7 Средства создания универсальных подпрограмм C++

7.1 Использование «структурных » типов в качестве формальных параметров

7.1.1 Параметры - многомерные массивы

Как было показано в главе 4, массивы можно использовать для передачи данных в подпрограммы. Однако, наличие контроля за его размерами, ограничивает применение многомерных массивов в таком описании.

Кроме того, иногда в подпрограмме нужно сформировать новый массив, размер которого заранее не известен, а определяется во время работы, и вернуть его из подпрограммы.

C++ позволяет применять вспомогательные массивы указателей на одномерные массивы, которые в свою очередь могут быть массивами указателей.

В этом случае, по каждой размерности массив является одномерным и по правилам C++ его размерность может быть опушена в спецификации формальных параметров.

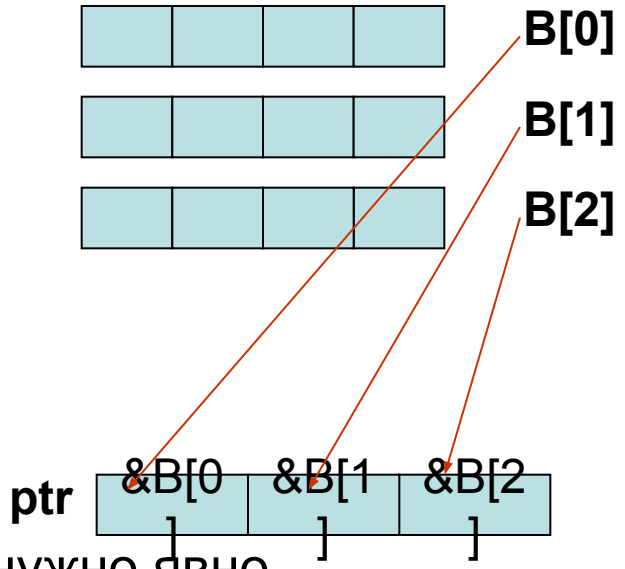
Такой подход позволяет в теле функции обрабатывать многомерные массивы с изменяющимися размерами.

Многомерные массивы (2)

Пример обработки матрицы с переменными размерами.

Написать программу переформирования матрицы путем замены в ней всех отрицательных элементов нулевыми с использованием подпрограмм.

```
float B[3][4]={1.2,-4.9 ,5.0,-8.1,
               -3,6.1,-8.5,9.6,
               3.3,-6.7,-1.2,7.8};
float *ptr[]={&B[0],&B[1],&B[2]};
```



Однако следует помнить, что адрес начала строки не типизированный и в программе его нужно явно преобразовать к типу float : **(float *)&B[0]**

Прототип функции переформирования:

```
void pereform(int n,int m, float *p[]);
```

Пример обработки матрицы переменного размера (2)

```
// Ex7_1.cpp
#include "stdafx.h"
#include <stdio.h>
void pereform(int n,int m,float * p[])
{  for(int i=0;i<n;i++)
   for(int j=0;j<m;j++)
     if (p[i][j]<0)
      p[i][j]=0;
}
int main(int argc, char* argv[])
{float B[3][4]={1.2,-4.9 ,5.0,-8.1,
                -3,6.1,-8.5,9.6,
               3.3,-6.7,-1.2,7.8};
float *ptr[]={(float*)&B[0],(float
  *)&B[1],(float *)&B[2]};
```

Параметр – массив указателей на строки матрицы

Исходная матрица размером 3X4

Формирование массива указателей на строки матрицы

Пример передачи матрицы переменного размера (3)

```
pereform(3,4,ptr);
puts("RESULT MATRIX");
for(int i=0;i<3;i++)
  { for(int j=0;j<4;j++)
printf("%5.2f",B[i][j]);
   printf("\n");
}
return 0;
}
```

