Семестр 1. Лекция 6. Обзор libc: stdio. C++: ссылки, new, delete. Введение в ООП.

Евгений Линский

13 Октября 2017

Стандартная библиотека (libc)

1. `cplusplus.com`
2. `cppreference.com`
3. MSDN

Обзор:

- stdio.h — ввод/вывод (файл, клавиатура, экран)
- *stdlib.h* — работа с памятью, алгоритмы
- *string.h* — работа со строками и массивами
- math.h — математические функции
- *time.h* — время
- *assert.h*, *stdint.h*

stdlib

```
// array to integer
int res1 = atoi("abc"); // undef. behavior (usually 0)
int res2 = atoi("0");
```

```
// endPtr -- первый символ, на котором сломалось
long strtol(char *buffer, char **endPtr, int base);
char *end; char *ptr = "25a";
int N = strtol(ptr, &end, 10);
if (ptr == end) {

}
```

1. А еще можно с помощью?

stdlib

```
// array to integer
int res1 = atoi("abc"); // undef. behavior (usually 0)
int res2 = atoi("0");
```

```
// endPtr -- первый символ, на котором сломалось
long strtol(char *buffer, char **endPtr, int base);
char *end; char *ptr = "25a";
int N = strtol(ptr, &end, 10);
if (ptr == end) {

}
```

1. А еще можно с помощью?
2. sscanf

stdlib

time(NULL) — текущее время в секундах.

```
srand (time(NULL));
int r1 = rand();
srand (time(NULL));
int r2 = rand();
```

1. Что не так?

stdlib

time(NULL) — текущее время в секундах.

```
srand (time(NULL));
int r1 = rand();
srand (time(NULL));
int r2 = rand();
```

1. Что не так?
2. $r_i = rand(r_{i-1})$, $r_0 = srand()$; (псевдослучайная последовательность) srand и rand связаны через глобальную переменную.

stdlib

time(NULL) — текущее время в секундах.

```
srand (time(NULL));
int r1 = rand();
srand (time(NULL));
int r2 = rand();
```

1. Что не так?
2. $r_i = rand(r_{i-1})$, $r_0 = srand()$; (псевдослучайная последовательность) srand и rand связаны через глобальную переменную.
3. *time(NULL)* возвращается текущее время в секундах; выполнение программы займет меньше секунды.

stdlib

time(NULL) — текущее время в секундах.

```
srand (time(NULL));
int r1 = rand();
srand (time(NULL));
int r2 = rand();
```

1. Что не так?
2. $r_i = rand(r_{i-1})$, $r_0 = srand()$; (псевдослучайная последовательность) srand и rand связаны через глобальную переменную.
3. *time(NULL)* возвращается текущее время в секундах; выполнение программы займет меньше секунды.
4. Для отладки можно сделать *srand(3)* и всегда получать одну и ту же последовательсность.

stdlib

```
void qsort (void* base, size_t num, size_t size,
            int (*compar)(const void*,const void*));

void* bsearch (const void* key, const void* base,
   size_t num, size_t size,
   int (*compar)(const void*,const void*));
```

```
ret = system("ls -l"); // return exit code
```

```
int* m = (int*) malloc(10000000);
if (m == NULL) {
  fpritnf(stderr, "..."); //y stderr нет буфера
  exit( EXIT_FAILURE); //clean up:  flushin buffers, etc
  // void abort (void); //without clean up
}
```

cstring

```
// не проверяют размер dst
// оптимизированы используют “широкие” команды sse, avx etc
void* memcpy ( void* dst, const void* src, size_t num );
char* strcpy ( char* dst, const char* src );
```

Разбить строку на токены по разделителям.

```
//char * strtok ( char * str, const char * delimiters );
char str[] ="This,a sample string.";
char* pch = strtok (str," ,");
while (pch != NULL) {
  printf ("%s\n",pch);
  pch = strtok (NULL, " ,.-");
}
```

1. Как работает?

