Error Handling

Program Design (II)

2022 Spring

Fu-Yin Cherng
Dept. CSIE, National Chung Cheng University

All materials are provided by C Programming: A Modern Approach by K. N. King, 2nd edition, 2008, W. W. Norton & Company.

Why we need to learn Error Handling?

- Although it’s common to have an error when we are developing programs
- Commercial programs need to be able to **recover** from errors instead of crashing!
- Programmers need to **anticipate errors** that might happen during the program's execution and try to recover from them.

Outline

- The `<assert.h>` Header: Diagnostics
- The `<errno.h>` Header: Error

Error detection and handling are **not** one of C’s strengths. Newer languages like C++ and Java have **exception handling** features that make this work easier.

The <`assert.h`> Header: Diagnostics

- `assert` define in the `<assert.h>` and can help us to **monitor** a program behavior and detect possible problems in advance
- `assert` is actually a macro but it’s designed to be used like a function.

```
void assert (expressions);
```

The `<assert.h>` Header: Diagnostics

- If the expressions return a **non zero** value: does nothing
- If the expressions return a **zero** value: writes a **message** to `stderr` and calls the `abort` function to terminate program
    - The C library function `void abort(void)` **abort** (中止) the program execution and comes out directly from the place of the call.

```
void assert (expressions);
```

The `<assert.h>` Header: Diagnostics

- For example, a program declares an array `a` of length 10 and assign 0 to element `i`
- We concern that the program will fail if `i` isn’t between 0 and 9.

```
int a[10];
int i;
scanf("%d", &i);
a[i] = 0;
```

The `<assert.h>` Header: Diagnostics

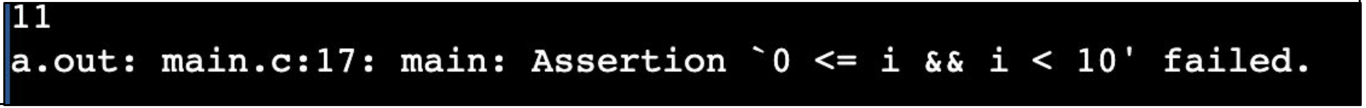
- We can use `assert` to **check** this condition first.

```
int a[10];
int i;
scanf("%d", &i);
assert(0 <= i && i < 10);
a[i] = 0;
```

The <assert.h> Header: Diagnostics

- If i's value < 0 or >= 10, the program will **terminate** after displaying a message like the following one (GCC compiler)
- Exact form of the message produced by assert may **vary** from different **compilers**

```
int a[10];
int i;
scanf("%d", &i);
assert(0 <= i && i < 10);
a[i] = 0;
```



```
11
a.out: main.c:17: main: Assertion `0 <= i && i < 10' failed.
```

The `<assert.h>` Header: Diagnostics

- `assert` will slightly increases the running time of a program because of the extra check it performs.
- This small extra time may be unacceptable if the running time is not critical.
- Hence, many programmers use `assert` during testing and then **disable** it when the program is finished

The `<assert.h>` Header: Diagnostics

- We can eaily disable `assert` by adding the definition of the **macro** `NDEBUG` **before** including the `<assert.h>` header
- The value of `NDEBUG` doesn’t matter, just need that fact that it’s defined.

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

The <`errno.h`> Header: Errors

- Some functions in the standard library indicate the failure by storing an **error code** (positive integer) in `erron`.
- `erron:` a `int` **variable** declared in `<errno.h>`
- Many functions in `<math.h>` rely on `errno`
- Let's see the following example to see how to use `errno`

The <errno.h> Header: Errors

- For example, **check** whether call of `sqrt` (square root) function has failed.
- It’s important to **store zero** in `errno` **before** calling the function
- since the prior libaray functions never clear `errno`

```
#include <errno.h>
#include <math.h>
…
errno = 0;
double x, y;
y = sqrt(x);
if(errno != 0){
    fprintf(stderr, "sqrt error");
    exit(1);
}
```

The `<errno.h>` Header: Errors

- The value stored in `errno` when an erro occurs is **often either** `EDOM` or `ERANGE`
  - both are macros defined in `<errno.h>`
- They represent two kinds of possible errors when a math functions is called.

The `<errno.h>` Header: Errors

- Domain errors (`EDOM`)
  - Happen when the arugument passed to a function is **outside** the function's **domain**
  - For example, passing a negative number to `sqrt` causes a domain error
- Range errors (`ERANGE`)
  - Happen when a function's **return value** is **too large** to be represented in the function's return type
  - For example, passing 10000 to the `exp` function usually causes a range error, becuase e^{10000} is too large to represent as a `double` on most computers
- We can compare `errno` to `EDOM` and `ERANGE` to determine which error occurred

The `perror` and `strerror` Functions

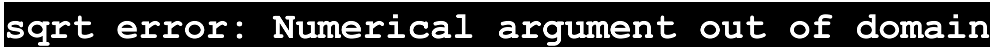
- Related to the `errno` variable, although they are not belongs to `<errno.h>`

```
void perror(const char *s); //from <stdio.h>
char *strerror(int errnum); //from <string.h>
```

The perror Functions

- When a function stores a nonzero value in errno, we may want to **display** a **message** that indicates the meaning of the error.
- perror prints the following message decided by the value of errno
- perror writes to the stderr

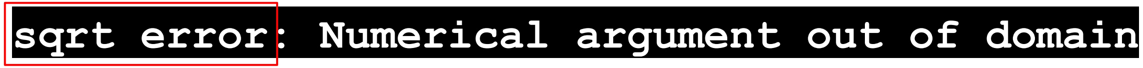
```
…
errno = 0;
y = sqrt(x); //if x = -1; cause domain errors
if(errno != 0){
    perror("sqrt error");
    exit(1);
}
```



```
sqrt error: Numerical argument out of domain
```

The `perror` Functions

```
void perror(const char *s);
```



```
sqrt error: Numerical argument out of domain
```

```
…
errno = 0;
y = sqrt(x);
if(errno != 0){
    perror("sqrt error");
    exit(1);
}
```

The `perror` Functions

- The error message that perror displays after `sqrt` error is **implmenetation-defined**
- Usually,
  - `EDOM` error produces: **`Numerical argument out of domain`**
  - `ERANGE` error produces: **`Numerical result out of range`**

The `strerror` Functions

- When passed an error code, `strerror` returns a **pointer** to a **string** describing the error. For example,

```
char *strerror(int errnum); //from <string.h>
```

```
…
puts(strerror(EDOM));  Numerical argument out of domain
```

The `strerror` Functions

- The argument to `strerror` is **usually** one of the **values** of `errno`, but `strerror` **will return a string for any integer** passed to it
- `strerror` is closely related to the `perror` function.
- `perror` displays the same message as the `strerror` returns

slido

What is the output of your computer for the statement: puts(strerror(1)); ?

ⓘ Start presenting to display the poll results on this slide.

Summary

- The `<assert.h>` Header: Diagnostics
- The `<errno.h>` Header: Error

```
void assert (expressions);

void perror(const char *s); //from <stdio.h>

char *strerror(int errnum); //from <string.h>
```