Вызов функции переформирования

Печать результата

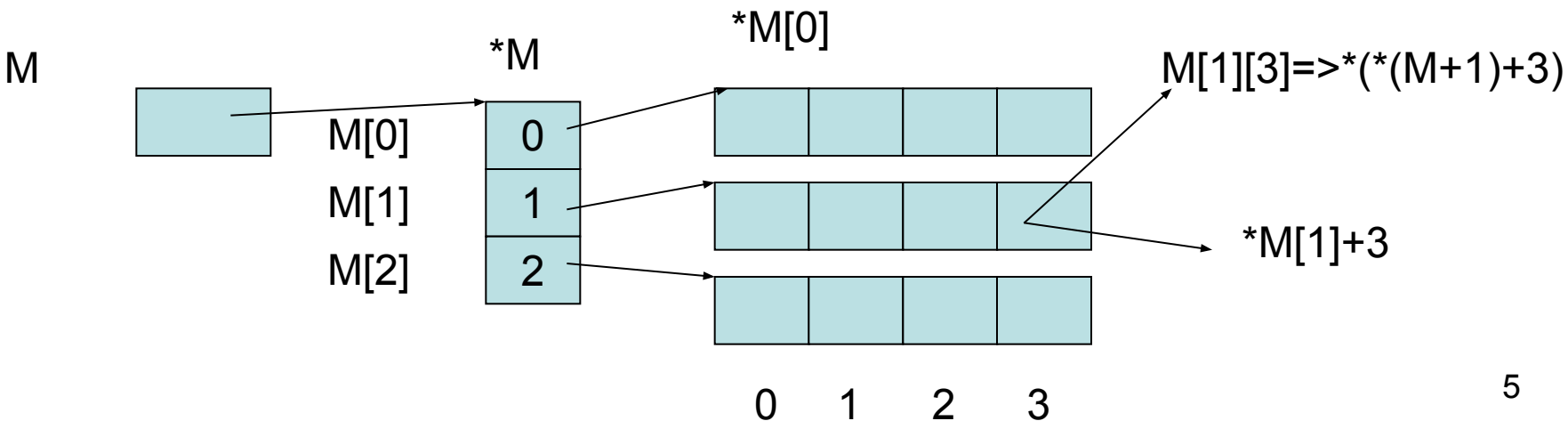
Пример формирования матрицы переменного размера

Пример. Написать программу, формирующую матрицу переменного размера в одной подпрограмме, а в другой – меняет отрицательные элементы этой матрицы на их абсолютное значение.

int **M;

Указатель **int **M** указывает на массив указателей **int *M**, каждый из элементов которого, в свою очередь адресует одномерный массив элементов целого типа.

Так как размеры массивов нигде не указаны и память под массивы не выделена, то все это можно сделать в программе во время выполнения, когда размеры массива становятся известны.



Пример формирования матрицы переменного размера (2)

```
// Ex7_2.cpp
#include "stdafx.h"
#include <stdio.h>
#include <stdlib.h>

int **matr(int &l,int &p)
{int **m;
 int i,j;
 printf(" input size of massiv    \n");
 scanf("%d %d",&l,&p);

 printf(" input %4d strok iz %4d elementov\n",l,p);
 m=new int* [l];
 for (i=0;i<l;i++)
 { m[i]=new int[p];
   for (j=0;j<p;j++)
     scanf("%3d",*(m+i)+j);}
 return m;
}
```

Функция, формирования матрицы целого типа и возвращающая указатель на эту матрицу (указатель на указатель)

Локальная переменная указатель на указатель целого типа, используемый для формирования матрицы

Выделение памяти под массив указателей на строки

Выделение памяти под строку

Обращение к элементу матрицы

Пример формирования матрицы переменного размера (3)

```
void sortmas(int **m,int n,int l)
{int i,j;
 for(i=0;i<n;i++
   for(j=0;j<l;j++)
    if (m[i][j]<0) m[i][j]=abs(m[i][j]);
  }
```

Формальный параметр указатель на указатель для передачи в подпрограмму адреса матрицы

```
int main(int argc, char* argv[])
{int n,l,**mat,i,j;
mat=matr(n,l);
```

