

Программирование и алгоритмизация

К.т.н., доцент

Гаврилов Андрей
Владимирович

<http://www.insycom.ru>

Email: andr_gavrilov@yahoo.com

V-241

Цели курса

- Алгоритмизация
 - Псевдокоды для записи алгоритма
 - Структурное программирование
 - Нисходящая разработка программ
- Дальнейшее изучение программирования на языке C
 - Отличия C++ от C,
 - Структуры данных
 - Списки, работа с указателями
 - Алгоритмы сортировки и поиска
- Объектно-ориентированное программирование
 - Основные концепции: классы и объекты, инкапсуляция, наследование, полиморфизм
 - Статическое и динамическое создание объектов
 - Виртуальные функции
 - Шаблоны классов и функций
- Особенности языка программирования C#
- Основы программирования в среде CBuilder и Visual Studio
 - Основные визуальные компоненты для создания пользовательского интерфейса,
 - Основные визуальные компоненты для работы с базами данных

Аттестация

Соответствие баллов (кредитов) видам учебной деятельности

Виды учебной деятельности	Баллы (максимально возможное количество)	Сроки сдачи
Лекции (9)	10 (посещение всех лекций)	в соответствии с расписанием
Лабораторная работа (9 занятий)	5 (45)	в соответствии с расписанием
Расчетно- графическое задание	5	РГЗ – до 13 недели
Курсовая работа	15	Зачетная неделя
Экзамен	25	Сессия

Литература

- Программирование на языке С.(.chm)
- В.Александркин, Н. Елманова. Начинаем работать в Borland C++ Builder (doc)
- C++ Builder. Обучалка. 7 уроков. (.HTML)
- Лекции по языку С. 8 лекций. (.Doc)
- К.Рейсдорф, К.Хендерсон. Borland C++Builder. Освой самостоятельно. (Pdf)
- Т.А.Павловская, Ю.А.Щупак. С++. Объектно-ориентированное программирование. Практикум. Уч. пособие. Питер, 2006. (pdf)
- Павловская Т. А. С/С++. Программирование на языке высокого уровня : учебник для вузов. - СПб., 2006.
- Н.Культин. С++ Builder в задачах и примерах. С.-Петербург: БХВ-Петербург, 2005. (pdf)
- Дейтел Х. М. Как программировать на С++. - М., 2007.
- Липпман. С++ для начинающих. (pdf)
- В.В.Мухортов. В.Ю. Рылов. Объектно-ориентированное программирование. Анализ и дизайн. Методическое пособие. Новосибирск: ООО «Новософт», 2002.
- Д.Хенкеманс, М.Ли. Программирование на С++. С.-Петербург: Символ, 2002. (pdf)
- Алгоритмы, методы, исходники (.chm)

Литература (2)

- Е.Л.Романов. Язык Си. Типы данных и управление памятью. Конспект лекций. Новосибирск: НГТУ, 2000.
- Е.Л.Романов. С/С++/ От дилетанта до профессионала <http://ermak.cs.nstu.ru/cprog/HTML/index.htm>
- Кузнецов С. М., Малоземов Б. В. Программирование и основы алгоритмизации : учебное пособие для ЭМФ специальность 220301, 140609, 140606 и 657900 дневной и заочной форм обучения. Новосиб. гос. техн. ун-т. - Новосибирск, 2006.
- Введение в программирование на С#. Интернет - Университет Информационных Технологий. - <http://www.INTUIT.ru> (pdf)
- А.П.Марченко. С#. Введение в программирование. Уч. пособие. МГУ, 2005. (pdf)
- Ч. Петцольд. Программирование на Microsoft Windows на С#. М.: «Русская редакция», 2002, том 1 (pdf)
- Обучающие уроки по С#. –

<http://www.programmer-lib.ru/csharp.php>

Гаврилосв А.В.
НГТУ, кафедра АИПМ

Лекция 1

Избранные разделы языка C:

- Многомерные массивы,
- Строки,
- Указатели,
- Функции,
- Работа с файлами,
- Структуры, объединения и битовые поля

