New Features in C99 (2)

2004.01.18
KLDP 작은 세미나
전웅 (woong@icu.ac.kr)

<목표>

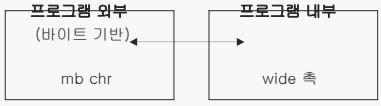
- C99의 새로운 기술을 이해하기 위한 배경 설명 (기존 표준과의 차이 및 도입 배경)
- C99에 새로 추가된 기술 일부에 대한 간략한 소개

<구성>

- 각 기술에 대한 소개 (32개 – 총 53개)
  - 표준에 의해 명시된 대표적 기술 중심 (명시되지 않은 기술도 다수지만 실제 프로그래밍 환경에 주는 영향 없음)
  - 지난 발표에 남겨둔 변화들 – C99 의 주요한 변화들

1. wide character library support in <wchar.h> and <wctype.h>

- 원래는 C90 AMD1 에 추가된 기술
- byte and character conflict
- multibyte character 와 wide character



- 기술의 발전 및 표준화 과정

1. wide character library support in <wchar.h> and <wctype.h> (cont')

- 기본 모델: 가능한 single-byte programming 와 유사하게

```
#include <stdio.h>
#include <wchar.h>

int main(void)
{
    int n = 0;

    while (getwchar() != WEOF)
        n++;
    wprintf(L"%d\n", n);
    return 0;
}
```

1. wide character library support in <wchar.h> and <wctype.h> (cont')

- parallelism 과 improvement
  - (1) parallelism (대부분의 함수)

char	-	wchar_t
int	-	wint_t
isdigit	-	iswdigit
fgetc	-	fgetwc
fprintf	-	fwprintf

  - (2) improvement (문제가 명백한 기존 함수들)
    - sprintf: buffer overrun
    - strtok: internal buffer
  - (3) wide chr func 이 존재하지 않는 경우 (위험성, 불필요)
  - (4) single-byte chr func 이 존재하지 않는 경우 (변환 함수)

1. wide character library support in <wchar.h> and <wctype.h> (cont')

- 과거 <stdlib.h> 에서 독립해 <wchar.h> 와 <wctype.h> 에 모아서 선언
- 여러 가지 가능성이 고려되었음

- generalized multibyte character
  - ISO/IEC 10646 의 문제
  - C 프로그램에서 사용되는 mb chr 의 엄격한 제한과 목적
  - 특별히 file 에 대해서만 허락되는 개념
  - 프로그램 내부에서 다룰 때는 그대로 제한

2. variable-length arrays

- 배열의 크기 명시: 과거에는 integer constant expression 만 허락
- numerical computing 에서의 불만
- runtime expression 으로 완화
- variable length array (VLA) 와 variably modified type 의 개념

```
int (*pvla)[n];
int vla[n];
```

2. variable-length arrays (cont')

```
void foo(int size, int vla[*]);

int bar(void)
{
    const int n = 10;               /* not constant */
    int m = 9;
    int a[n];                                /* VLA */
    int twodim[m][n];
    size_t s = sizeof(a);     /* runtime evaluation */
    /* ... */
}

void foo(int size, int vla[size])
{    /* ... */    }
```

2. variable-length arrays (cont')

- size expression 의 평가 여부에 유의

```
i = 0;
int a1[++i];                          /* evaluated */
int s1 = sizeof(++i);           /* not evaluated */
int s2 = sizeof(int [++i]);           /* evaluated */
int s2 = sizeof(int (*)[++i]);    /* unspecified */
```

3. static and type qualifiers in parameter array declarators

- static

```
void func(int a[static 10], int b[static 10])
{
    for (i = 0; i < 10; i++)
        a[i] += b[i];
}
```

- 형한정어

```
void func(int a[const], int * const b)
```

4. type-generic math macros in <tgmath.h>

- 모델: FORTRAN 의 generic function
  - 따라서, intrinsic 이나 overloading 의 이름 사용하지 않음
- 함수의 데이터형이 인자의 데이터형으로 결정

```
#include <tgmath.h>
float f = SOME_REAL_NUM;
result = sin(f);
```