cstring

```
// не проверяют размер dst
// оптимизированы используют “широкие” команды sse, avx etc
void* memcpy ( void* dst, const void* src, size_t num );
char* strcpy ( char* dst, const char* src );
```

Разбить строку на токены по разделителям.

```
//char * strtok ( char * str, const char * delimiters );
char str[] ="This,a sample string.";
char* pch = strtok (str," ,");
while (pch != NULL) {
  printf ("%s\n",pch);
  pch = strtok (NULL, " ,.-");
}
```

1. Как работает?
2. Внутри strtok есть статическая переменная, которая хранит место в строке, откуда надо начать при следующем вызове (если str == NULL). Если str не NULL, то начать надо с str.

cstring

```
// не проверяют размер dst
// оптимизированы используют “широкие” команды sse, avx etc
void* memcpy ( void* dst, const void* src, size_t num );
char* strcpy ( char* dst, const char* src );
```

Разбить строку на токены по разделителям.

```
//char * strtok ( char * str, const char * delimiters );
char str[] ="This,a sample string.";
char* pch = strtok (str," ,");
while (pch != NULL) {
  printf ("%s\n",pch);
  pch = strtok (NULL, " ,.-");
}
```

1. Как работает?
2. Внутри strtok есть статическая переменная, которая хранит место в строке, откуда надо начать при следующем вызове (если str == NULL). Если str не NULL, то начать надо с str.
3. strtok разрушает строку, расставляя в ней 0, чтобы printf “знал”, где остановиться при выводе очередного токена.

time

time — текущее время в секундах с 1970 года.

```
time_t t1 = time (NULL);
f();
time_t t2 = time (NULL);
time_t duration = t2 - t1;
```

Секунды слишком долго!

```
clock_t t1 = clock();
f();
clock_t t2 = clock();
time_t duration = (t2 - t1) / CLOCKS_PER_SEC;
```

clock returns the **processor** time consumed by the program.

1. Что не так с clock в многопоточных программах?

time

time — текущее время в секундах с 1970 года.

```
time_t t1 = time (NULL);
f();
time_t t2 = time (NULL);
time_t duration = t2 - t1;
```

Секунды слишком долго!

```
clock_t t1 = clock();
f();
clock_t t2 = clock();
time_t duration = (t2 - t1) / CLOCKS_PER_SEC;
```

clock returns the **processor** time consumed by the program.

1. Что не так с clock в многопоточных программах?
2. 2 потока, 2 core –> clock насчитает в два раза больше чем time.

assert

```
//Гарантировано выявим ошибки программиста на стадии отладки
void print_array(int* a, size_t n) {
  assert (a != NULL); // если условие не выполнено, то abort
  ...
}
// для релиза (а не DEBUG)
gcc -DNDEBUG a.cpp // в этом случае assert компилируется в ;
// все равно что #define NDEBUG
```

1. Как реализовано?

assert

```
//Гарантировано выявим ошибки программиста на стадии отладки
void print_array(int* a, size_t n) {
  assert (a != NULL); // если условие не выполнено, то abort
  ...
}
// для релиза (а не DEBUG)
gcc -DNDEBUG a.cpp // в этом случае assert компилируется в ;
// все равно что #define NDEBUG
```

1. Как реализовано?
2. #ifdef NDEBUG ; #else if(...) else abort() ... #endif

assert, stdint

```
FILE* f = fopen(...);
assert(f != NULL);
```

1. Хорошая идея?

stdint.h

```
int16_t a;
uint64_t b;
```

Может не компилироваться, если на платформе нет типа!

assert, stdint

```
FILE* f = fopen(...);
assert(f != NULL);
```

1. Хорошая идея?
2. Нет. Ошибка не выявить на стадии отладки, а можент произойсти всегда! А при NDEBUG мы отключим проверку.

stdint.h

```
int16_t a;
uint64_t b;
```

Может не компилироваться, если на платформе нет типа!

