Программирование с использованием функций на языке C++

Часть 2

Ссылка

тип &имя_ссылки = имя_переменной;

```
int a = 10;
int &i1 = a;
int& i2 = a;
int & i3 = a;
i3 += 2; // меняем значение a
```

*ссылка — это псевдоним переменной, указатель - адрес

Константные ссылки

```
float t = 5;
const int &r = t;
// компилятор создает в памяти временный объект, равный t
t = 10; // не меняет значение временного объекта
cout << r << " " << t; //5 10
```

Передача параметров в функцию по ссылке (использование ссылки)

```
void partial (int n, int &dec, int &num){
dec = n / 10 % 10;
num = n % 10;
}

int main(){
int n, d, m;
cin >> n;
partial(n, d, m);
cout << "Число десятков - " << d << endl;
cout << "Число единиц - " << m << endl;
}
```

Элементарная функция с ссылочными аргументами

```
void swap (int &x, int &y){
int t;
t = x;
x = y;
y = t;
}
```

Использование константной ссылки в качестве параметра функции

```
void inc (int& x, const int& y){
x += y;
//y++; - ошибка
}
```

Указатель

```
int i = 10, *pi;
pi = &i;
*pi += 2;
```

Передача параметров в функцию по ссылке (использование указателя)

```
void partial (int n, int *dec, int *num){
*dec = n / 10 % 10;
*num = n % 10;
}

int main(){
int n, d, m;
cin >> n;
partial(n, &d, &m);
cout << "Число десятков - " << d << endl;
cout << "Число единиц - " << m << endl;
}
```

Ссылка на указатель и указатель на ссылку

```
int t;
int *p = &t;
int *&rp = p;

const int &cr = *(new int(5));
const int *crp = &cr;
```

Пример функции библиотеки cmath с параметром-указателем

```
double modf(double val, double * intptr);


int main(){
double number, intpart, fracpart, *intpartPtr;
//ввод number;
fracpart = modf (number, &intpart);
//fracpart = modf (number, intpartPtr);
//…
}
```

Использование указателя на константу и константного указателя в качестве параметра функции

```
void inc (int x, const int *y){
x += *y;
//*y = x; - ошибка при изменении значения
y = &x; // изменяем адрес - допустимо
}

void inc (int x, int * const y){
x += *y;
//y = &x; - ошибка при изменении адреса
*y = x; // изменяем значение - допустимо
}
```

Использование одномерного статического массива в качестве параметра функции. Способ 1

```
void onestat(int mas[], int n){
//аналогично void onestat(int mas[3],int n){
//…тело функции…
mas[n - 1] = 20;
}

int main(){
const int n = 3;
int mas[n] = {1, 2, 3};
onestat(mas, n);
}
```

Использование одномерного статического массива в качестве параметра функции. Способ 2

```
void onestat(int *const mas, int n){
//…тело функции…
mas[n - 1] = 20;
}

int main(){
const int n = 3;
int mas[n] = {1, 2, 3};
onestat(mas, n);
}
```

Использование одномерного динамического массива в качестве параметра функции

```
void onedyn(int *mas, int n){
/* функция может работать со статическими и динамическими
 одномерными массивами */
mas[n - 1] = n;
}

int main(){
int n = 3;
const int m = 5;
int *dynmas = new int[n];
int statmas[m];
onedyn(dynmas, n);
onedyn(statmas, m);
}
```

Использование двумерного статического массива в качестве параметра функции. Способ 1

```
const int n = 3;
//void twostat(const int m, int mas[][m]){
void twostat(int mas[][n]){
// аналогично void twostat(int mas[n][n]){
// можно опустить только внешнюю размерность
mas[0][n - 1] = n;
/* mas[0]++; ошибка, попытка изменить указатель mas[0]*/
}

int main(){
int mas[n][n] = {1, 2, 3, 4, 5, 6, 7, 8, 9};
twostat(mas);
}
```

*первую размерность можно не указывать, но вторую обязательно – так программа сможет вычислить, где находится каждый элемент (умножая количество байтов, приходящееся на один элемент, на индекс нужного элемента)

