Задание для подготовки к Контрольной работе #2

○ – вопрос
□ – задача
🦇 – продвинутая задача

Указатели

Основы работы с указателями

Адреса переменных. Операция получения адреса

Каждая переменная в языке C хранится где-то в памяти и имеет адрес. Адрес переменной это просто номер первого байта соответствующей области памяти. Чтобы получить адрес переменной нужно перед переменной поставить &(амперсанд). Обычно адреса записываются в шестнадцатеричной системе счисления, например 0x7FFFB014 это адрес ячейки под номером 2147463188 (такое число получится если перевести 0x7FFFB014 из шестнадцатеричной системы в десятичную).

```
double x = 123.456;
printf("Address of x is: %p\n", &x);
```

Объявление указателей

Указатель это переменная, которая хранит адреса переменных. Тип указателя такой: <тип переменной>*. Тип переменной, на которую указывает указатель, нужно знать, чтобы правильно выполнять операцию разыменования. Если тип переменной, на которую будет указывать указатель, не известен, то можно использовать указатель `void*`. Размер указателя обычно равен 4 байта на 32-х битных системах и 8 байт на 64-х битных. Так как указатель это переменная, то у него тоже есть адрес. Пример:

```
int a = 100;  // Переменная типа int, присвоено значение, равное 100
int* q;       // Переменная типа указатель на int, значение не присвоено
int* p = &a;  // Переменная типа указатель на int, присвоено значение, равное адресу a
void* pv = &a;// Переменная типа ' указатель на что - то ', присвоено значение, равное адресу a
```

Операция разыменования

Чтобы доступиться к переменной по указателю нужно поставить символ * перед указателем. Операция получения значения переменной по указателю называется операцией разыменования.

```
int a = 100;
int* p = &a; // Теперь можно использовать a с помощью p. Можно сказать, что *p это синоним a.
printf("%d", *p);   // Напечатает 100
*p += 10;           // Значение переменной a увеличится на 10
void* pv = &a;
printf("%d", *pv); // Неправильно, так как неизвестно какой тип у *pv.
```

Арифметика указателей. Операции, которые можно производить с указателями:

1. Присваивание другому указателю или адресу.

   ```
   float a[5] = {1.2, 3.4, 5.6, 7.8, 9.0};
   float* p = &a[3];
   char* pc = &a[0];  // Warning: Слева и справа разные типы. Тем не менее в языке C это не
       ошибка и работать будет. Указатель float* приведётся к указателю char*.
   ```

2. Сравнение двух указателей. Операция ==

3. Сложение указателя с целым числом. Возвращается указатель, смещённый на n * sizeof(type)

   ```
   p = p + 3; // p увеличится 3 * sizeof(float)
   ```

4. Вычитание двух указателей. Возвращается разница двух адресов, делённая на sizeof(type).

   ```
   printf("%lu", p - a); // Напечатает 3. Массив часто ведёт себя как указатель на 0 элемент
   ```

5. Операция взятия индекса: `p[2]`, `p[-1]`, `a[3]` и т. д.
   `a[i]` это то же самое что и `*(a + i)`.

□ Задача #1: Адрес

Пусть есть указатель p на переменную типа int, которая имеет адрес 0x7FFFB014. Переменная int на рассматриваемой системе имеет размер 4. Чему равны значения (в шестнадцатеричной системе счисления):

- p + 1
- p + 2
- p + 6

Применение указателей

Передача адресов переменных в функцию

Указатели часто используются чтобы изменять передаваемые значения в функциях:

```
// Неправильно:
void normalize(float x, float y)
{
    float sum = x + y;
    x = x / sum;
    y = y / sum;
    // Изменятся x и y - копии a и b
}
// ...
float a = 20.0, b = 80.0;
normalize(a, b);
// a и b не изменятся: a=20.0, b=80.0
```

```
// Правильно:
void normalize(float* x, float* y)
{
    float sum = *x + *y;
    *x = *x / sum;
    *y = *y / sum;
    // Изменятся переменные a и b
}
// ...
float a = 20.0, b = 80.0;
normalize(&a, &b);
// a и b изменятся:a=0.2, b=0.8
```

□ Задача #2: Передача по адресу

Написать функцию `void make_positive(float* x)`, которая делает число положительным. Например:

```
float x = -1.2; y = 4.5;
make_positive(&x); // x изменится и станет равным 1.2
make_positive(&y); // y не изменится
```

Передача в функцию с использованием указателя на константу

Ещё один часто используемый способ передачи в функцию – передача с использованием указателя на константу

```
void some_func(const float* p)
{
        printf("%f\n", *p);  // OK
        *p = 10.0;           // Неправильно!
        // Нельзя менять то, на что указывает p, так как используется const
}
```

○ Какие преимущества такой передачи по сравнению с обычной передачей и передачей с использованием указателя? Когда следует использовать такой способ передачи?

