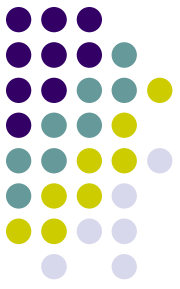


assertions - requirements



```
#include <assert.h>
```

```
assert(expression);
```

- **assert** is a macro-function.
- The expression has to be of logical (bool) type – a claim. That claim represents programmer's assumption. In other words, programmer is asserting that the expression is true.
- Look about assert in C standard.



```
int main() {
    int_fast16_t a, b;
    printf("Enter a whole number between 1 and 10: ");
    scanf("%"SCNdFAST16_T, &a);

    printf("Enter another whole number between 1 and 10: ");
    scanf("%"SCNdFAST16_T, &b);

    some_processing(a, b);
    return 0;
}
```

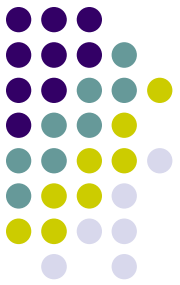


```
int main() {
    int_fast16_t a, b;
    printf("Enter a whole number between 1 and 10: ");
    scanf("%"SCNdFAST16_T, &a);
    while (a < 1 || a > 10) {
        printf("Error. Between 1 and 10! Again:");
        scanf("%"SCNdFAST16_T, &a);
    }
    printf("Enter another whole number between 1 and 10: ");
    scanf("%"SCNdFAST16_T, &b);
    while (b < 1 || b > 10) {
        printf("Error. Between 1 and 10! Again:");
        scanf("%"SCNdFAST16_T, &b);
    }

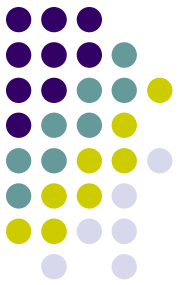
    some_processing(a, b);
    return 0;
}
```



```
// Function accepts two integers between 1 and 10
int_fast16_t foo(int_fast16_t x, int_fast16_t y)
{
    int_fast16_t p = x * x * x * x;
    int_fast16_t q = y * y * y * y;
    return p + q;
}
```



```
// Function accepts two integers between 1 and 10
int_fast16_t foo(int_fast16_t x, int_fast16_t y)
{
    while (x < 1 || x > 10) {
        printf("Error. Between 1 and 10! Again:");
        scanf("%"SCNdFAST16_T, &x);
    }
    while (y < 1 || y > 10) {
        printf("Error. Between 1 and 10! Again:");
        scanf("%"SCNdFAST16_T, &y);
    }
    int_fast16_t p = x * x * x * x;
    int_fast16_t q = y * y * y * y;
    return p + q;
}
```



```
// Function accepts two integers between 1 and 10
int_fast16_t foo(int_fast16_t x, int_fast16_t y)
{
    if (x < 1 || x > 10 || y < 1 || y > 10)
    {
        // ???
    }

    int_fast16_t p = x * x * x * x;
    int_fast16_t q = y * y * y * y;
    return p + q;
}
```



```
// Function accepts two integers between 1 and 10
int_fast16_t foo(int_fast16_t x, int_fast16_t y)
{
    if (x >= 1 && x <= 10 && y >= 1 && y <= 10)
    {
        int_fast16_t p = x * x * x * x;
        int_fast16_t q = y * y * y * y;
        return p + q;
    }
    else
    {
        // ???
    }
}
```



```
// Function accepts two integers between 1 and 10
int_fast16_t foo(int_fast16_t x, int_fast16_t y)
{
    if (x >= 1 && x <= 10 && y >= 1 && y <= 10)
    {
        int_fast16_t p = x * x * x * x;
        int_fast16_t q = y * y * y * y;
        return p + q;
    }
    else
    {
        printf("Some error report");
        exit(1);
    }
}
```




```
// Function accepts two integers between 1 and 10
int_fast16_t foo(int_fast16_t x, int_fast16_t y)
{
    if !(x >= 1 && x <= 10 && y >= 1 && y <= 10)
    {
        printf("Some error report");
        exit(1);
    }

    int_fast16_t p = x * x * x * x;
    int_fast16_t q = y * y * y * y;
    return p + q;
}
```



```
#include <assert.h>

// Function accepts two integers between 1 and 10
int_fast16_t foo(int_fast16_t x, int_fast16_t y)
{
    assert(x >= 1 && x <= 10 && y >= 1 && y <= 10);

    int_fast16_t p = x * x * x * x;
    int_fast16_t q = y * y * y * y;
    return p + q;
}
```



```
#include <assert.h>
```

```
// Function accepts two integers between 1 and 10
```

```
int_fast16_t foo(int_fast16_t x, int_fast16_t y)
{
```

```
    assert(x >= 1 && x <= 10);
```

```
    assert(y >= 1 && y <= 10);
```

```
    int_fast16_t p = x * x * x * x;
```

```
    int_fast16_t q = y * y * y * y;
```

```
    return p + q;
```

```
}
```

assert vs error handling



- Assertions should be used to “catch” logically impossible situations and discover programming errors, so if the “impossible” occurs, then something fundamental is clearly wrong.
- Job of error handling is to manage error conditions that **are possible**, although some may be extremely unlikely to occur in practice.
- Understand the distinction and do not use one in place of another.

Asserts are useful during development



- Assertion checking is useful during development, but since they should always be true, in the release version of the program we usually do not want them to be checked.
- That is why asserts can be removed at compile time with defining `-DNDEBUG`
- Assertions are most often disabled in the **release** version - C preprocessor completely removes assertions at compile time.



static assert

- Statically checkable claim, i.e. the claims validity can be check during compile time.
- **`_Static_assert`**
- Roughly similar to `#error` with some checks
- `_Static_assert(_Alignof(char) == 1, "alignment of char must be 1");`
- It has to have the error message text.