- 프로그램 porting 시에 유용

5. universal character names (₩u and ₩U)

- ₩u, ₩U 로 시작해 ISO/IEC 10646 에 의해 정의된 문자를 지정하는 UCN 을 명칭, 문자(열) 상수, 주석 등에 사용 가능
- UCN 을 매개로 비표준 문자를 사용하는 프로그램의 의미를 유지하며 이식성 확보

6. extended identifiers

- C90 에서 잘못된 행동을 통한 확장으로 얻은 행동을 정식으로 보장 받음
- (implementation 이 허락한다면) 명칭에 확장 문자 사용 가능

```
typedef int 레코드;
레코드 전체합, 현재;
for (현재 = 0; 현재 < 전체합; 현재++)
```

- 완전한 이식성 보장은 받지 못하지만, UCN 지원을 통한 이식성 확보는 가능

7. hexadecimal floating-point constants and %a and %A printf/scanf conversion specifiers

- binary floating-point 환경 (FLT_RADIX == 2) 에서 significand 를 손쉽게 기술하는 방법

```
0x10ff0100p-8f
   == 00010000111111110000000100000000(2) * 2-8
```

8. compound literals

- compound literal 은 이름 없는 대상체를 기술하는 방법 제공

```
void func(struct foo param);
func((struct foo) { 123, 3.14159 });

void func2(struct foo *p);
func2(&(struct foo) { 123, 3.14159 });

int *p = (int [3]) { 1, 2, 3 };
char *p = (char []) { "modifiable string" };

const int *a1 = (const int []) { 1, 2, 3 };
const int *a2 = (const int []) { 1, 2, 3 };
```

8. compound literals (cont')

- designated initializer, 비상수 초기치와 결합해 다양하게 활용 가능

```
struct foo {
    int i, j, k;
};

void func(struct foo bar);
func((struct foo) { .i = f1(), .k = f2() });
```

8. compound literals (cont')

- 포인터로 가리킬 때는 storage duration 에 유의해야 함

```
int i=0, *p[3];
p[i] = (int [1]) { i++ };
p[i] = (int [1]) { i++ };
p[i] = (int [1]) { i++ };
```

```
for (i = 0; i < 3; i++) {
    p[i] = (int p[1]) { i++ };    /* wrong */
}
```

9. _Pragma preprocessing operator

- **#pragma** 지시자의 단점

```
#define struct_pack(n) #pragma pack(n)
struct_pack(1)            /* doesn't work */
```

```
#if defined(A_COMPILER)     /* A_COMPILER */
#    define pack strict_pack
#else                       /* B_COMPILER */
#    define pack __pack(1)
#endif

#pragma pack              /* does work??? */
```

9. _Pragma preprocessing operator (cont')

- **_Pragma** operator 의 사용

```
#define struct_pack(n) _Pragma("pack("#n")")
struct_pack(1)            /* #pragma pack(1) */
```

```
#if defined(A_COMPILER)
#    define pack "strict_pack"
#else                                /* B_COMPILER */
#    define pack "__pack(1)"
#endif

_Pragma(pack)
         /* #pragma strict_pack or __pack(1) */
```

10. boolean type in <stdbool.h>

- **_Bool** 을 통해서 boolean type 제공
- **_Bool** 은 0, 1 저장 가능한 unsigned integer type
- **<stdbool.h>** 의 **bool**, **true**, **false**: 각각 **_Bool**, 1, 0 과 동일
- scalar type (pointer 포함) 의 **_Bool** 형으로의 변환은 0과의 비교로 결과 결정

10. boolean type in <stdbool.h> (cont')

```
_Bool b, *pb;
b = 0;                  /* b = 0 */
b = -1;                 /* b = 1 */
pb = &b;                /* not null pointer */
b = pb;                 /* 따라서, b = 1; */
b = (_Bool) 0.5         /* (int) 0.5 과 비교 */
```

```
#include <stdbool.h>

bool b1, b2;
if (func1() && !func2())
    b1 = true, b2 = false;
```