Структуры

Объявление структуры

```
struct city
{
    char name[50];   // Название города
    int population;  // Население
    float area;      // Площадь города
};
typedef struct city City;
```

Создание экземпляра структуры

```
City x = {"Moscow", 12228685, 2511.0};
x.area *= 2; // Увеличим площадь в 2 раза
City best_cities[] = {
    {"New York", 8175133, 1213.37},
    {"Tokyo", 13617445, 2187.66 },
    {"Shanghai", 24152700, 6341.0},
    {"Dolgoprudniy", 104238, 30.52}};
```

Передача структуры в функцию

Структуры в функции передаются как обычные переменные

```
// Передача по значению (x копируется)
void print_name(City x)
{
    printf("%s", x.name);
    // Используем точку .
}
// Передача через указатель
void increase_population(City* p, int n)
{
    p->population += n;
    // Используем стрелочку ->
}
```

Рассмотрим задачу парсирования из строки в структуру. Предположим, что информация о городе записана в строке в следующем формате: `char str[100] = ‘‘Moscow-12228685-2511.0’’`, тогда для считывания из строки и записи нужных значений в структуру можно использовать функцию `sscanf()`:

```
char str[100] = ‘‘Moscow-12228685-2511.0’’;
City a;
sscanf(str, "%[^-]-%d-%f, , &a.name, &a.population, &a.area");
// Выражение %[^-] означает считываем всё до знака - в строку
```

☐ Задача #3: Парсим координаты

- Объявите структуру Point, описывающую точку в пространстве. Эта структура должна иметь поля x, y, z типа float.
- Напишите функцию `void print_point(Point a)`, которая принимает на вход эту структуру и печатает её на экран в следующем формате: (x, y, z).
- Предположим, что информация о точке записана в строке в следующем формате: `char str[50] = ‘‘(1.5, 4.0, 3.7)’’`. Создать новую структуру Point и считать информацию из строки в эту структуру с помощью sscanf().
- Напишите функцию `Point parse_point(char* str)`, которая считывает информацию из строки `str` в структуру и возвращает эту структуру. Нужно использовать функцию `sscanf()`.

Следующий код должен выводить на экран координаты, записанные в строках s1 и s2:

```
int main()
{
    char s1[] = "(3.2, 535.0, -74.78)";
    char s2[] = "(-777, 0.000, 123456789.987654321)";
    Point a = parse_point(s1);
    Point b = parse_point(s2);
    print_point(a);
    print_point(b);
}
```

Строки

Символы

Тип char – тип целых чисел от -128 до 127
Символы кодируются в соответствии с таблицей ASCII, например, символ '9' это просто число 57.

```
char a = 100; // Как число от -128 до 127
char b = 'A'; // Как символ
if ('9' == 57)
    printf("Char is a number\n");
```

Некоторые символы ASCII:

'0' = 48	**'A'** = 65	нулевой символ **'\0'** = 0
'1' = 49	**'B'** = 66	символ переноса строки **'\n'** = 10
'2' = 50	...	табуляция **'\t'** = 9
...	**'Z'** = 90	backspace **'\b'** = 8
'9' = 57	**'a'** = 97	звуковой сигнал **'\a'** = 7
' ' = 32	...	возврат каретки **'\r'** = 13
	'z' = 122	

Строки как массив символов

Строка это просто массив из символов. Строка должна заканчиваться символом '\0'.

```
char s1[10];          // Объявление строки
char s2[10] = "abc";  // Объявление + инициализация

s1 = "xyz"; // Неправильно! Так присваивать строки нельзя, так как строка это просто массив
strcpy(s1, "xyz"); // Чтобы присвоить строке значение нужно использовать функцию strcpy()
                   // Для использования функций работы со строками нужно подключить <string.h>
```

☐ Задача #4: Обращение строки

Написать функцию `void reverse(char* str)`, которая переворачивает строку.

☐ Задача #5: Шифр Цезаря

Написать функцию `void encrypt(char* str)`, которая изменяет строку следующим образом: все символы 'a' меняются на 'b', все символы 'b' меняются на 'c' ... все символы 'z' меняются на 'a'. То же самое для заглавных букв. Остальные символы не меняются.

Функции для работы со строками

- `strlen(str)` – вычисляет количество символов в строке str.
- `strcpy(dest, src)` – копирует содержимое строки src в строку dest.
- `strcmp(str1, str2)` – сравнивает строки str1 и str2. Возвращает 0, если они равны.
- `strstr(str, substr)` – возвращает указатель на первый символ вхождения строки substr в строку str.
- `strcat(str1, str2)` – добавляет строку str2 в конец строки str1.

