Язык C++

Стандартная библиотека Си

История

- Язык программирования Си до стандартизации не обеспечивал встроенной функциональности, как, например, операции ввода-вывода
- В 1983 году Американский национальный институт стандартов (ANSI) сформировал комитет для принятия стандарта языка Си, известный как «ANSI Си»
- Эта работа вылилась в создание так называемого стандарта C89 в 1989. Часть итогового стандарта была набором библиотек, названная **Стандартная библиотека ANSI Си**.
- Последующие версии стандарта языка Си добавляли некоторые новые и наиболее полезные заголовочные файлы в библиотеку. Поддержка этих новых расширений зависела от реализации.

Задачи библиотеки

- основной набор математических функций
- обработка строк
- конвертация типов
- файловый и консольный ввод-вывод

Структура

- Стандартная библиотека ANSI Си состоит из 29 заголовочных файлов
- Каждый из которых можно подключать к программному проекту при помощи одной директивы
- Каждый заголовочный файл содержит объявления одной или более функций, определения типов данных и макросы.

Список заголовочных файлов

Имя	Описание
<assert.h>	Содержит макрос утверждений, используемый для обнаружения логических и некоторых других типов ошибок в отлаживаемой версии программы
<complex.h>	Набор функций для работы с комплексными числами.
<ctype.h>	Содержит функции, используемые для классификации символов по их типам или для конвертации между верхним и нижним регистрами независимо от используемой кодировки
<errno.h>	Для проверки кодов ошибок, возвращаемых библиотечными функциями.
<fenv.h>	Для управления средой, использующей числа с плавающей запятой.
<float.h>	Содержит заранее определенные константы, описывающие специфику реализации свойств библиотеки для работы с числами с плавающей запятой
<inttypes.h>	Для точной конвертации целых типов.

Список заголовочных файлов

Имя файла	Описание
<iso646.h>	Для программирования в кодировке ISO 646.
<limits.h>	Содержит заранее заданные константы, определяющие специфику реализации свойств целых типов, как, например, область допустимых значений (_MIN, _MAX).
<locale.h>	Для setlocale() и связанных констант. Используется для выбора соответствующего языка.
<math.h>	Для вычисления основных математических функций
<setjmp.h>	Объявляет макросы setjmp и longjmp, используемые для переходов
<signal.h>	Для управления различными исключительными условиями
<stdarg.h>	Для доступа к различному числу аргументов, переданных функциям.

Список заголовочных файлов

Имя файла	Описание
<stdbool.h>	Для булевых типов данных.
<stdint.h>	Для определения различных типов целых чисел.
<stddef.h>	Для определения нескольких полезных типов и макросов.
<stdio.h>	Реализует основные возможности ввода и вывода в языке Си.
<stdlib.h>	Для выполнения множества операций, включая конвертацию, генерацию псевдослучайных чисел, выделение памяти, контроль процессов, окружения, сигналов, поиска и сортировки.
<string.h>	Для работы с различными видами строк.
<tgmath.h>	Для типовых математических функций.

Список заголовочных файлов

Имя файла	Описание
<time.h>	Для конвертации между различными форматами времени и даты.
<wchar.h>	Unicode
<wctype.h>	Unicode

Stdio.h.

- Работа с файлами
- Форматированный ввод-вывод
- Вывод ошибок

printf

int printf(const char *format [, argument]...);

Return Value.

Returns the number of characters printed, or a negative value if an error occurs.

Формат ввода-вывода

%[flags] [width] [.precision] type

Type

тип вводимого параметра

Flags

специфика формата выводимого значения

Width

Минимальное выводимое число символов

Precision

Максимальное выводимое число символов после запятой

Формат ввода-вывод. Type

Символ	Тип	Пример
d *or* i	Signed decimal integer	392
u	Unsigned decimal integer	7235
o	Unsigned octal	610
x	Unsigned hexadecimal integer	7fa
X	Unsigned hexadecimal integer (uppercase)	7FA
f	Decimal floating point, lowercase	392.65
F	Decimal floating point, uppercase	392.65
e	Scientific notation (mantissa/exponent), lowercase	3.9265e+2
E	Scientific notation (mantissa/exponent), uppercase	3.9265E+2
g	Use the shortest representation: %e or %f	392.65

Поток

- **Поток данных** в программировании — абстракция, используемая для чтения или записи файлов, сокетов и т. п. в единой манере.
- Поддержка потоков включена в большинство языков программирования
- При запуске процесса ему предоставляются предопределенные стандартные потоки
- **Стандартные потоки ввода-вывода** — потоки процесса, имеющие номер (дескриптор), зарезервированный для выполнения некоторых «стандартных» функций:
  1. Стандартный ввод (stdin)
  2. Стандартный вывод (stdout)
  3. stderr

Структура FILE

FILE — структура, содержащая информацию о файле или текстовом потоке, необходимую для выполнения ее операций ввода и вывода операций, включая:

- файловый дескриптор (неотрицательное целое число)
- текущую позицию в потоке
- индикатор конца файла
- индикатор ошибок
- указатель на буфер потока, если возможно