Многомерные массивы и операции с НИМИ

```
int a[2][3];
```

	1	2	3	4	5		
[1	2	5]	6	7	8	9	10
[6	7	2]	11				

```
double A[3][2]={{10,20}, {30,40}, {50,60}};
```

```
double A[3][2]={10,20,30,40,50,60};
```

```
int A[3][5]={1,2,3,4,5,6,7,8,9,10,11};
```

```
int A[3][5]={{1,2,3},{4,5,6,7,8},{9,10,11}};
```

```
int arr[][3]={1, 2, 3,
```

	1	2	3		
4, 5, 6,	4	5	6	7	8
7, 8, 9};	9	10	11		

```
int arr[2][3];  
for (i=0; i<2;i++)  
for (j=0; j<3;j++) scanf("%d",&arr[i][j]);
```

```
for (i=0; i<2;i++) {  
for (j=0; j<3;j++)  
printf("%d ",arr[i][j]);  
printf("\n");  
}
```

1	2	5
6	7	2


```
int a[2][3];
int at[3][2];
int i,j;
printf("Enter array\n");
for (i=0;i<2;i++)
for (j=0;j<3;j++) {
scanf("%d",&a[i][j]);
at[j][i]=a[i][j];
}
printf("Транспонированный массив\n");
for (i=0;i<3;i++) {
for (j=0;j<2;j++) printf("%d ",at[i][j]);
printf("\n");}
```

A[4][5], B[4][5]

...

for (i=0;i<4;i++)

for (j=0;j<5;j++) C[i][j]=a[i][j]+b[i][j];

...

A[M][N] B[N][L]

C[M][L]

const M=3, N=4, L=5;

int a[M][N];

int b[N][L];

int c[M][L];

...

for(i=0;i<L;i++)

for(k=0;k<M;k++)

for(j=0;j<N;j++) c[k][i]=c[k][i]+a[k][j]*b[j][i];

...

Пример: ввод в массив mas символов до точки с пропуском пробелов

```
#include <stdio.h>
void main(void)
{char ch, mas[50];
int k;
for (k=0;(ch=getchar())!='.');
```

```
{ mas[k]=ch;
if (ch!=' ') k++;
}
}
```

Строки

- одномерные массивы символов,
заканчивающиеся символом с кодом ноль ('\\0')

```
char str[20];
```

```
str[0]='H'; str[1]='e'; str[2]=str[3]='l';
```

```
str[4]='o'; str[5]='\\0';
```

```
char str[]={'H','e','l','l','o','\\0'};
```

```
char* str1="Hello";
```

```
char str[]="Привет";
```

```
char str[80]= "Привет";
```

Функции для работы со строками

```
#include <string.h>
```

```
strcpy(s1,s2); // копирование строки
```

```
strcpy(s1,"Hello ");
```

```
char *s=strcat(s1,s2); // результат –  
конкатенация строк
```

```
int n=strcmp(s1,s2); // сравнение строк
```

```
int n=strlen(s); // длина строки
```

```
int x=atoi(s); // преобразование строки в  
число
```

Переменные и указатели

- Переменная – ячейка в памяти, содержимое которой мы используем в программе, как значение переменной, обращаясь к ней, используя *идентификатор* (например, *a*, *abc*, *A12 ...*)
- Указатель – ячейка в памяти, содержащая адрес некоторой переменной. Обращение к указателю, используя его идентификатор (например, **a*, **a12 ...*)
- Ссылка – адрес переменной, не имеющий своего идентификатора (например, адрес массива *a[]* - *&a[0]*)

Типизированные указатели

`char *c; // указатель на char`

`int *i, j; // указатель на int и просто int`

`i=&j; // i присвоить адрес j`

`*i=1; // разыменованный указатель j=1`

`*` - операция косвенной адресации

`&` - операция получения адреса

Указатели

`void *p; // нетипизированный указатель`

`float *pf, f; // типизированный указатель`

`pf=&f; // pf присвоить адрес f`

`p=pf; // одному указателю присвоить значение другого`

`pf=(float *) p; // явное указание типа при разыменовании`

Пример: вычисление суммы квадратов первых 100 натуральных чисел

```
int  *ptr;  
long *sum;  
*sum = 0;  
for (*ptr = 1; *ptr <= 100; (*ptr)++)  
    *sum = *sum + (*ptr)*(*ptr);
```

Массивы и указатели

Описание

```
int x[]={1,2,3,4,5};          int x[5]; const int *y=x;
```

Адрес элемента массива

&x[3]

y+3

Значение элемента массива

x[3]

*(y+3)

Имя массива – неизменяемый указатель, инициализированный начальным значением.