Вызов функции формирования матрицы mat

```
printf("\n ===== inputed massiv ==== \n");
for(i=0;i<n;i++)
 for(j=0;j<l;j++)
  printf("%4d%c",mat[i][j],(j==l-1)?'\n':' ');
```

Печать сформированной матрицы

Пример формирования матрицы переменного размера (4)

```
sortmas(mat,n,l);
```

Вызов подпрограммы преобразования матрицы

Фактический параметр – имя матрицы

```
printf("\n sorted  massiv\n");
for(i=0;i<n;i++)
 for(j=0;j<l;j++)
  printf("%4d%c",mat[i][j],(j==l-1)?'\n':' ');
```

Печать переформированной матрицы

Удаление матрицы

```
for (i=0;i<n;i++)
 delete [] mat[i];
delete [] mat;
  return 0;
}
```

Сначала удаляются строки

Затем удаляется массив указателей

7.1.2 Параметры - строки

При программировании функций работающих со строками обычно используют прием, принятый в стандартных функциях обработки строк.

Этот прием заключается в том, что такие функции пишут так, чтобы их можно было вызывать и как процедуры, и как функции.

Рассмотрим несколько примеров.

Пример1. (Ex7_3)Написать подпрограмму удаления «лишних» пробелов.

Описание заголовка функции:

```
char * strdel(const char * tstring, char * trez)
```

Дубликат адреса результата

Исходная строка

Строка результат

```
strcpy(trez,tstring);
while((ptr=strstr(trez,"  "))!=NULL)
strcpy(ptr,ptr+1);
return trez;}
```

Тело функции

Параметры – строки (2)

Вызов функции **strdel** в основной программе:

```
int main(int argc, char* argv[])
{char st[40],st2[40],*ptr2;
puts("input string : world and space");
gets(st);
puts("isxodnaya stroka");
puts(st);
strdel(st,st2);
puts("Result string 1");
puts(st2);
printf("Result string 2:\n");
ptr2=new char [40];
puts(strdel(st,ptr2));
  return 0;
}
```

Определение переменных

Вызов подпрограммы как процедры

Выделение памяти под результат

Вызов подпрограммы как функции

Параметры – строки (3)

Пример 2. Написать подпрограмму нахождения максимального слова строки.(**Ex7_4.cpp**) .

```
char * maxworld(const char * s,char* slmax)
{char slovo[10];
unsigned int i,j,dls,maxl;
dls=0;slmax[0]='\0';maxl=0;j=0;
for(i=0;i<=strlen(s);i++)
{if ((s[i]==' ')||(s[i]=='\0'))
 {slovo[j]='\0';
  if (dls>maxl){ maxl=dls;
   strcpy(slmax,slovo);}
  slovo[0]='\0';
  j=0;
  dls=0;}
  else {dls++;
   slovo[j++]=s[i];}
} return slmax;}
```

Заголовок функции
Параметры – исходная строка и максимальное слово

Если это конец очередного слова, то проверяем его длину

Если текущее слово длиннее максимального, то сохраняем его и его длину

Переходим к следующему слову, обнуляя все вспомогательные данные

Если слово не закончено, то накапливаем его длину и само слово

Параметры – строки (4)

```
int main(int argc, char* argv[])
{char st[80],maxsl[10];
puts("input string : world and space");
gets(st);
printf("V stroke slovo ");
printf("""%s"" - macsimalno \n",maxworld(st,maxsl));
puts("input string : world and space");
gets(st);
maxworld(st,maxsl);
printf("V stroke slovo ");
printf("""%s"" - macsimalno \n", maxsl);
   return 0;
}
```

Вызов подпрограммы как функции

Вызов подпрограммы как процедуры

Параметры – строки (5)