Объявление	Описание
FILE* stdin	стандартный поток ввода (обычно клавиатура)
FILE* stdout	стандартный поток вывода (обычно дисплей терминала)
FILE* stderr	стандартный поток ошибок (обычно дисплей терминала)

Работа с файлами

FILE* fopen(const char *filename, const char *mode)

filename – имя открываемого файла

mode – тип доступа

Mode	Описание
“r”	Opens for reading. If the file does not exist or cannot be found, the fopen call fails
“w”	Opens an empty file for writing. If the given file exists, its contents are destroyed.
“a”	Opens for writing at the end of the file (appending) without removing the EOF marker before writing new data to the file; creates the file first if it doesn't exist.
“r+”	Opens for both reading and writing. (The file must exist.)
“w+”	Opens an empty file for both reading and writing. If the given file exists, its contents are destroyed.
“a+”	Opens for reading and appending; the appending operation includes the removal of the EOF marker before new data is written to the file and the EOF marker is restored after writing is complete; creates the file first if it doesn't exist.

Работа с файлами

int fclose(FILE *stream)

stream - Pointer to FILE structure

Return Value

fclose returns 0 if the stream is successfully closed

Работа с файлами

```
int main(int argc, char* argv[]) {
   FILE* fileHandle;
   fileHandle = fopen("test.dat", "w");
   if(fileHandle == NULL)
   {
       printf("Can't create file");
       return 1;
   }
   fclose(fileHandle);
   return 0;
}
```

Чтение\Запись в файл

- int fprintf(FILE *stream, const char *format [, argument]...);
- int fputc(int c, FILE *stream);
- int fputs(const char *str, FILE *stream);
- size_t fwrite(const void *buffer, size_t size, size_t count, FILE *stream);

- int fscanf(FILE *stream, const char *format [, argument]...);
- int fgetc(FILE *stream);
- char *fgets(char *str, int n, FILE *stream);
- size_t fread(void *buffer, size_t size, size_t count, FILE *stream);

EOF (End of File)

- возвращается следующими функциями при переходе курсора в конец файла:

- fscanf
- fgetc

```
int a = fgetc(file);
   while(a != EOF) {
       fputc(a, stdout);
       a=fgetc(file);
   }
```

Преобразование типов

- int atoi(const char *str)
- double atof(const char *str);
- long atol(const char *str)
- long strtol(const char *nptr, char **endptr, int base)
- unsigned long strtoul(const char *nptr, char **endptr, int base)

Генерация псевдослучайных последовательностей

int rand(void)

- Generates a pseudorandom number

Return Value

rand returns a pseudorandom number. The rand function returns a pseudorandom integer in the range 0 to RAND_MAX (32767).

---

void srand(unsigned int seed)

- Sets a random starting point.

Генерация псевдослучайных последовательностей

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

int main () {
  printf("First number: %d\n", rand() % 100);

  srand(time(NULL));
  printf("Random number: %d\n", rand() % 100);
  printf("Again the first number: %d\n", rand() %100);
  return 0;
}
```

Контроль процесса выполнения программы

void exit (int status)

- Terminate calling process

Status - Status value returned to the parent process. Generally, a return value of 0 or EXIT_SUCCESS indicates success, and any other value or the constant EXIT_FAILURE is used to indicate an error or some kind of abnormal program termination.

!NB Указатель на функцию

```
int increment(int value) {
    return value + 1;
}

int main(int argc, char* argv[]) {
    printf("%p\n", &increment);

    int (*fp)(int) = increment;
    int (*fp2)(int) = &increment;
    printf("%d\n", fp(2));
    printf("%d\n", (*fp)(3));

    return 0;
}
```

!NB Указатель на функцию

```c
int increment(int value) {
    return value + 1;
}

void print_unary_function(int(*fp)(int), int value) {
    printf("%d\n", fp(value));
}

int main(int argc, char* argv[]) {
    printf("%p\n", &increment);
    print_unary_function(increment, 2);

    return 0;
}
```

Контроль процесса выполнения программы

```
int main () {
   FILE * pFile;
   pFile = fopen ("myfile.txt","r");

   if (pFile==NULL) {
       printf ("Error opening file");
       exit (1);
   }
   else {
       /* file operations here */
   }
   return 0;
}
```

Контроль процесса выполнения программы

int atexit(void (__cdecl *func)(void))

- Processes the specified function at exit.

```
void fnExit1(void) {
  puts("Exit function 1.");
}
void fnExit2(void) {
  puts("Exit function 2.");
}

int main(){
  atexit(fnExit1);
  atexit(fnExit2);
  puts("Main function.");
  return 0;
}
```

Сортировка

```
void sort(int* arr, size_t size) {
    for(int i = 0; i < size - 1; ++i) {
        for(int j = 0; j < size - i - 1; ++j) {
            if(arr[i] > arr[i+1]) {
                int temp = arr[i];
                arr[i] = arr[i+1];
                arr[i+1] = temp;
            }
        }
    }
}
```

Сортировка

```
int main () {
    int arr[] = { 5, 4, 3, 2, 1};
    size_t cnt = sizeof(arr)/sizeof(int);

    sort(arr, cnt);
}
```

А если хотим отсортировать не int?