11. extended integer types and library functions in <inttypes.h> and <stdint.h>

- 다양한 특성의 정수형을 이식성을 갖추며 손쉽게 사용하도록 허락
- 제공되는 정수형의 형태
  - 정확한 크기를 갖는 정수형
  - 최소한 명시된 크기를 갖는 정수형
  - 최소한 명시된 크기를 갖는 빠른 정수형
  - 대상체 포인터를 변환하기 위한 정수형
  - 가장 큰 크기를 갖는 정수형
- 선택적으로 제공되는 type 과 강제되는 type 으로 구분됨

11. extended integer types and library functions in <inttypes.h> and <stdint.h> (cont')

- 제공되는 정보
  - 제공되는 정수형의 최대값, 최소값
  - 표준이 제공하는 typedef 정수형의 최대값, 최소값
  - 최소한의 크기가 보장되는 정수형의 상수를 만드는 매크로
  - 가장 큰 정수형의 상수를 만드는 매크로
- hosted environment 를 위해 **<inttypes.h>** 이 제공하는 매크로 및 함수들
  - 출력에 사용할 수 있는 매크로
  - 기본 연산 (절대값 등) 과 변환 (문자열 -> 정수형) 에 사용할 수 있는 함수들

12. conversion of array to pointer not limited to lvalues

- 배열의 포인터로의 변환 (decaying)
- C90/C99 의 차이:

```
typedef struct { int x[10]; } foo;
foo func()
{
    foo r;
    r.x[2]=3;
    return r;
}

sizeof(foo().x);                    /* okay in both */
printf("%d\n",foo().x[2]);    /* okay only in C99 */
```

그 외의 기술 (13~15)

- inline functions
  - 강제가 아닌 추천임
- **__func__** predefined identifier
  - 매크로가 아닌 자동으로 정의되는 명칭임
- preprocessor arithmetic done in **intmax_t/uintmax_t**
  - 전처리기 연산에서 환경 검사는 금물
  - cross-implementation 에서 차이 보일 수 있음 (C99 는 실행 환경 특성 따르도록 요구)

그 외의 기술 (16~18)

- the **long long int** type and library functions
  - 상위 호환성 위해 가능하면 확장 정수형이 아닌 표준 long long int 형을 사용해야
- integer constant type rules
  - 새 정수형의 추가와 본격적인 확장 정수형의 지원으로 정수 상수의 데이터형을 결정하는 과정에 미세한 차이 발생
- integer promotion rules
  - 동일한 이유로 정수 진급 규칙에서도 변화 발생

그 외의 기술 (19~20)

- new block scopes for selection and iteration statements

```
for (int i = 0; i < max; i++)
```

- trailing comma allowed in enum declaration
  - code 의 machine generation 을 쉽게 해줌. 집합체형 초기치의 규칙과 균형을 맞추는 변화

```
enum { FOO = 1, BAR, FOOBAR, };
```

그 외의 기술 (21~23)

- additional predefined macro names

```
__STDC_HOSTED__
__STDC_IEC_559__
__STDC_IEC_559_COMPLEX__
__STDC_ISO_10646__
```

- standard pragmas

```
#pragma STDC …
```

- relaxed restrictions on portable header names
  - 문자(영문 대소문자)로 시작
  - 문자와 숫자로 구성
  - 8글자.1글자
  - 대소문자 무시 가능

그 외의 기술 (24~26)

- additional strftime conversion specifiers
  - 주로 ISO 8601 형식을 지원하기 위한 것
- deprecate ungetc at the beginning of a binary file
- the vscanf family of functions in **<stdio.h>** and **<wchar.h>**

그 외의 기술 (27~32)

- additional math library functions in `<math.h>`
- floating-point environment access in `<fenv.h>`
- additional floating-point characteristics in `<float.h>`
- IEC 60559 (also known as IEC 559 or IEEE arithmetic) support
- complex (and imaginary) support in `<complex.h>`
- LIA compatibility annex

질문