☐ Задача #6: Строка + указатели

Предположим, что задана строка `char str[100] = "A bicycle can't stand on its own because it is two-tired"`, что напечатают следующие строки:

- `printf("%lu", strlen(str))`
- `printf("%s", str)`
- `printf("%s", str + 33)`
- `printf("%s", strstr(str, "bec"))`
- `printf("%lu", strstr(str, "can") - str)`

Динамическое выделение памяти

```
int* arr = malloc(n * sizeof(int));
// ...
free(arr);
```

☐ Задача #7: Среднее и дисперсия + malloc()

На вход программе подаётся целое число n и n вещественных чисел типа double x_i. Нужно найти среднее этих чисел μ и дисперсию D. Для выделения памяти используйте malloc() и free().

$$\mu = \frac{1}{n}\sum_{i=0}^{n-1} x_i \qquad D = \frac{1}{n}\sum_{i=0}^{n-1}(x_i - \mu)^2$$

Выделение памяти для 2D массива

Сначала нужно выделить память для указателей на строки. А затем выделить память для каждой из строк.

```
// Выделяем память под 2D массив
int** A = malloc(n * sizeof(int*));
for (int i = 0; i < n; ++i)
        A[i] = malloc(m * sizeof(int));
// Работаем с 2D массивом A как с обычным 2D массивом
// ...
// Освобождаем память
for (int i = 0; i < n; ++i)
        free(A[i]);
free(A);
```

☐ Задача #8: Malloc 2D

На вход программе подаются 2 целых числа n и m. Затем на вход подаются $n \times m$ вещественных чисел типа. Нужно напечатать матрицу, отзеркаленную по вертикали. Всю необходимую память выделить динамически.

Сортировка (qsort)

Сортировка чисел:

```
int cmp_int(const void* a, const void* b)
{
    int* pa = (int*)a;
    int* pb = (int*)b;
    return (*pa - *pb);
}



int main ()
{
    int arr[] = {8, 2, 5, 1};
    qsort(arr, 4, sizeof(int), cmp_int);
}
```

Сортировка структур:

```
int cmp_city(const void* a, const void* b)
{
    City* pa = (City*)a;
    City* pb = (City*)b;
    return (pa->population -
            pb->population);
}
int main ()
{
    City arr[] = {{''A'', 78, 5.4},
                  {''B'', 54, 1.9},
                  {''C'', 97, 4.2}};
    qsort(arr, 3, sizeof(City), cmp_city);
}
```

☐ Задача #9: Сортировка

Создать массив структур City и инициализировать его 5-ю различными городами. Отсортировать этот массив по плотности населения.

Файлы

Считывание/запись.

```
FILE* fin = fopen("input.txt", "r");
int x, y, z;
fscanf(fin,"%d%d%d", &x, &y, &z);
fclose(fin);


FILE* fout = fopen("output.txt", "w");
fprintf(fout,"Hello file!\n");
fclose(fout);
```

Посимвольное считывание/запись.

```
FILE * f = fopen("input.txt", "r");
int c, number_of_digits = 0;

while ((c = fgetc(f)) != EOF)
{
    if (c >= '0' && c <= '9')
        number_of_digits += 1;
}
fclose(f);
```

☐ Задача #10: Среднее и дисперсия + malloc() + file

Решить задачу #7, но только считывание должно проводится не из стандартного входа, а из файла “input.txt”.

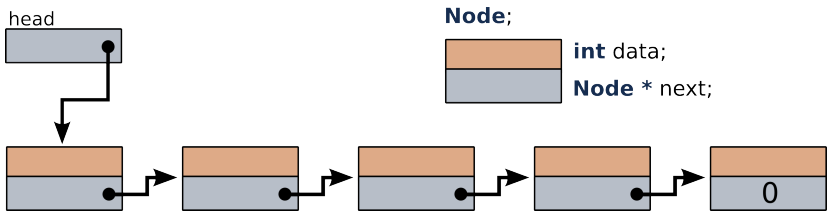
☐ Задача #11: Количество слов в файле.

Посчитать число слов в файле “input.txt”. Слово это любая последовательность символов, разделённая пробелом, символом табуляции('\t') или символом перевода строки('\n').

✒ Задача #12: qraces

Решить задачу qraces-1 из контрольной работы 2015-2016. + добавить считывание из файла “input.txt” и запись в файл “output.txt”.

Связный список



Базовый исходный код связного списка можно найти тут:
github.com/v-biryukov/cs_mipt_faki/tree/master/term1/seminar11_list/programms

☐ Задача #13 Перевернуть список

Написать функцию `void list_reverse(Node** p_head)`, которая переворачивает связный список. Первый элемент становится последним, а последний первым.

✒ Задача #14 Есть ли цикл

Написать функцию `int list_is_loop(Node* head)`, которая проверяет, если в связном списке цикл.

✒ Задача #15 Устранить цикл

Написать функцию `int list_is_loop(Node** p_head)`, которая проверяет, если в связном списке цикл и если цикл есть, то он устраняется. Подробней можно прочитать тут:
www.geeksforgeeks.org/detect-and-remove-loop-in-a-linked-list