Введение в C++

Структуры, препроцессор, динамическая память

Структуры, препроцессор, динамическая память

Структуры

Формат: struct <имя-структуры> { <список-полей> } ;

struct <имя-структуры> <описатели-переменных> ;

struct [<имя-структуры>] { <список-полей> } <описатели-переменных> ;

Аналоги структур в реальной жизни – различные формуляры, анкеты, учетные карточки.

Существует два способа доступа к полям структурных переменных:

<имя-структурной-переменной> **.** <имя-поля>

<указатель-на-структурную-переменную> **->** <имя-поля>

Пример: электронная картотека для домашней библиотеки

```
struct form {
        char *ptitle;          наименование книги
        char *pauthor;         автор
        float price;           стоимость
              } ;
struct form books [ 1024 ] ;        картотека есть массив структур
struct form  *pbook = books ;       указатель на массив структур
```

Структуры, препроцессор, динамическая память

Разработаем getprice – функцию для поиска книги в картотеке по её названию и выдачи стоимости книги.

Вариант 1 – используем индексы

```
float getprice (unsigned size, struct form lib[], char *s )
{
        while ( size ) {   size--;
            if ( strcmp (s, lib[size].ptitle) == 0 )  return lib[size].price ;
                    }
        return 0. ;
}
```

Вариант 2 – используем адресную арифметику

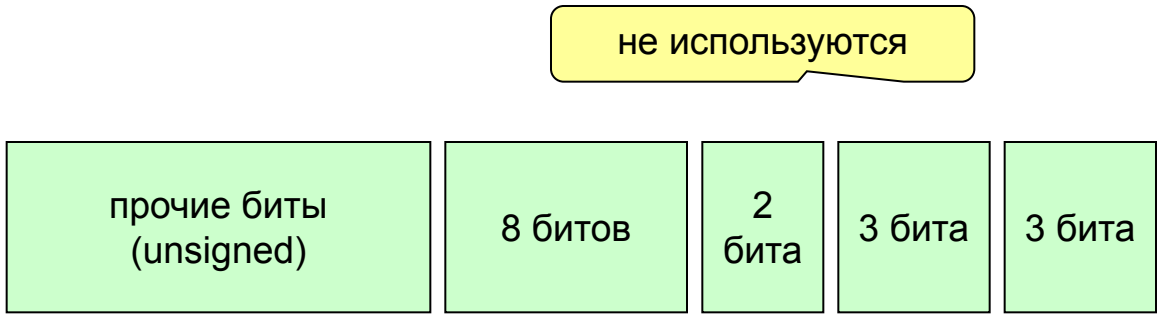
```
float getprice (unsigned size, struct form *pbook, char *s )
{
        while ( size-- ) {
            if ( strcmp (s, pbook -> ptitle) == 0 )  return pbook -> price ;
            pbook++;    }
        return 0. ;
}
```

Структуры, препроцессор, динамическая память

Битовые поля

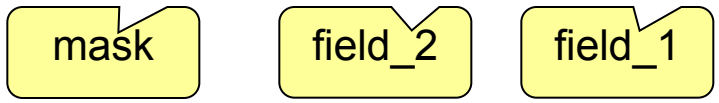
Рассмотрим структуру специального вида:

```
struct {
    unsigned field_1 : 3 ;
    unsigned field_2 : 3 ;
    unsigned         : 2 ;
    unsigned mask  : 8 ;
      } fields ;
```



Работа с битовыми полями по именам:

```
fields.field_1 = 0;
fields.field_2 = 07;
fields.mask ^= fields.mask;
```



Именованные битовые поля используются для экономии памяти и для работы с аппаратными регистрами.

Структуры, препроцессор, динамическая память

Объединения

Формат: union <имя-объединения> { <список-полей> } ;

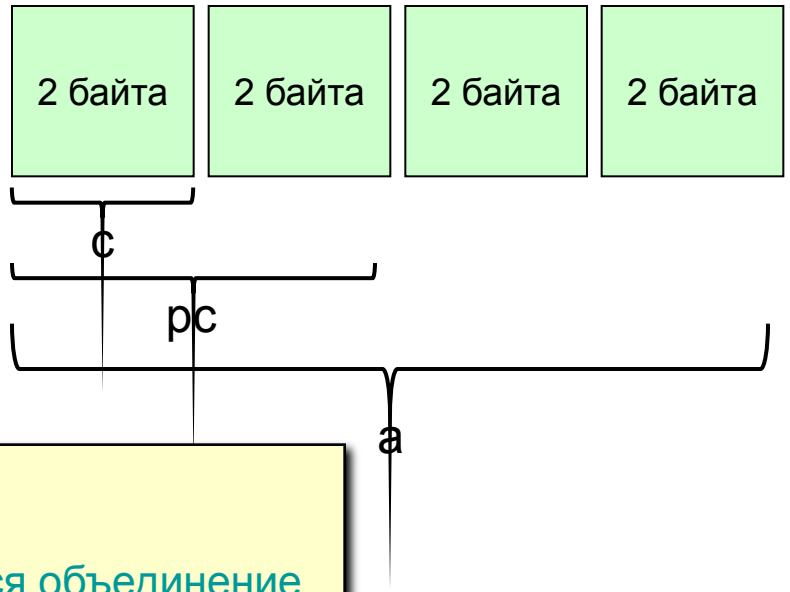
union <имя-объединения> <описатели-переменных> ;

union [<имя-объединения>] { <список-полей> } <описатели-переменных> ;

Объединения – это такие структуры, в которых поля накладываются друг на друга.