```
int main () {
    double arr[] = { 5, 4, 3, 2, 1};
    size_t cnt = sizeof(arr)/sizeof(double);
    sort(arr, cnt);
}
```

Копипаст кода плохое решение:

- Не решает проблему сортировку пользовательских типов
- Сложно исправлять ошибки
- Сложно поменять алгоритм (например на qsort)

Что мешает сделать функцию обобщенной?

```
void sort(int* arr, size_t size) {
    for(int i = 0; i < size - 1; ++i) {
        for(int j = 0; j < size - i - 1; ++j) {
            if(arr[i] > arr[i+1]) { // 1. Получение i-го элемента
                                    // 2. Сравнение двух элементов произвольных типов
                int temp = arr[i];  // 3. swap произвольных элементов
                arr[i] = arr[i+1];
                arr[i+1] = temp;
            }
        }
    }
}
```

Получение i-го элемента произвольного типа

```
int main () {
    int arr[] = { 5, 4, 3, 2, 1};
    size_t cnt = sizeof(arr)/sizeof(int);

    void* some_arr = (void*)arr;

    void* pointer_on_third = some_arr + sizeof(int) * 2; // Арифметика указателей
    int third  = *(int*) pointer_on_third;

    return 0;
}
```

Сравнение двух элементов

```
int compare_int(const void* a, const void* b) {
   return *(int*)a - *(int*) b;
}

int compare_double(const void* a, const void* b) {
   return *(double*)a - *(double*) b;
}
```

swap

```
// Поменять все байты местами
void swap(void* a, void* b, size_t size_of_el) {
    for(int i = 0 ; i < size_of_el; ++i) {
        char ch = *(char*)(a + i);
        *(char*)(a + i) = *(char*)(b + i);
        *(char*)(b + i) = ch;
    }
}
```

Сортировка

```
void sort(
  void* arr,                                  // обезличенный указатель на массив
  size_t size,                                // размер массива
  size_t size_of_el,                          // размер одного элемента массива
  int(*compare)(const void*, const void*)     // функция сравнения
) {
  for(int j = 0; j < size - 1; ++j) {
      for(int i = 0; i < size - j - 1; ++i) {
          if(compare(arr + i * size_of_el, arr + (i + 1) * size_of_el) > 0) {
              swap(arr + i * size_of_el, arr + (i + 1) * size_of_el, size_of_el);
          }
      }
  }
}
```

Сортировка

```
void qsort (
        void *base,
        size_t num,
        size_t width,
        int (__cdecl *compare )(const void *, const void *)
);
```

- Performs a quick sort

base - Start of target array.

Num - Array size in elements.

width - Element size in bytes.

compare - Comparison function. The first parameter is a pointer to the key for the search and the second parameter is a pointer to the array element to be compared with the key.

Сортировка

```
int compare (const void * a, const void * b) {
    return ( *(int*)a - *(int*)b );
}

int main () {
    int values[] = { 40, 10, 100, 90, 20, 25};

    qsort(values, 6, sizeof(int), compare);
    for (int i = 0; i < 6; ++i)
       printf("%d ",values[i]);
    return 0;
}
```

Сортировка

```
struct SCow {
   float weight_kg;
   int   milk_liter;
};

int main (){
   struct SCow herd[] = {
       { 120.0, 10 },
       { 100.1, 12 },
       { 102.2, 17 },
       { 111.3, 9  },
   };
   size_t cnt = sizeof(herd)/sizeof(struct SCow);
   qsort(herd, cnt, sizeof(struct SCow), cow_comparator);
   return 0;
}
```

Сортировка

```
int cow_comparator(const void* a, const void* b) {
    struct SCow* first = (struct SCow*)a;
    struct SCow* second = (struct SCow*)b;

    return first->milk_liter - second->milk_liter;
}

int cow_comparator_by_weight(const void* a, const void* b) {
    struct SCow* first = (struct SCow*)a;
    struct SCow* second = (struct SCow*)b;

    return -(first->weight_kg - second->weight_kg);
}
```

Поиск

void *bsearch (const void *key,

const void *base,

size_t num,

size_t width,

int (__cdecl *compare) (const void *, const void *));

- Performs a binary search of a sorted array.

key - Object to search for.

base - Pointer to base of search data.

Num - Number of elements.

Width **-** Width of elements.

compare - Callback function that compares two elements. The first is a pointer to the key for the search and the second is a pointer to the array element to be compared with the key.

Поиск

```
int compareints(const void * a, const void * b){
  return ( *(int*)a - *(int*)b );
}

int main (){
   int values[] = { 10, 20, 25, 40, 90, 100 };
   int * pItem;
   int key = 40;
   pItem = (int*) bsearch (&key, values, 6, sizeof(int), compareints);

   if (pItem != NULL)
       printf ("%d is in the array.\n", *pItem);
   else
       printf ("%d is not in the array.\n", key);

   return 0;
}
```