Виконала: студентка групи комп. мат. 1, Войцехівська Тетяна Віталіївна
Проект: 10. Змінні оточення.

Бібліотеки для роботи з системним оточенням на Ci/Ci++

План:

- Системне оточення(середовище)
- Бібліотека <env> на Ci
- Бібліотека filesystem з Ci++17

Системне оточення

Набір усіх змінних середовища, які мають значення, разом відомі як середовище (environment). А що таке змінні оточення(environment variables)?

Змінна оточення (середовища) - це змінна, яка буде доступна для всіх програм та застосувань. Ми можемо отримати доступ до цих змінних з будь-якої точки програми, не оголошуючи і не ініціалізуючи у програмі. За допомогою даних змінних можна передавати до програми додаткові специфічні параметри, які можуть використовуватися кількома програмами одночасно і вони характеризують "середовище", в якому виконується програма.

Наприклад, якщо Ви розробляєте якусь систему, що складається з декількох програм, що працюють незалежно, і, наприклад, Ви хочете повідомити всім програмам робочий каталог, тимчасову директорію або якийсь інший специфічний параметр. Тоді змінні оточення це єдиний розумний метод, адже передавати такі параметри через командний рядок безглуздо.

Як приклад, Вам потрібно, щоб результат коду завжди записувався у файл, розміщений в системній директорії. Для такого випадку доцільно буде використати змінну оточення %SystemRoot%, що вказує на шлях до системної директорії

Бібліотека <env> на Ci

Цей інструмент визначений набором функцій для роботи зі змінними оточення (environment variables): getenv, secure_getenv , putenv, setenv, unsetenv, clearenv.

- Як почати використовувати бібліотеку?

Для успішного використання функцій для роботи зі змінними оточення (без ворнінгів) потрібно підключати <stdlib.h>. Але на деяких компіляторах це обов'язкова умова.

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

- Функція char * **getenv** (const char *name)

Ця функція повертає С-рядок(що закінчується нульовим символом), який є значенням name (С-рядок) змінної середовища. Ви не повинні змінювати цей рядок. У деяких не-Unix системах, які не використовують бібліотеку GNU C, вона може бути перезаписана наступними викликами getenv (але не будь-якою іншою функцією бібліотеки). Якщо змінна середовища не знайдена, то повертається NULL pointer(покажчик).

Приклад використання getenv():

```
char * my_env = getenv("SystemRoot");
if(my_env)
    printf("SD:  %s \n", my_env);
else
    printf("Not found SystemRoot\n");
```

Результат:

```
SD:  C:\Windows
```

Як бачимо, отримали правильний шлях до системної директорії.

- Функція char * **secure_getenv** (const char *name)

Ця функція схожа на getenv, але вона повертає NULL pointer(покажчик), якщо середовище є ненадійним. Це трапляється, коли програмний файл має SUID або SGID (права доступу Unix, що дозволяють користувачам запускати виконуваний файл із правами файлової системи власника або групи виконуваного файлу відповідно та змінювати поведінку в каталогах). Бібліотеки загального призначення завжди повинні віддавати перевагу цій функції над getenv, щоб уникнути вразливості, якщо на бібліотеку посилається програма SUID/SGID.

- Функція int **putenv** (const char *varname)

Функція встановлює значення змінної середовища шляхом зміни існуючої змінної або створення нової. Параметр varname вказує на рядок у формі var=x, де x — нове значення для змінної середовища var.

Назва не може містити пробіл або символ рівності (=).
Наприклад, цей приклад недійсний через пробіл між PATH та NAME.

```
PATH NAME=/lib/user
```

Подібним чином наступний приклад не є дійсним через символ рівності між PATH та NAME.

```
PATH=NAME=/lib/user
```

Тому що система інтерпретує всі символи після першого символу рівності “=” як значення змінної середовища.

Putenv() повертає 0 у разі успіху. Якщо putenv() виходить з ладу, повертається -1, а errno вказує на помилку.

Відмінність від функції setenv полягає в тому, що в середовище поміщається конкретний рядок (string), заданий як рядок параметрів. Якщо користувач змінить рядок (string) після виклику putenv, це автоматично відобразиться в середовищі. Це також вимагає, щоб рядок (string) не був автоматичною змінною, область видимості якої залишається перед видаленням змінної з середовища. Те саме стосується, звичайно, динамічно виділених змінних, які звільняються пізніше.

Ця функція є частиною розширеного інтерфейсу Unix. Ви повинні визначити _XOPEN_SOURCE перед додаванням будь-якого хедеру.

Приклад використання putenv():

```c
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
   char *pathvar;

   if (-1 == putenv("PATH=/:/home/userid")) {
      printf("putenv failed \n");
      return EXIT_FAILURE;
   }

   pathvar = getenv("PATH");
   printf("The current path is: %s\n", pathvar);
   return 0;
}
```

Результат:

```
The current path is: /:/home/userid
```

- Функція int **setenv** (const char *name, const char *value, int replace) ¶

Додає, змінює та видаляє змінні середовища. Запис із ім’ям name замінюється на value « name = value » (навіть якщо value порожній рядок). NULL pointer(покажчик) для параметра value є неприпустимим.

