Функции и другие дополнения

Лекция 7

Перегрузка функций

- typedef int myint;
- void f(myint g){
- cout<<g;
- }
- void f(int g){
- cout<<g;
- }
- int main(){
- myint t=4;
- f(t);
- }

Формы рекурсивных функций

- 1. С выполнением действий на рекурсивно спуске
- void func();
  { S;
  if (условие) func();
  }
- 2. С выполнением действий на рекурсивном возврате:
  void func();
  { if (условие) {func(); S;}
  }
  3. С выполнением действий как на рекурсивном спуске, так и на рекурсивном возврате:
  void func();
  { S1;
  if (условие) {func(); S2;}
  }
  void func();
  { if(условие) {S1; func(); S2;}
  }

Виды рекурсий

- Прямая рекурсия – функция содержит вызовы самой себя. Косвенная рекурсия – функция вызывает себя через вызов в другой функции

Встроенные функции inline

- Вызов функции заменяется ее содержимым(но увеличивается компилируемый код)
- Используют для коротких функций
- Компилятор может отклонить запрос…(например, если функция длинная)
- Компилятор автоматически конвертирует в inline…
- -> нет необходимости в использовании inline
- Замечание: не действует правило не определять в заголовочных файлах…!

Встроенные функции

```
void f()
{
   cout<<5;
}
int main()
{
   f();
   f();
}
```

```
inline void f()
{
   cout<<5;
}
//при компиляции код:
int main()
{
   cout<<5;
   cout<<5;
}
```

Перегрузка функций

- Определение функций с одним именем, но разными параметрами
- Тип возврата не учитывается при перегрузке:
- int f(){
- int b=4;
- return b;
- }
- double f(){
- double c =8.0;
- return c;}
- int main(){
- int c=f();
- cout<<c;}

Перегрузка функций

```
typedef int myint;// псевдоним типа
void f(myint g){
cout<<g;
}
void f(int g){
    cout<<g;
}
int main(){
myint t=4;
f(t);
 }
```

Параметры по умолчанию

- void f(int t, int p=34);
- int main(){
- f(6);
- }
- void f(int t, int p=34){ //проблема…
- cout<<t<<"_"<<p;
- }

Параметры по умолчанию

- int g=5;
- int sqr(int x)
- {return x*x}
- void f(int t, int p=(++g, sqr(g)));
- int main(){
- f(6);
- }
- void f(int t, int p){ //проблема…
- cout<<t<<"_"<<p;
- }

Указатели на функции

- Псевдонимы типов
- using ff=int(*)(int,int);
- //typedef int(*ff)(int,int);
- int plus(int x, int y){
- return x+y;
- }
- int mult(int x,int y){
- return x*y;
- }
- int myop(int x, int y, ff f){
- return (*f)(x,y);
- }
- int main(){
- int a=2,b=4;
- cout<<myop(a,b,mult)<<endl;
- }

Указатели на функцию

```
#include <functional>
using namespace std;
int plus(int x, int y){
return x+y;
}
int mult(int x,int y){
return x*y;
}
int myop(int x, int y, std::function<int(int,int)> f){
   return f(x,y);
}int main(){
int a=2,b=4;
cout<<myop(a,b,mult)<<endl;
}
```

Аргументы командной строки

- Доступ к аргументам командной строки из программы
- int main(int argc, char *argv[]);
- int main(int argc, char **argv);
- argc – argument count(кол-во аргументов переданных в программу: имя программы – это первый аргумент);
- argv-argument values (значения аргументов)
- Аргументы командной строки передаются как строки

assert

- Макрос препроцессора для обработки условного выражения во время выполнения
- #include <cassert>
- using namespace std;
- void f(int i){
- assert(i>=5 && i<10); //программа завершится при ложном условии
- cout<<i;}
- int main(){
- f(24);
- }
- Удобно использовать в функциях для проверки параметров и возвращаемого значения

assert

- #include <cassert>
- using namespace std;
- void f(int i){
- assert((i>=5 && i<10) && " error i"); //программа завершится при ложном условии
- cout<<i;}
- int main(){
- f(56);
- }
- Обычно assert не включают в релиз
- #define NDEBUG отключит все assert

static_assert

- Вызывает ошибку при ложном условии во время компиляции
- #include <cassert>
- using namespace std;
- int main(){
- const int t= 5;
- static_assert(t==7, " Error t");
- cout<<t;
- }

exit

- Если необходимо завершить программу, то используйте exit()
- // Example program
- #include <iostream>
- #include <string>
- #include <cstdlib>
- using namespace std;
- void f(int t){
- if (t>4) exit(0);
- cout<<t;}
- int main(){
- f(7);
- }