Массивы и указатели (2)

```
int a[10], *p;  
char *p;  
char *str="Strings With Capital Words";  
p=str;  
while (*p) putc(*p++);
```

Массивы и указатели (3)

$p=a;$ \longleftrightarrow $p=\&a[0];$

$a[5]$ \longleftrightarrow $*(p+5)$

Массивы указателей

```
char *ext[]={ "exe", "com", "dat", "c", "pas", "cpp" }
```

```
int **p;
```

ФУНКЦИИ

- Функция до её использования должна быть описана или объявлена в виде (имена параметров игнорируются):
 - *Прототип_функции;*
- Функция должна быть описана только один раз в виде:
 - *Прототип_функции { тело_функции }*
- Формат прототипа:
 - *Тип_возвращаемого_значения имя(список_параметров)*
- Формат списка параметров:
 - *тип имя_аргумента, ..., тип имя_аргумента*
- Функцию можно определить со спецификатором inline
 - `inline int fac(int n) {return (n<2)?1:n*fac(n-1);}`
- Программа начинает выполняться с функции main.

Возвращаемое значение

- Любая функция, если она не объявлена как `void`, должна возвращать значение.
- Это значение задается в инструкции `return выражение;`
- инициализации неименованной переменной возвращаемого типа.
- Функция может иметь несколько инструкций `return`.
- Если функция не возвращает значения, то выражение в инструкции `return` может быть пустым или вызовом функции типа `void`.

Функции

Описание функции:

```
Тип_функции имя_функции(формальные параметры );
```

```
{
```

```
Тело функции
```

```
}
```

```
/* не возвращает значение*/
```

```
Тип_функции имя_функции(формальные параметры );
```

```
{
```

```
Тело функции
```

```
return(переменная-результат того же типа);
```

```
}
```

```
/* возвращает значение функции заданного типа*/
```


Пример функции нахождения суммы массива чисел

```
// Результат функции
// | Имя функции
// ||
int sum(int A[], int n)
// ----- Формальные параметры
//----- Тело функции (блок)
{
int s,i; // Локальные переменные блока
for (i=s=0; i<n; i++) // Последовательность операторов блока
s +=A[i];
return s ; // Значение результата в return
}
```

Пример: функция вычисления квадратного корня

```
#include <stdio.h>
float sqrt(arg)
float arg;
    { int  count;
      float root = arg/2.0;
      for (count = 1; count <= 5; count++)
          root = 0.5*(root + arg/root);
      return (root);
    }
void main()
    { float dat;
      printf("\nЗадайте положительное вещественное число ... ");
      scanf("%f", &dat);
      printf("\n\nКорень из числа %.3f равен %.3f", dat, sqrt(dat));
    }
```

Пример . Функция удаления лишних пробелов

```
// Удаление лишних пробелов при посимвольном переписывании
#include <stdio.h>
void nospace(c1[],c2[])
{
int i,j;
for (j=0,i=0;c1[i]!=0;i++)          // Посимвольный просмотр строки
{
if (c1[i]!=' ')                    // Текущий символ не пробел
{
if (i!=0 && c1[i-1]==' ')          // Первый в слове -
c2[j++]=' ';                      // добавить пробел
c2[j++]=c1[i];                    // Перенести символ слова
}                                  // в выходную строку
}
c2[j]=0;
}
```

Пример. Функция поиска слова максимальной длины в строке

```
// Поиск слова максимальной длины посимвольная обработка
// Функция возвращает индекс начала слова или 1, если нет слов
int find(char s[])
{
    int i,n,lmax,imax;
    for (i=0,n=0,lmax=0,imax=-1; s[i]!='\0'; i++)
    {
        if (s[i]!=' ') n++;           // символ слова увеличить счетчик
        else                          // перед сбросом счетчика
        {
            // фиксация максимального значения
            if (n > lmax) { lmax=n; imax=i-n; }
            n=0;
        }
    }
    // то же самое для последнего слова
    if (n > lmax) { lmax=n; imax=i-n; }
    return imax;
}
```