Использование двумерного статического массива в качестве параметра функции. Способ 2

```
const int n = 3;

void twostat(int (*const mas)[n]){
//…тело функции…
mas[0][n - 1] = 20;
}

int main(){
int mas[n][n] = {1, 2, 3, 4, 5, 6, 7, 8, 9};
twostat(mas);
}
```

Использование двумерного статического массива в качестве параметра функции. Способ 3

```
const int n = 3;

void twostat (int * const *const mas){
//…тело функции…
mas[0][n - 1] = 20;
}

int main(){
int mas[n][n] = {1, 2, 3, 4, 5, 6, 7, 8, 9};
twostat(mas);
}
```

Использование двумерного динамического массива в качестве параметра функции

```
void twodyn(int **mas, int n, int m){
/* функция может работать со статическими и динамическими двумерными
  массивами */
mas[n - 1][m - 1] = 20;
}

int main(){
int n = 2, m = 3;
int **dynmas = new int*[n];
for (int i=0; i<n; i++)
   mas[i] = new int [m];
int statmas[2][3];
twodyn(dynmas, n, m);
twodyn(statmas, n, m);
}
```

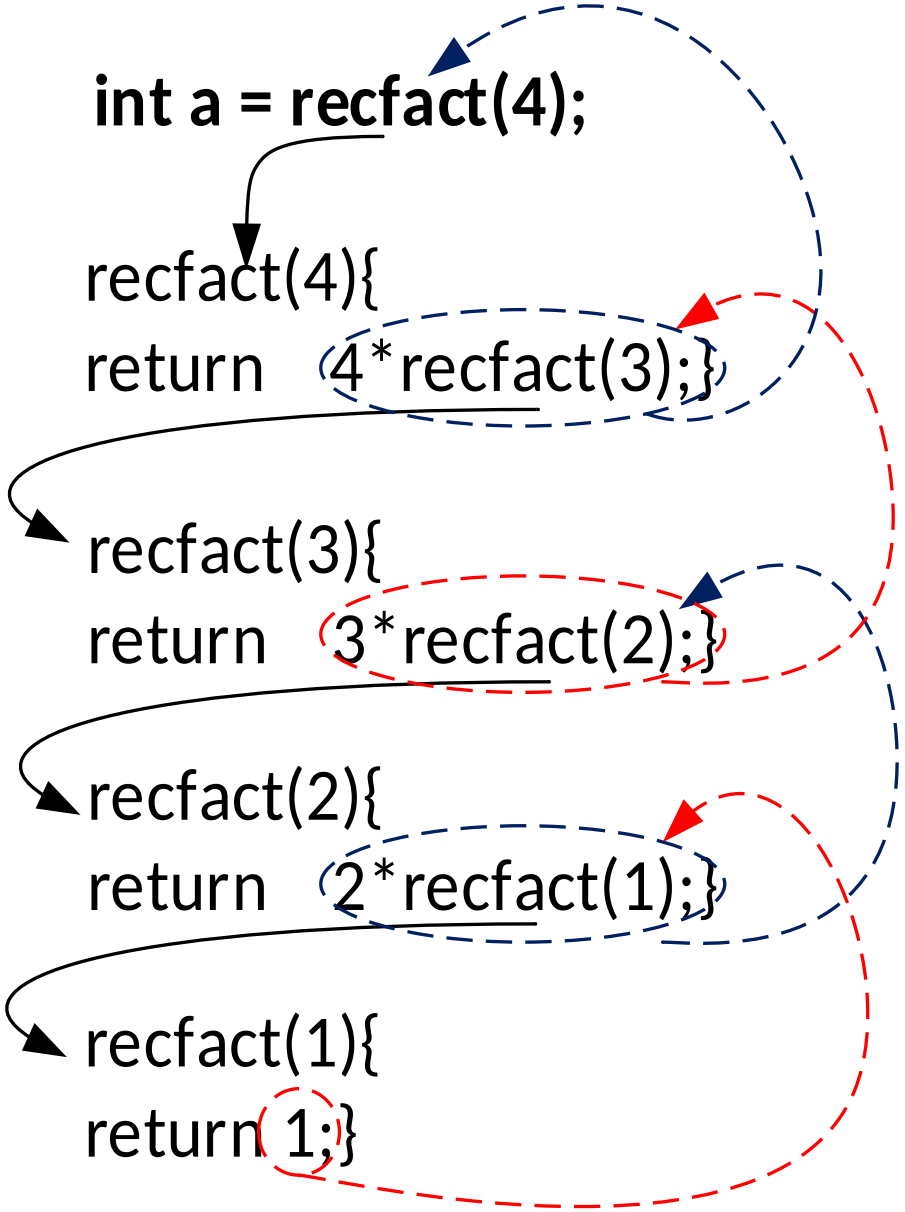
Итерация свойственна человеку,
рекурсия божественна.
— *L. Peter Deutsch*