1980 Bell Labs, Бьёрн Страуструп

```
void swap(int* a, int *b) {
  int temp = *a;
  *b = *a;
  *a =temp
}
int c = 3; int d = 5;
swap(&c, &d);
```

Ссылки:

```
void swap(int& a, int& b) {
  int temp = a;
  b = a;
  a =temp
}
int c = 3; int d = 5;
swap(c, d);
```

Почему в C++ обязательно наличие объявление функции в файле, где она вызывается?

Приведение типов указателей. new и delete.

Более строгие правила по сравнению с C:

```
int *pi = ...;
char *pc = ...;
pi = (int*) pc;

void *pv = pi;
pc = (char*) pv;
```

new/delete — операторы (транслирует компилятор, а не линкуется из библиотеки), а не функции как malloc.
Поэтому они “знают” про sizeof, возвращают “нужный” тип указателя, а также ... (обсудим в ООП).

```
int *pi = new int [1000];
....
delete [] pi;
```

Нельзя удалять с помощью delete то, что создано malloc! (new/free тоже)
У них разный формат представления данных в памяти (“НО ЭТО НЕ ТОЧНО!”).

Три идеи:

1. Инкапсуляции.
   - языковая конструкция, позволяющая связать переменные с функциями, предназначенными для обработки этих данных (получится новый тип данных — класс).
   - механизм языка, позволяющий ограничить доступ одних компонентов программы к другим;
2. Наследование
   - один класс может наследовать (“копировать себе”) данные и функциональность некоторого существующего класса, способствуя повторному использованию компонентов программного обеспечения.
3. Полиморфизм
   - называется способность функции обрабатывать данные разных типов (классов)

Исторический взгляд

Повышение уровня абстракции:

1. Ассемблер (load t1, 45; store 46, t2)
2. Язык высокого уровня (for; if; int a)
3. Функции (void f() { for; if; int a })
4. Структуры (struct point_s { int x; int y;};)
5. Классы (переменные и функции, которые они обрабатывают)

```
class MyArray {
  int* array;
  size_t size;
  void insert(size_t index, int value);
};
```

Лингвистический взгляд

C:

1. Покрась дом красным
2. Передвинь дом вправо на три метра

C++:

1. Дом, покрасься красным
2. Дом, передвинься вправо на три метра

Практический взгляд

Две роли: программист-автор компонента и программист-пользователь компонента.

- **Инкапсуляции**. Компонент “Микроволновка”. Для нас черный ящик. Механизм и таймер упрятан внуть и закрыт кожухом, до него не добраться. На внешней панели две простые “крутилки”: мощность и время.
- **Наследование**. Новый компонент не с нуля, а на основе того, что уже есть. Температура в автомобиле:
  - Печка только греет.
  - Добавили к печке “охлаждатель” получили кондиционер.
  - Добавили к кондиционеру термометр и блок управления климат-контроль.
- **Полиморфизм**. В 1900 договорились об интерфейсе между эл. сетью и устройством (в стене две дырки, у устройства два штырька). В 1950 сделали новое устройство — компьютер. Его можно включать в эл. сеть, поскольку он выполняет требования интерфейса.

Черный ящик для динамического массива

```
int *array = new int[size];
array[i] = ...
...
delete[] array;
```

Задачи программиста:

- не забыть создать
- не выйти случайно за границы
- не перепутать размер
- не забыть удалить

Черный ящик для динамического массива

array.h

```
class Array {
private:
    size_t size;
    int *data;
public:
    Array( int size); // конструктор
    ~Array(); // деструктор
};
```

array.cpp

```
#include "array.h"
Array :: Array(size_t s) {
  size = s;
  data = new int [size];
}

Array :: ~Array() {
  delete[] data;
}
```

Черный ящик для динамического массива

```
#include "array.h"
main() {
   Array a(100); // вызов конструктора Array(...)
   a.data[34] = 3434; //compilation error
} //вызов деструктора  Array()
```