int`는 `void` 형이 지원되지 않던 시절 `void`의 역할을 대신하기도 했음

```
proc() /* int proc() */
{
    /* do some jobs */
    return;
}
```

- ◆ implicit `int`가 사라졌으므로 반환형이 `void`가 아닌 함수에서 `return;`은 더 이상 허락될 필요 없음 – C99 에서 금지
- ◆ 그 이전 (C99 이전) 에는 알 수 없는 반환값이 사용되지 않으면 위와 같은 행동이 허락되었음

## 8. Reliable integer division

- ◆ C90에서 정수 나눗셈 연산의 두 피연산자 중 하나라도 음수면, 결과의 반올림 방향은 implementation-defined - 양의 정수 나눗셈의 경우 round toward zero (소수부 자름) 로 반올림됨

```
int q = 7 / -3;    /* -2 or -3 */  
int r = 7 % -3;    /* 1 or -2 */
```

단,  $a \% b == a - ((a / b) * b)$  를 만족

- ◆ C99에서는 음의 정수 나눗셈 역시 양의 정수 나눗셈과 마찬가지로 round toward zero로 정의되었음

## 9. Designated initializers

- ◇ 공용체 초기화의 문제 – 선언된 첫번째 멤버만을 초기화할 수 있었음

```
union {  
    int a;  
    double b;  
} foo = { 12 };
```

- ◇ 배열, 구조체 초기화의 문제 – 중간에 존재하는 멤버를 초기화하기 위해서는 그 앞의 요소나 멤버의 초기치를 모두 명시해야 함
- ◇ C99에서는 특정 멤버나 배열 요소를 지정해 초기화할 수 있는 방법 제공

```
union { int a; double b; } foo = { .b = 3.14 };  
int a[100] = { [52] = 7903 };
```

## 10. Relaxed constraints on aggregate and union initializers

---

- ◆ C90에서는 object의 storage duration이 static 이라도 데이터형이 aggregate 혹은 union type 일 경우, 초기치에는 상수 수식만이 허락됨
- ◆ C99에서는 scalar type의 초기치와 마찬가지로 auto storage duration의 aggregate 혹은 union type object의 초기치에 대한 제한 완화

```
void func(int n)
{
    int array[10] = { n, n+1, n+2, };
    /* wrong in C90, valid in C99 */
}
```

## 11. // comment

- ◆ C++와의 호환성 증진을 위해 // 형태의 주석을 받아들임
- ◆ 기존의 많은 컴파일러가 C 프로그램에 대해서도 이미 // 주석을 지원했으나 이는 표준을 따르지 않는 확장 (non-conforming extension)
- ◆ // 주석의 도입은 드문 경우지만 quiet change에 해당

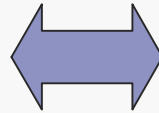
```
int c_or_cpp = 10 /** wow */ 10  
;
```

## 12. Remove implicit function declaration

---

- ◆ 함수의 implicit declaration 제거 – 모든 명칭은 사용되기 전에 적절히 선언되어야 한다는 현대 프로그래밍 언어의 원칙

```
int main(void)
{
    foo(SOME_VALUE);
    ...
}
```



```
int main(void)
{
    extern int foo();
    foo(SOME_VALUE);
    ...
}
```

## 13. Mixed declaration and code

- ◇ C90에서는 block 내에서 선언이 문장에 우선해야 함
- ◇ C99에서는 block 내에서 선언과 문장이 섞어 나올 수 있음

```
{
    double result, sum;

    /* logical unit #1 */
    int first_result;
    /* statements */

    /* logical unit #2 */
    int second_result;
    /* statements */
    ...
}
```

```
{
    double result, sum;

    { /* logical #1 */
        int first_result;
        /* statements */
    }

    { /* logical #2 */
        int second_result;
        /* statements */
    }
}
```



## 14. Macro with a variable number of arguments

---

- ◆ C90에서 한 매크로가 받을 수 있는 인자의 개수는 매개변수의 개수에 의해 고정됨

```
#define debug(s, args) printf(s, args)
debug_print("%d, %d\n", i, j); /* wrong */
```

- ◆ 확장이 아닌 순수한 C90에서 가변인자 매크로를 흉내내기 위한 편법

```
#define debug(args) printf args
debug((" %d, %d\n", i, j));
```

---

```
#define _ ,
#define debug(s, args) printf(s, args)
debug("%d, %d\n", i _ j);
```