Рекурсивные функции

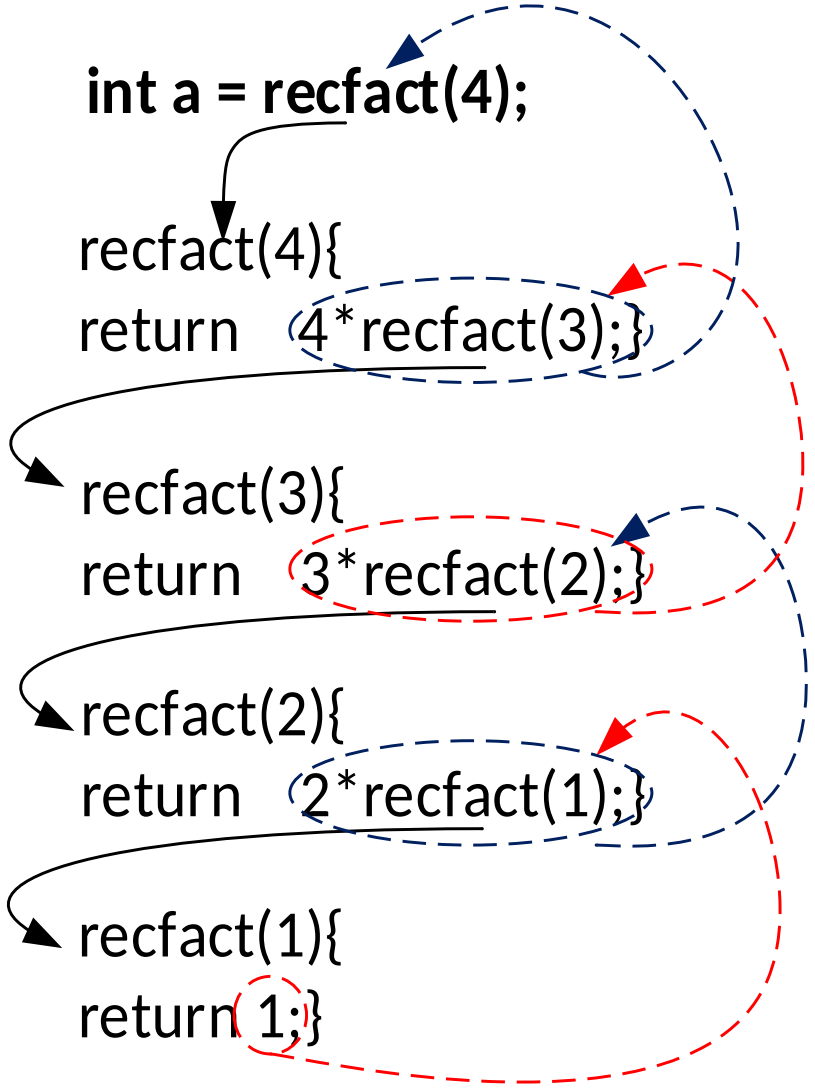
```
long long recfact (long n){
return (n <= 1) ? 1 : n * recfact(n - 1);
/*
if (n <= 1) return 1;
else return (n * recfact(n - 1));
*/
}
```