Плохая функция, возвращающая ссылку

- int & func(int i)
- { return i;}
- int & func1(int &i)
  { return i;}
- int main()
- {
- int p= 5;
- cout<<func1(func(p));
- }
- Замечание: возврат ссылки на статическую переменную

Левые функции

```
// Example program
#include <iostream>
using namespace std;
int &mymin(int n, int *a){
   int id=0;
for(int i=1;i<n;i++)
   id=a[id]<a[i]?id:i;
   return a[id];
}
int main(){
int a[]{4,7,3,9,12};
mymin(5, a)=100;
cout<<a[2];
}
```

Левые функции

```
// Example program
#include <iostream>
using namespace std;
int *mymin(int n, int *a){
   int id=0;
  for(int i=1;i<n;i++)
    id=a[id]<a[i]?id:i;
    return &a[id];
}
int main(){
int a[]{4,7,3,9,12};
*mymin(5, a)=100;
cout<<a[2];
}
```

Левые функции. Маленькая задача

- Написать функцию, возвращающую указатель на минимум из двух чисел …
- C++ позволяет запретить использовать функцию как левую: const…

Функции с переменным числом параметров

- Тип имя_функции(аргументы…)
- Необходимо разработать алгоритм доступа к параметрам…

Функции с переменным числом параметров

```
double f(double n,...)//считаем что 0 в конце
{    double *p=&n;//настроили на начало списка
     double sum=0;
     double count=0;
  while(*p)    {
     sum+=(*p);
    p++;
     count++;    }
  return ((sum)?sum/count:0.0);
   cout<<sum<<endl;
 cout<<count<<endl;}
int main() {
double t = f(4.0,8.0,3.0,10.0,0);
   cout<<t;}
```

Функции с переменным числом параметров

```
using namespace std;
double f(int n,...)
{    int *p=&n;//настроили на начало списка
     p++; //настраиваем на тип double
 double *pp=(double*)p;    double sum=0;
double count =n;
    for(;n--;pp++)
   {  sum+=(*pp);
   }
return ((sum)?sum/count:0.0);
 }
int main() {
 double t = f(4.0,8.0,3.0,10.0,0);
    cout<<t;
}
```

Функции с переменным числом параметров

```
using namespace std;
double f(double *n,...)//считаем что нулевой указатель в конце
{
double **p=&n;//адрес адреса
double sum=0;
double count=0;
while(*p!=nullptr){
 sum+=(**p);
 p++;
 count++;
}
 return ((sum)?sum/count:0.0);
}
int main() {
double a=8.0, b=4.0;
double t = f(&a,&b,nullptr);
cout<<t;
}
```

Функции с переменным числом параметров. Макросы

- stdarg.h
- va_list определяем тип указателя
- va_start устанавливает указатель типа va_list на явный параметр
- va_arg перемещает указатель на следующий параметр
- va_end обнуляет указатель

Функции с переменным числом параметров. Макросы

- Как работает:
- Обьявляем объект типа va_list – как указатель…
- Инициализируем указатель с помощью va_start. Va_list связывается с последним явным параметром (перед многоточием)
- Передвигаемся с помощью va_arg по списку параметров. Тип параметров надо явно указать.
- После всей обработки вызывается va_end

Функции с переменным числом параметров. Макросы

```
// Example program
#include <iostream>
#include <string>
#include <stdarg.h>
using namespace std;
double f(int n, double a,...)//считаем что 0 в конце
{
va_list p;//указатель обьявлен
double sum=a;
double count=0;
 va_start(p, a);// инициализировали указатель
 while(n--)  {
 sum+=va_arg(p, double);
      count++;
}
va_end(p);
return ((sum)?sum/count:0.0);}
int main() {
double t = f(3, 4.0,8.0,3.0);
 cout<<t;}
```

Функции с переменным числом параметров. Макросы

```
#include <stdarg.h>
using namespace std;
double f(int n, double a,...)
{    va_list p;//указатель обьявлен
     double sum=a;
     double count=0;
     va_start(p, a);// инициализировали указатель
    while(n--)    {
        sum+=va_arg(p, double);
        count++;
     }
 va_end(p);
 return ((sum)?sum/count:0.0);}
int main() {
double t = f(3, 6.0,7.0,5.0);
cout<<t;
}
```

Функции с переменным числом параметров. Макросы

```
// Example program
#include <iostream>
#include <string>
#include <stdarg.h>
using namespace std;
double f(double a,...)//считаем что 0 в конце
{   va_list p;//указатель обьявлен
   double sum=0;
   double count=0;
   double k=a;
   va_start(p, a);// инициализировали указатель
   do   {      sum+=k;      count++;      }
   while(k=va_arg(p,double));
   va_end(p);      return ((sum)?sum/count:0.0);}
   int main() { double t = f(6.0,7.0,5.0,0.0);
   cout<<t;}
```