## 14. Macro with a variable number of arguments (Cont')

---

- ◆ C99에서는 **<stdarg.h>** 방식과 유사한 가변인자 매크로 지원

```
#define debug(s, ...) printf(s, __VA_ARGS__)
```

- ◆ 이 방법은 일반 함수의 가변인자 지원과 유사하다는 장점을 갖지만, 기존 implementation에서 찾아보기 어려운 형태 - 사용 중인 implementation이 C99 방식을 지원한다면 미래를 위해 표준 방식 사용을 권장

## 15. Empty macro argument

---

- ◆ 다수의 implementation이 확장으로 비어 있는 (empty token) 매크로 인자를 지원 하지만, C90에서는 엄격하게 잘못된 행동

```
#define make_obj(prefix, num) \  
    char *prefix ## num = #num  
decl_ptr(element, 13);  
    /* char *prefix13 = "13"; */  
decl_ptr(element, );    /* wrong in C90 */  
    /* char *prefix = ""; */
```

- ◆ C99에서는 허락

## 16. %lf allowed in printf

- ◆ C90에서 **printf()**에 사용되는 format specifier 중에는 **%lf**가 존재하지 않았음 - 그럼에도 많은 사용자들이 **double** 형의 값을 출력하기 위해 오해로 사용하고, 이를 위해 다수의 implementation이 확장으로 지원해 왔음

```
double d = 3.14159;
printf("%lf", d);    /* wrong in C90 */
/* same as printf("%f\n", d); in C99 */
```

- ◆ C99에서는 **%lf**를 **double**을 위한 specifier로 정식 정의했음 (결국 **%f** 와 동일)

## 17. `snprintf()` family

---

- ◆ **`sprintf()`** 류의 함수는 buffer overrun 과 관련된 위험성으로 안전성을 고려한 프로그램에서는 확장으로 제공되는 **`snprintf()`** 류의 함수를 사용해 왔음
- ◆ **`snprintf()`** 함수는 C89 시절에 제안되었으나 2/3 투표를 얻지 못해 거절
- ◆ C99에서 다수의 prior art를 확보하여 받아들여지게 되었음

## 18. Idempotent type qualifiers

---

- ◆ C90에서 중복된 type qualifier 적용은 잘못된 것

```
const const int i;    /* wrong in C90 */
typedef const int cint;
const cint foo;      /* wrong in C90 */
```

- ◆ `<signal.h>`의 `sig_atomic_t`를 위해 중복된 type qualifier 적용을 무해한 것으로 정의

```
typedef volatile int sig_atomic_t;
volatile sig_atomic_t foo;    /* okay in C99 */
```

## 19. va\_copy() macro

- ◇ C90은 가변인자 제어에 사용되는 **va\_list** 형의 object를 이식성 있는 방법으로 복제할 수 있는 방법 제공하지 않음

```
◇ int foo(int n, ...)
◇ {
◇     int i, sum;
◇     va_list ap, aq;

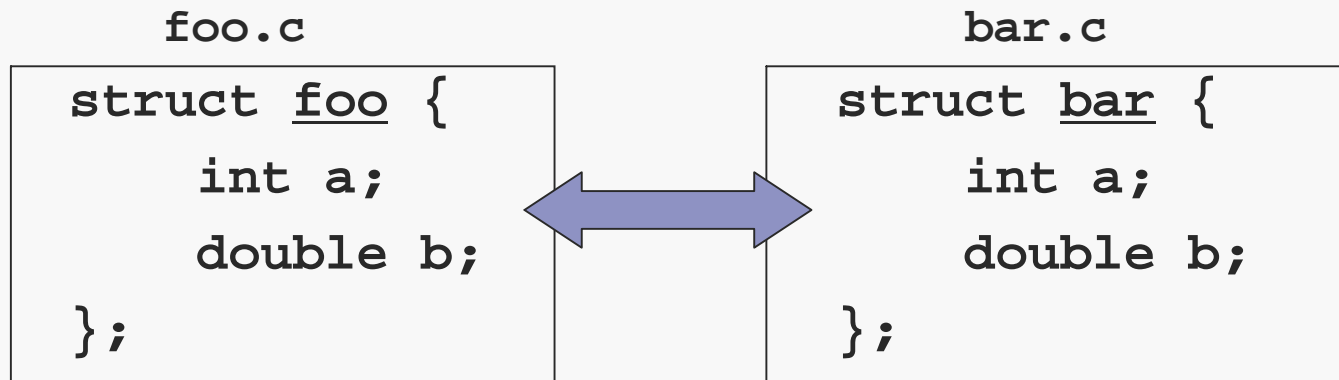
◇     va_start(ap, n);
◇     while ((i = va_arg(ap, int)) >= 0) sum += i;
◇     aq = ap;    /* may not work */
◇     for (i = 0; i < n; i++) sum += va_arg(ap, int);
◇     for (i = 0; i < n; i++) sum -= bar(va_arg(aq, int));
◇     va_end(ap);

◇     return sum;
◇ }
```