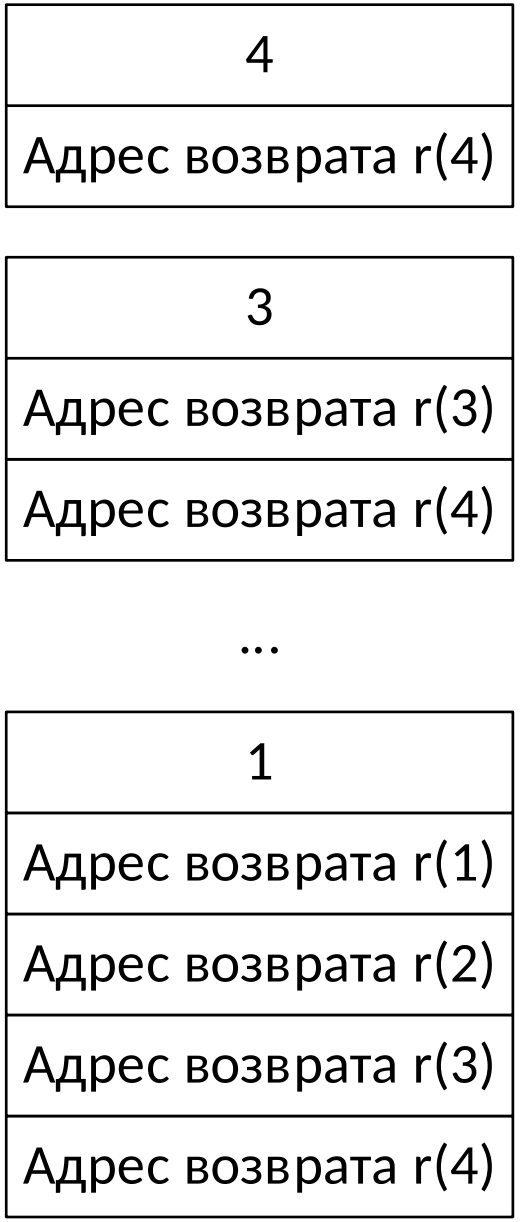
Рекурсивные спуск и подъем

```
long long recfact (long n){
if (n <= 1) return 1;
else return (n * recfact(n - 1));
}
```



Рекурсивная функция на стеке





Виды рекурсий

```
//нисходящая
long long recfact (long n){
if (n <= 1) return 1;
else return (n * recfact(n - 1));
}

//восходящая
long long recfact (long n, long i = 1){
if (i < n) return (i * recfact(n, i + 1));
else return n;
}
```

Пример рекурсивной функции - Число делителей

```
unsigned dividersCount(unsigned i, unsigned num){
if (i < 1)
     return 0;
bool is = num % i;
return (!is + dividersCount(i - 1, num));
}

int main(){
dividersCount(10, 10);
}
```

Пример рекурсивной функции - Сумма делителей

```
unsigned dividersSum(unsigned i, unsigned num){
if (i < 1)
     return 0;
bool is = num % i;
return (!is * i + dividersSum(i – 1, num));
}
```

Пример рекурсивной функции - Наибольший общий делитель

```
int nod(int m, int n){//алгоритм Евклида
if (m % n == 0)
   return n;
return nod(n, m % n);
}
```

Косвенная рекурсия

```
int x, y;
int sumJ(int i, int j);
int sumI(int i);

int sumI(int i){
if (i <= x)
   return i + sumJ(i, 1);
else
   return 0;
}

int sumJ(int i, int j){
if (j <= y){
   return i + j + sumJ(i, j + 1);
   }
else
   return sumI(i + 1);
}
```

$$\sum_{i=1}^{x}\left(i+\sum_{j=1}^{y}(i+j)\right)$$

Модификаторы класса памяти auto и register

- auto;
- register;

```
int main(){
int a;// аналогично auto int a;
register float b;
}
```

Модификаторы класса памяти static для локальных переменных

- static;

```
int sum (int x, int y){
static int count = 0;
cout << "Вы вызвали эту функцию "    << ++count << " раз";
return (x + y);
}
```