Якщо середовище вже містить запис із ключем name, параметр replace керує дією. Якщо replace дорівнює нулю, то нічого не відбувається. В іншому випадку старий запис буде замінено новим.

Зауважте, що ви не можете повністю видалити запис за допомогою цієї функції.

Якщо функція успішна, вона повертає 0. В іншому випадку середовище не змінюється, повертається значення -1 і встановлюється errno, що вказує на помилку.

Ця функція спочатку була частиною бібліотеки BSD, але тепер є частиною стандарту Unix.

Приклад використання setenv():

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
   setenv ("PYTHON", "Something", 50);
   printf("PYTHON = %s\n", getenv("PYTHON"));
}
```

Результат:

```
PYTHON = Something
```

- Функція int **unsetenv** (const char *name)

За допомогою цієї функції можна повністю видалити запис із середовища. Якщо середовище містить запис із ключем name, весь цей запис буде видалено. Виклик цієї функції еквівалентний виклику putenv(), коли частина рядка value порожня.

Функція повертає -1, якщо name є NULL pointer(покажчик) , вказує на порожній рядок або вказує на рядок, що містить символ “=”. Він повертає 0, якщо виклик вдався.

Ця функція спочатку була частиною бібліотеки BSD, але тепер є частиною стандарту Unix. Проте версія BSD не мала значення, що повертається.

Приклад використання unsetenv():

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
   setenv ("PYTHON", "Something", 50);
   printf("PYTHON = %s\n", getenv("PYTHON"));

   unsetenv("PYTHON");
   printf("PYTHON = %s\n", getenv("PYTHON"));
}
```

Результат:

```
PYTHON = Something
PYTHON = (null)
```

- Функція int **clearenv** (void)

Функція clearenv() видаляє всі записи з середовища. За допомогою putenv і setenv нові записи можна додати пізніше.

Якщо функція успішна, вона повертає 0. В іншому випадку повертається значення відмінне від нуля.

Бібліотека filesystem з Сі++17

Одним із найпомітніших нововведень, що з'явилися в C++17, стала бібліотека <filesystem>. Ця бібліотека, як і багато інших важливих особливостей сучасного C++, була запозичена з проекту Boost. У 2015 р. вона була включена до технічної специфікації стандарту для збору відгуків і, нарешті, увійшла до складу стандарту C++17 з деякими змінами, внесеними за результатами відгуків.

- Основні елементи бібліотеки

1. Клас Path: Клас Path — це нова функція C++17, яка забезпечує уніфікований інтерфейс для керування шляхами у файловій системі. Він надає різні функції, такі як побудова з рядка, нормалізація, порівняння та функції запиту, такі як file_size() і exists().

2. Категорія помилок файлової системи: ця нова функція надає категорію помилок для помилок файлової системи. Він надає спосіб класифікувати помилки для кращої обробки операцій файлової системи.

3. Ітератор каталогу (directory_iterator): ця нова функція надає ітератор для перегляду вмісту каталогу. Він забезпечує зручний спосіб виконання операцій над усіма файлами та підкаталогами в каталозі.

4. Рекурсивний ітератор каталогу(recursive_directory_iterator): ця нова функція надає ітератор для перегляду вмісту каталогу та його підкаталогів. Він забезпечує зручний спосіб виконання операцій над усіма файлами та підкаталогами в каталозі та його підкаталогах.

5. Операції копіювання та переміщення: ця нова функція надає функції для копіювання та переміщення файлів і каталогів. Вона забезпечує зручний спосіб виконання операцій у файловій системі.

- Основні функції бібліотеки

1. fs::copy() – Ця функція використовується для копіювання вмісту файлу з одного місця в інше.

2. fs::rename() – Ця функція використовується для перейменування файлу або каталогу.

3. fs::remove() – Ця функція використовується для видалення файлу або каталогу.

4. fs::exists() – Ця функція використовується, щоб перевірити, чи існує файл або каталог.

5. fs::create_directory() – Ця функція використовується для створення нового каталогу.

6. fs::create_directories() – Ця функція використовується для створення кількох каталогів одночасно.

7. fs::last_write_time() – ця функція використовується для визначення часу останнього запису до файлу чи каталогу.

8. fs::file_size() – Ця функція використовується для отримання розміру файлу.

9. fs::directory_iterator() – Ця функція використовується для перебору вмісту каталогу.

10. fs::space() – Ця функція використовується для отримання інформації про доступний дисковий простір.

- Підсумок

Бібліотека файлової системи C++17 надає простий у використанні та ефективний набір функцій для роботи з файлами та каталогами. Він має можливість маніпулювати файлами та каталогами різноманітними способами, наприклад створювати та видаляти їх, переміщувати та переглядати їхній вміст. Бібліотека також надає функції доступу до атрибутів файлів і керування змінними середовища, пов’язаними з файлами та каталогами. Бібліотека надає повний набір функцій і класів для роботи з файлами та каталогами, що робить її безцінним інструментом для будь-якого розробника C++.