Пример: использование указателей в качестве параметров функций

```
void abs(arg)
  int *arg; /* Формальный параметр */      { *arg
  = (*arg >= 0) ? (*arg) : -(*arg);        return;
}
```

Обращение к функции:

```
int value;
```

```
abs(&value);
```

Еще пример:

```
scanf("%d", &alpha); // ввод числа alpha
```

Время существования и область видимости переменных

В предыдущем примере (sqrt)

dat – локальная переменная, т.к. описана в функции main(), хотя для вызываемых в ней функций она – глобальная.

Если ее вынести за пределы функции, она будет глобальной переменной

Время жизни локальной переменной – время выполнения функции, в которой она описана

Пример

```
//-----Численное интегрирование произвольной функции
double INTEG(double a, double b, int n, double(*pf)(double))
    // a,b - границы интегрирования
    // n - число точек
    // pf - подынтегральная функция (указатель на нее)
{
double s,h,x;
for (s=0., x=a, h = (b-a)/n; x <=b; x+=h)
    s += (*pf)(x) * h; // использование указателя на функцию pf
return(s); // возвращение вычисленного значения
}
extern double sin(double); // описание ссылки на фактическую функцию
// главная программа с вызовом функции
void main()
{ cout << INTEG(0.,1.,40,sin); }
```

Работа с файлами

FILE – тип дескриптора файла, содержащего информацию о нем (создается системой)

```
FILE *fopen(pathname, type)
```

```
/* Имя открываемого файла */
```

```
const char *pathname;
```

```
/* Тип доступа к файлу */
```

```
const char *type;
```

Функция возвращает указатель типа FILE при нормальном завершении операции и NULL в случае возникновения ошибки

Тип доступа к файлу

"r" - существующий текстовый файл открыть для чтения

"w" - текстовый файл открыть для записи (в случае существования файла его содержимое разрушается)

"a" - текстовый файл открыть для записи в конец файла (в случае отсутствия файл создается вновь)

"r+" - существующий файл открыть для чтения и записи

"w+" - текстовый файл открыть для чтения и записи (в случае существования файла его содержимое разрушается)

Добавление символа 'b' в конец каждой из этих строк позволяет открыть двоичный файл для выполнения соответствующих операций.

Пример

```
#include <stdio.h>
void main()
{
FILE *stream;
if ((stream = fopen("data", "r")) == NULL)
printf("Ошибка при открытии файла");
}
```

Заккрытие файла

Имя функции и назначение: `fclose` -
закрывает файл, предварительно
открытый для ввода/вывода

Формат и описание аргументов:

```
int fclose(stream)
```

```
/* Указатель на открытый файл */
```

```
FILE *stream;
```

Пример

```
#include <stdio.h>
FILE *fd;          // Переменная fd - указатель на дескриптор файла
fd = fopen("aaa.txt","r");
// режим "чтение текстового файла"
// имя файла - строковая константа
// или указатель на строку
// fopen() возвращает указатель на дескриптор файла
// запомнить результат функции
if (fd == NULL)
    printf("Файл не открыт\n");
else
    {
        // Указатель на дескриптор файла
        fscanf(fd,.....); // при обращении к файлу
        fclose(fd);       // Закрывать файл
    }
```

Функции для работы с файлами библиотеки SysUtils C++Builder

Установочные и завершающие операции

Операция	Содержание
FileExists(FileName)	Функция. Проверяет существование файла с именем FileName. Возвращает True, если файл существует, и False в противном случае.
FileCreate(FileName);	Функция. Создает файл с именем FileName. Возвращает дескриптор файла. При ошибке возвращается -1. Дескриптор файла – это целое число, с которым ИСР связывает файл.
FileGetAttr(FileName);	Функция. Для файла с именем FileName возвращает атрибут Attr. При ошибке возвращается -1.
FileSetAttr(FileName, Attr);	Функция. Для файла с именем FileName устанавливает атрибут Attr. При ошибке возвращается -1.
FileOpen(FileName, Mode);	Функция. Открывает файл с именем FileName в режиме Mode. Возвращает дескриптор открытого файла. При ошибке возвращается -1.
FileClose (Handle);	Процедура. Закрывает файл с дескриптором Handle.
DeleteFile(FileName);	Функция. Удаляет на диске файл с именем FileName. Возвращает True, если операция прошла успешно, и False в противном случае.
RenameFile(OldName, NewName);	Функция. Изменяет имя файла OldName на NewName. Возвращает True, если файл переименован, или False в противном случае.