Еще пример локальной static переменной

```
float avg(float value);

int main(){
float x, y;
cin >> x >> y;
avg(x);
cout << avg(y);
}

float avg(float value){
static float sum = 0;
static int count = 0;
count++;
sum += value;
return sum / count;
}
```

Модификация функции поиска числа делителей (1 аргумент)

```
unsigned dividersCount(unsigned i){
static int num = i;
if (i < 1) return 0;
bool is = num % i;
return (!is + dividersCount(i - 1));
}
```

Модификация функции вычисления суммы делителей (1 аргумент)

```
unsigned dividersSum(unsigned i){
static int num = i;
if (i < 1) return 0;
bool is = num % i;
return (!is * i + dividersSum(i - 1));
}
```

Модификатор volatile

Объект объявляется как volatile (неустойчивый, асинхронно изменяемый), если его значение может быть изменено незаметно для компилятора, например переменная, обновляемая значением системных часов.

Модификатор сообщает компилятору, что не нужно производить оптимизацию кода для работы с данным объектом.

Синтаксис описания указателя на функцию

```
тип имя_функции(список_параметров){
тело_функции
}
тип (*имя_указателя)(список_параметров);

имя_указателя = &имя_функции;

имя_указателя(список_значений);
```

Пример создания указателя на функцию

```
int func(float x, bool f){
//тело_функции
}

int (*pFunc)(float, bool);

int main(){
pFunc = &func;
//аналогично pFunc = func;
int a = pFunc (1.5, true);
//аналогично (*pFunc)(1.5, true);
}
```

Указатель на функцию (простой пример)

```
int func1(float x, bool f){return 1;}
int func2(float x, bool f){return 2;}
int (*pFunc)(float, bool);

void funcUsingPFunc(float a1, bool a2){
pFunc(a1,a2);
}

int main(){
if (true)
   pFunc = func1;
else
   pFunc = func2;
funcUsingPFunc(1.5, true);
}
```

Указатель на функцию (более оптимальный пример)

```
int func1(float x, bool f){return 1;}
int func2(float x, bool f){return 2;}
void funcUsingPFunc(int (*pfunc)(float, bool), float a1, bool
 a2){
pFunc(a1, a2);
}

int main(){
if (true)
   funcUsingPFunc(func1, 1.5, true);
else
   funcUsingPFunc(func2, 1.5, true);
}
```

Указатель на функцию (оптимальный и читаемый пример)

```
typedef int (*PFunc)(float, bool);

int func1(float x, bool f){/*тело_функции*/}

int func2(float x, bool f){/*тело_функции*/}

int funcUsingPFunc(PFunc p, float f, bool b){
return p(f,b);
}

int main(){
cout << (true ? funcUsingPFunc(func1,1.5, true)
              : funcUsingPFunc(func2,1.5, true) );
}
```

Использование массива указателей на функции

```
typedef int(*pfunc)(int);

int func1(int i) { return i+1; }
int func2(int i) { return i+2; }
int func3(int i) { return i+3; }

int main(){
pfunc arr[3] = {func1, func2, func3};
for (int i = 0; i < 3; i++)
cout << arr[i](i);
}
```

Функции с переменным числом аргументов

тип имя_функции(список_аргументов,...){

/*в теле функции обращение к списку аргументов через макросы va_start, va_arg, va_end

или используя адрес первого известного параметра и арифметику указателей */

}

```
#include <stdarg.h>;

void va_start(va_list ap, lastfix);
type va_arg(va_list ap, type);
void va_end(va_list ap);
```

Пример эффективного построения программы с использованием массива указателей на функцию

```
// случайное заполнение значений ячеек массива
void rFill  (float* arr, unsigned int cols);
// вывод содержимого массива на экран консоли
void print(float* arr, unsigned int cols);
// циклический сдвиг элементов массива влево на 1 ячейку
void shift(float* arr, unsigned int cols);


int main(){
…
typedef void (*fPtr)(float*, unsigned int);
fptr fArr[4] = {&rFill, &print, &shift, &print};
for(int i = 0; i < 4; i++)
     fArr[i](arr,cols);
}
```