Пример 3. Написать подпрограмму нахождения максимального слова строки, его длины и номнра в строке.(**Ex7_4a.cpp).**

```
char * infmaxw(const char * s,char* slmax,int & maxl,int
   & maxnum)
{char slovo[10];int i,j,kols,dls;
kols=0;dls=0;slmax[0]='\0';maxl=0;maxnum=0;j=0;
for(i=0;i<=strlen(s);i++)
{ if ((s[i]==' ')||(s[i]=='\0'))
    {kols=kols+1;
     slovo[j]='\0';
     if (dls>maxl)
   { maxl=dls;
     maxnum=kols;
     strcpy(slmax,slovo);}
     slovo[0]='\0';
   j=0; dls=0;}
  else {dls++;
        slovo[j++]=s[i];}
}
       return slmax;}
```

Заголовок подпрограммы

slmax – максимальное слово, возвращается в оп
maxl – длина максимального слова, возвращается в оп по ссылке
maxnum –номер максимального слова, возвращается в оп по ссылке

dls – длина текущего слова
kols – счетчик количества слов

7.1.3 Параметры структуры

В отличие от массивов и строк, имя структуры не является указателем, поэтому для передачи в подпрограмму параметров типа структуры, которые должны передаваться по адресу, необходимо использовать ссылки или указатели.

Пример. Дан массив целых чисел на 10 элементов. Объединить данные о массиве в структуру massiv, содержащую 3 поля:

массив, его текущий размер и сумму.

Написать подпрограмму, получающую структуру massiv в качестве параметра, вычисляющую сумму элементов массива и возвращающую эту структуру, как результат, с вычисленной суммой элементов.

Реализовать передачу в подпрограмму параметр структуру можно с использованием указателя, а можно описать его как ссылку.

Результат будет одинаков, а вот синтаксис описания и вызова подпрограммы будут отличаться.

7.1.3.1 Использование указателя

Сумма элементов массива . Подпрограмма проектируется с возможностью вызова ее как процедуры, и как функции.

Описание структуры massiv

```
struct mas{int n; int a[10]; int s;} massiv;

int summa(struct mas *x)
{ int i,s=0;
  for(i=0;i< x->n;i++) s+=x->a[i];
  x->s=s;
  return s;
}
```

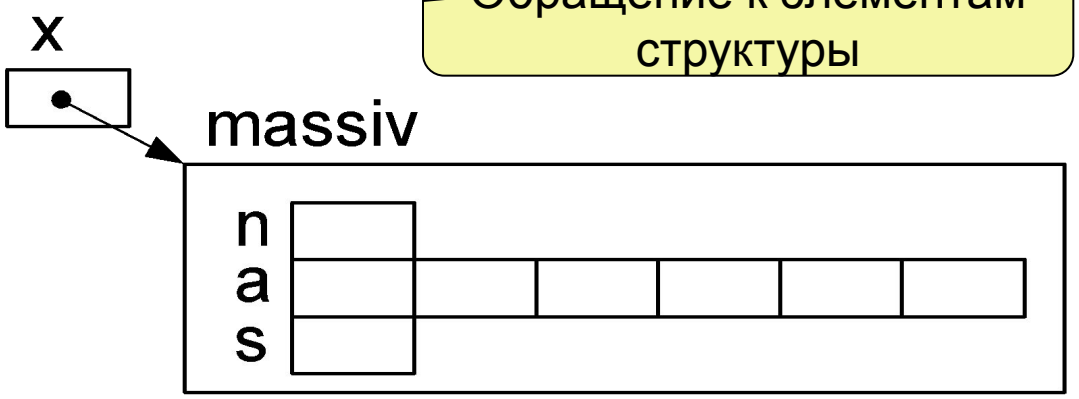
Формальный параметр – указатель на структуру

Накопление суммы

Обращение к элементам структуры

Вызов:

```
summa(&massiv);
```



7.1.3.2 Использование ссылки

Сумма элементов массива.