Атрибуты файлов

Атрибут	Значение
faReadOnly	Только для чтения
faHidden	Скрытый
faSysFile	Системный
faVolumeID	Метка диска
faDirectory	Каталог
faArchive	Архивный
faAnyFile	Любой

Режимы файлов

Режим	Значение
Флаги открытия	
fmCreate	Файл создается
fmOpenRead	Только для чтения
fmOpenWrite	Только для записи
fmOpenReadWrite	Для чтения и записи
Флаги совместного использования файла	
fmShareExclusive	Запретить другим доступ
fmShareDenyWrite	Запретить другим запись
fmShareDenyRead	Запретить другим чтение из файла
fmShareDenyNone	Разрешить другим все

Операции перемещения по файлу

Операция	Содержание
<code>FileSeek(Handle, Offset, Origin);</code> File – файл, Seek – поиск, Handle – дескриптор, Offset – смещение, Origin – начало.	Функция. В файле, связанном с дескриптором Handle, смещает маркер на Offset относительно позиции Origin (для Origin возможны: 0 – начало файла, 1 – текущая позиция, 2 – конец файла). Возвращает новую позицию маркера от начала файла. При ошибке возвращается -1.

Операции ввода/вывода

Операция	Содержание
FileRead(Handle, Buffer, Count);	Функция. Читает из файла, связанного с дескриптором Handle, в буфер Buffer Count байт. Возвращает число фактически считанных байтов. При ошибке возвращается -1.
FileWrite(Handle, Buffer, Count);	Функция. Пишет в файл, связанный с дескриптором Handle, из буфера Buffer Count байт. Возвращает число фактически записанных байтов. При ошибке возвращается -1.

Специальные операции

Операция	Содержание
DiskFree(Drive);	Функция. Возвращает размер свободного места на диске Drive в байтах. При ошибке возвращается -1.
DiskSize(Drive);	Функция. Возвращает размер диска Drive в байтах. При ошибке возвращается -1.

Составные типы данных

- Структура – конгломерат переменных
- Объединение – наложение переменных
- Битовое поле – организация доступа к отдельным битам
- Перечисление – список именованных целых констант
- Имена типов, введенные пользователем:

```
typedef тип новое_имя;
```

Структуры

Формат описания:

```
struct имя {  
    тип имя_члена;  
    ...  
    тип имя_члена; }  
список_переменных;
```

Пример

```
struct complex {  
    float re,im} a;
```



a

Массивы структур:

```
struct complex d[5];
```

Доступ к членам:

```
complex c,*b=&c;  
a.re=0.; d[1].im=3.;  
*b.re=1.; b->im=2.;
```

Присваивание:

```
c=*b;
```

Операции доступа к члену
структуры:

- . – через переменную
- > - через указатель

Объединения

Формат описания:

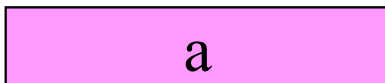
```
union {  
    тип имя_члена;  
    ...  
    тип имя_члена; }  
список_переменных;
```

Пример:

```
union {  
    int i;  
    struct {char h_b,l_b;} j;  
} a;
```

```
a.j.h_b = '\0'; a.j.l_b='1';  
cout << a.i << '\n';
```

Результат: 49



а

Битовые поля – члены структуры или объединения

Формат:

```
struct или union {  
    тип имя: длина;  
    ...  
    тип имя: длина; }  
список_переменных  
;
```

Здесь тип – это
int, signed, unsigned

Пример:

```
struct { int a:4; int b:4;  
        char c; } c;  
    c.a=0; c.b=3;  
    c.c='\0';  
void* ptr=&c;  
cout << *((int*)ptr);
```

0	3	0
---	---	---