## 20. New struct type compatibility rule

---

- ◆ 두 개의 translation unit (소스 파일) 사이에서 호환될 수 있는 구조체, 공용체, 열거형의 규칙에서 tag는 중요하지 않았음



- ◆ C99에서는 tag의 동일성도 요구



# ⟨New features in C99⟩

---

- ◇ restricted character set support via digraphs and `<iso646.h>` (AMD1)
- ◇ wide character library support in `<wchar.h>` and `<wctype.h>` (AMD1)
- ◇ more precise aliasing rules via effective type
- ◇ restricted pointers
- ◇ flexible array member
- ◇ variable length arrays
- ◇ `static` and type qualifiers in parameter array declarators
- ◇ complex (and imaginary) support in `<complex.h>`
- ◇ type generic math macros in `<tgmath.h>`
- ◇ the `long long int` type and library functions

# ⟨New features in C99⟩ (Cont’)

---

- ◇ increased minimum translation limits
- ◇ additional floating-point characteristics in `<float.h>`
- ◇ remove implicit **int**
- ◇ reliable integer division
- ◇ universal character names (`\u` and `\U`)
- ◇ extended identifiers
- ◇ hexadecimal floating-point constants and `%a` and `%A` `printf/scanf` conversion specifiers
- ◇ compound literals
- ◇ designated initializers
- ◇ `//` comments

# ⟨New features in C99⟩ (Cont’)

---

- ◇ extended integer types and library functions in `<inttypes.h>` and `<stdint.h>`
- ◇ remove implicit function declaration
- ◇ preprocessor arithmetic done in `intmax_t/uintmax_t`
- ◇ mixed declarations and code
- ◇ new block scopes for selection and iteration statements
- ◇ integer constant type rules
- ◇ integer promotion rules
- ◇ macros with a variable number of arguments
- ◇ the `vscanf` family of functions in `<stdio.h>` and `<wchar.h>`
- ◇ additional math library functions in `<math.h>`

# ⟨New features in C99⟩ (Cont’)

---

- ◇ floating-point environment access in `<fenv.h>`
- ◇ IEC 60559 (also known as IEC 559 or IEEE arithmetic) support
- ◇ trailing comma allowed in enum declaration
- ◇ `%lf` conversion specifier allowed in `printf`
- ◇ inline functions
- ◇ the `snprintf` family of functions in `<stdio.h>`
- ◇ boolean type in `<stdbool.h>`
- ◇ idempotent type qualifiers
- ◇ empty macro arguments
- ◇ new `struct` type compatibility rules (tag compatibility)

# ⟨New features in C99⟩ (Cont’)

---

- ◇ additional predefined macro names
- ◇ `_Pragma` preprocessing operator
- ◇ standard pragmas
- ◇ `__func__` predefined identifier
- ◇ **`va_copy`** macro
- ◇ additional `strftime` conversion specifiers
- ◇ LIA compatibility annex
- ◇ deprecate `ungetc` at the beginning of a binary file
- ◇ remove deprecation of aliased array parameters
- ◇ conversion of array to pointer not limited to lvalues
- ◇ relaxed constraints on aggregate and union initialization
- ◇ relaxed restrictions on portable header names
- ◇ return without expression not permitted in function that returns a value (and vice versa)

# <C99 compilers>

---

- ◆ Comeau C/C++ compiler + Dinkumware library  
<http://www.comeaucomputing.com>  
<http://www.dinkumware.com>
- ◆ Intel C compiler (?)  
<http://www.intel.com/software/products/compilers>
- ◆ SGI C compiler (for IRIX)  
<http://www.sgi.com/developers/devtools/languages/c.html>
- ◆ gcc (partly)  
<http://gcc.gnu.org>
- ◆ lcc-win32 (partly)  
<http://www.cs.virginia.edu/~lcc-win32>