_Generic макрос

Теги: _Generic, макрос, перегрузка функции в си

Дженерики

В Си стандарта C11 появились дженерик макросы, которые позволяют создавать функции общего назначения. Дженерик макрос позволяет выбрать функцию в зависимости от типа аргументов, переданных в функцию. Это, в некотором роде, сродни перегрузке методов по типу аргумента во многих языках, однако, это не динамическая диспетчеризация – определение типа аргумента происходит только на этапе компиляции.

Так как это нововведение Си 11, то для работы потребуется новая версия компилятора, которая поддерживает стандарт. Для примера используется gcc версии 5.3.0 с флагом -std=c11. На Windows без проблем ставится пакет MinGW, в состав которого входят все необходимые утилиты.

Рассмотрим на простом примере: функция возвращает переданный аргумент плюс один. Сначала определим все функции, которые могут быть выполнены, со всеми типами аргументов, которые нам необходимы

```
int foo_int(int a) {
    return a + 1;
}
float foo_float(float a) {
    return a + 1.0;
}
char foo_char(char a) {
    return a + 1;
}
```

Далее сам макрос

```
#define foo(X)          \
    _Generic((X),       \
    int: foo_int,       \
    char: foo_char,     \
    float: foo_float,   \
    default: foo_int    \
)(X)
```

Эта система похожа на оператор switch. Вместо foo будет подставлена одна из функций, а далее (X) – вызов с этим аргументом. Дефолтное значение – когда тип определить нельзя. Если тип не определён и нет дефолтного значения, или оно не может быть использовано функцией по умолчанию, то это ошибка. Вот программа

```
#include <stdio.h>

int foo_int(int a) {
    return a + 1;
}
float foo_float(float a) {
    return a + 1.0;
}
char foo_char(char a) {
    return a + 1;
}
```

```
#define foo(X)          \
    _Generic((X),       \
    int: foo_int,       \
    char: foo_char,     \
    float: foo_float,   \
    default: foo_int    \
)(X)

int main() {
    float a = foo(5.0f);
    int b = foo(5);
    char c = foo('a');

    printf("%.3f\n", a);
    printf("%d\n", b);
    printf("%c\n", c);

    return 0;
}
```

Программа выведет

```
6.000
6
b
```

Для функции с двумя аргументами немного сложнее, надо описать все комбинации. В нашем случае функция может принимать аргументы типа float и int, складывать их и возвращать целое

```
#include <stdio.h>

int baz_int_int(int a, int b) {
    return a + b;
}

int baz_int_float(int a, float b) {
    return a + (int) b;
}

int baz_float_int(float a, int b) {
    return (int) a + b;
}

int baz_float_float(float a, float b) {
    return (int) a + (int) b;
}

#define baz_int(X, Y) _Generic((Y),\
    int:   baz_int_int,\
    float: baz_int_float\
)

#define baz_float(X, Y) _Generic((Y),\
    int:   baz_float_int,\
    float: baz_float_float\
)

#define bar(X, Y) _Generic((X),\
    int:   baz_int(X, Y),\
    float: baz_float(X, Y)\
)(X, Y)

int main() {
```

```
    int d = bar(1, 2);
    int e = bar(1.0f, 2.0f);

    printf("%d\n", d);
    printf("%d\n", e);

    return 0;
}
```

Конечно, можно вместо трёх макросов объединить всё в один

```
#include <stdio.h>

int baz_int_int(int a, int b) {
    return a + b;
}

int baz_int_float(int a, float b) {
    return a + (int) b;
}

int baz_float_int(float a, int b) {
    return (int) a + b;
}

int baz_float_float(float a, float b) {
    return (int) a + (int) b;
}

#define bar(X, Y) _Generic((X),\
    int:   _Generic((Y),\
        int:   baz_int_int,\
        float: baz_int_float\
    ),\
    float: _Generic((Y),\
        int:   baz_float_int,\
        float: baz_float_float\
    )\
)(X, Y)

int main() {
    int d = bar(1, 2);
    int e = bar(1.0f, 2.0f);

    printf("%d\n", d);
    printf("%d\n", e);

    return 0;
}
```

Вот другой пример: макрос, который выводит имя типа, переданного в него

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

#define typeof_name(X) _Generic ((X),\
    int:            "signed integer",\
    unsigned:       "unsigned integer",\
    char:           "signed char",\
    unsigned char:  "unsigned char",\
    float:          "float",\
    double:         "double",\
    char*:          "modifiable string",\
```

```
    const char*:    "const string",\
    default:        "unknown"\
)

#define print_type(X) printf("%s\n", typeof_name(X))

int main() {
    print_type('a');
    print_type((int64_t) 32);
    print_type("string");

    return 0;
}
```

Заметьте важные особенности – без явного указания ‘a’ будет рассматриваться как int, а строка как модифицируемая, хотя попытка её изменить приведёт к ошибке. Для точного определениятипа можно его явно привести

```
print_type((unsigned char)'a');
print_type((int64_t) 32);
print_type((const char*)"string");
```

Также тип может быть аргументом макроса

```
#include <stdio.h>

#define is_compatible(x, T) _Generic((x), T:1, default: 0)

int main() {
    int a = is_compatible(3, int);
    int b = is_compatible(3, float);
    printf("a = %d and b = %d", a, b);

    return 0;
}
```