Описание структуры massiv

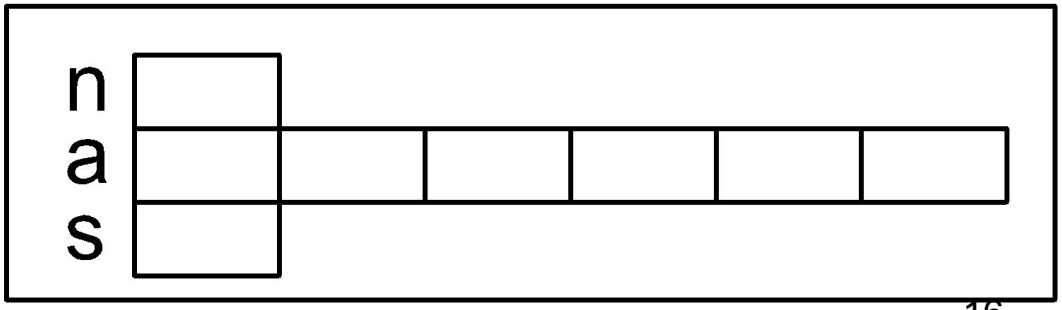
```
struct mas{int n; int a[10]; int sum;}
  massiv;

int summa(struct mas &x)
{ int i,s=0;
  for(i=0;i< x.n;i++) s+=x.a[i];
  x.s=s;
  return s;
}
```

Формальный параметр – ссылка на структуру

Обращение к элементам структуры

x=massiv



Вызов:

```
summa(massiv);
```

7.1.3.3 Применение массива структур

Сумма элементов массива структур.

Описание массива структур

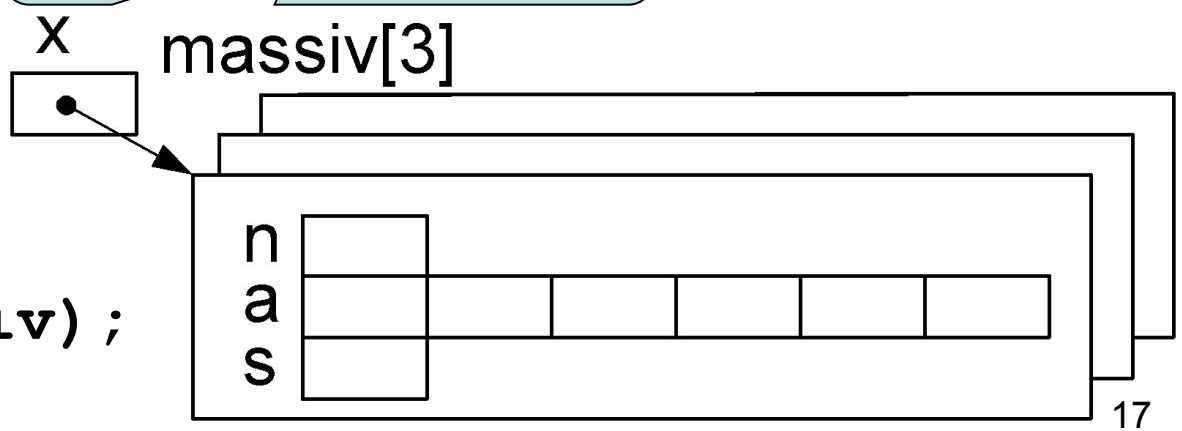
```
struct mas{int n;int a[10];int sum;} massiv[3];

int summa(struct mas *x)
{ int i,k,s,ss=0;
  for(k=0;k<3;k++,x++)
    { for(s=0,i=0;i<x->n;i++) s+=x->a[i];
      x->s=s;
      ss+=s;
    }
  return ss;
}
```

Формальный параметр – указатель на массивна структур

Сумма элементов одного массива

Сумма всех элементов массива структур



Вызов: **summa(massiv);**

7.2 Параметры функции

Как уже отмечалось, функция характеризуется типом возвращаемого значения, именем и сигнатурой.

Сигнатура определяется количеством, порядком следования и типами параметров.

При использовании имени функции без последующих скобок и параметров, имя функции выступает в качестве указателя на эту функцию, и его значением служит адрес размещения функции в памяти.

Это значение адреса может быть присвоено другому указателю, и затем уже этот новый указатель можно применять для вызова функции.

Однако в определении нового указателя должен быть тот же тип, что и возвращаемое функцией значение, и та же сигнатура.

Указатель на функцию определяется:

<тип_функции>(* <имя>)(<спецификация_параметров>);

Например: **int (*ptrfun)(int,int);**

При определении указатель на функцию может быть инициализирован, но в качестве значения должен быть адрес функции, тип и сигнатура которой соответствуют определяемому указателю.

Параметры функции (2)

При присваивании указателей на функции тоже надо следить за соответствием типов возвращаемых значений и сигнатур правой и левой частей операции присваивания.

Пример.

```
char f1(char){…}
char f2(int){…}
void f3(float){…}
int f4(float){…}
int f5(int){…}
void (*ptr1)(float)=f3;
int (*ptr2)(int);
char (*ptr3)(int);
void main ()
{ ptr2=f5; ptr3=f2;
 prt2=f4; ptr3=f1;
}
```

Описание функций

Инициализированный указатель

Определение указателей на функции

Корректное присвоение указателя на функцию

Ошибка присвоения. Несоответствие типов или сигнатур

Параметры функции (3)

Пример (Ex7_6). Написать программу вычисления элементарных функций.

```
#include "stdafx.h"
#include <stdio.h>
int add(int n,int m) {return n+m;}
int sub(int n,int m) {return n-m;}
int mul(int n,int m) {return n*m;}
int div(int n,int m) {return n/m;}
int main(int argc, char* argv[])
{ int (*ptr)(int,int);
  int a=6, b=2; char c='+';
  while (c!=' ')
   {  printf("%d%c%d=",a,c,b);
      switch (c)  { case '+': ptr=add; c='-';break;
                    case '-': ptr=sub; c='*';break;
                    case '*': ptr=mul; c='/';break;
                    case '/': ptr=div; c=' ';  }
      printf("%d\n",a=ptr(a,b));   }
  return 0;  }
```

6+2=8
8-2=6
6*2=12
12/2=6

Указатель на функцию

Присвоение значения указателю

Вызов функции по указателю

Параметры функции (4)

Пример2. Написать программу вычисления значения интеграла функции одной переменной на отрезке a,b с точностью eps.

```
// Ex7_7.cpp
#include "stdafx.h"
#include <stdio.h>
#include <math.h>
float (* funuk)(float);
float integral(float(*funuk)(float),
float a,float b,float eps)
{int i,n,k;
float s1,s2,x,d;
n=5;
d=(b-a)/n;
s2=1.0e+10;
k=0;
```

Указатель на функцию одной переменной вещественного типа

Заголовок функции нахождения интеграла функции одной переиенной

Установка начальных значений

Формальный параметр – указатель на функцию

Параметры функции (5)

```
do
{s1=s2;
s2=0;n=n*2;
d=d/2;
x=a;k++;
for(i=1;i<=n;i++)
{  s2=s2+funuk(x);
    x=x+d;
 }
  s2=s2*d;

} while(fabs(s2-s1)>eps);
 return s2;
}
float f1(float x)
{return x*x-1;}
float f2(float x)
{return 2*x;}
```

Цикл расчета интеграла

Функции, для которых ищется интеграл

Параметры функции (6)

```
int main(int argc, char* argv[])
{float a,b,eps;
puts("input a,b,eps for y=x^2-1");
scanf("%f %f %f",&a,&b,&eps);
printf("Value integral= %10.5f\n",integral(f1,a,b,eps));
puts("input a,b,eps for y=2*x");
scanf("%f %f %f",&a,&b,&eps);
printf("Value integral= %10.5f\n",integral(f2,a,b,eps));
   return 0;
}
```

Ввод исходных данных

Ввод исходных данных

f2,f1- параметры функции

Вызов функции с параметром функцией