Примеры:

```
union bytes  {   объединение 3-х полей
     char c, *pc ;
     double a;
}  value ;
//  примеры доступа к полям:
value.c;   value.a;   *value.pc
```




```
struct list  {
     struct list *next ;
     union bytes value;   полем структуры является объединение
}  z [512];
//  примеры доступа к полям:
z[n].value.a;     *z[m].value.pc;
```

Структуры, препроцессор, динамическая память

Определение типа

Формат: typedef <спецификация-типа> <описатели> ;

Примеры:

```
typedef  char*  STRING ; вводится новый тип STRING
STRING  p,  x[], f() ;        определены: переменная p и массив x объектов типа
                    STRING; функция f возвращает объекты типа STRING
```

```
typedef struct node  {       вводится новый тип  T, который удобен для
    struct node  *left;      описания узлов двоичных деревьев
    struct node  *right;     тип Tp описывает указатели на узлы
    STRING  s;          двоичных деревьев
}  T,  *Tp ;
```

```
typedef  void* DRAW ( char, void* );       вводится новый тип функций DRAW

Объявление
DRAW box ;      ⇔      void*  box ( char, void* ) ;
```

Структуры, препроцессор, динамическая память

Препроцессор

Препроцессор есть текстовый процессор, который просматривает исходный текст до компиляции и решает три задачи:

- обрабатывает символические имена;
- отыскивает и вставляет файлы;
- осуществляет условную компиляцию.

Директивы #define и #undef

формат: #define <имя> <текст>

#define <имя> (<список-параметров>)<текст> // это макрос

#undef <имя>

примеры:

```
#define  LENGTH  80
#define  ABS (x) ( (x) >0 ? (x) : -(x) )
#define  MAX (x, y) ( (x) > (y) ? (x) : (y) )
```

Зачем нужны скобки? Потому что возможны побочные эффекты, например

```
#define  MAX (a, b+c) ( a > b + c ? a : b + c )
#define  ABS ( ++p ) (  ++p >0 ? ++p : -++p )
```

Структуры, препроцессор, динамическая память

Директива #include

формат: #include <имя-пути>
#include "имя-пути"

примеры:

```
#include <iostream>     // ищет файл в стандартных директориях
#include "c:file1.h"    // ищет файл в конкретной директории
#include "file2.h"      // ищет файл в текущей директории
```

Условная компиляция реализуется с помощью следующих директив:

#if #elif #else #endif defined undef

формат: #if <константное-выражение-1> [<текст-1>]
[#elif <константное-выражение-2> [<текст-2>]]
[#elif <константное-выражение-3> [<текст-3>]]
.....
[#else [<текст-n>]]
#endif

Такие конструкции часто используются при инсталляции и настройке программных продуктов.

Структуры, препроцессор, динамическая память

пример:

```
#if ( sizeof (void* ) == 2 )  #define _SMALL_
#elif ( sizeof (void* ) == 4 )  #define _LARGE_     // модель памяти
#else printf ( " ? ") ;
#endif
```

пример:

```
#if ( defined ( _SMALL_) && undef ( _LARGE_ ) )
      #include "file.h"
#endif
```

Популярные макросы для работы с символами находятся в файле <ctype.h>:

isalpha (c) - символ есть буква?
isdigit (c) - символ есть цифра?
isalnum (c) - символ есть буква или цифра?
isspace (c) - пробельный символ (пробелы, табуляция, переводы строки?
islower (c) - символ нижнего регистра?
isupper (c) - символ верхнего регистра?
……

Структуры, препроцессор, динамическая память

Динамическая память

Динамическая память выделяется и освобождается по явным запросам из программы. Время жизни объектов, размещенных в динамической памяти, контролируется программистом явным образом.

В заголовочном файле <alloc.h> описаны прототипы функций

malloc, calloc, realloc, free - все они рабочие, но являются анахронизмами.

Более корректный способ получения динамической памяти следующий:

< указатель > = new <тип-объекта> [(<инициализаторы>)]

Если динамическая память будет успешно выделена, операция new вернет адрес выделенной памяти. В противном случае она вернет значение NULL.

Освободить ранее полученную динамическую память можно операцией delete

delete < указатель >

Попытка освободить динамическую память, которая не была вами получена, приведет к ошибке времени выполнения.

примеры:

```
int *p = new int (3);
char *s = new char [20];
```

Структуры, препроцессор, динамическая память

Пример: ввод строк в динамическую память и выдача их в обратном порядке.

```
#include <process.h>
#include <stdio.h>
typedef struct z {                    // сделайте самостоятельно поясняющий
        char *s;                    // рисунок к этому примеру
        struct z *next }  L, *Lp;      // здесь используется линейный список
Lp first = NULL;                      // первоначально список пуст
char buffer[256];                     // рабочий буфер для ввода строк
void allocate ( char *s) {            // размещение одной строки
Lp p;                        // рабочий указатель
        if ( !( p = new L ) ) exit (1);       // аварийный выход
        if ( !(p -> s = new char[strlen ( s )] ) ) exit (1);
        strcpy ( p - > s, buffer );    // копирование из буфера в динам. память
        p -> next = first;   first = p;  }     // цепляем узел в начало списка
main ()  {
        do  allocate ( gets (buffer) );  while ( strlen ( buffer) > 1) ;       // ввод строк
        while ( first ) { puts ( first -> s);  first = first -> next; }       // вывод строк
        exit (0);
}
```

Структуры, препроцессор, динамическая память

Программный стек

Программный стек – часть памяти компьютера, которая используется для размещения автоматических переменных и обмена данными между функциями.

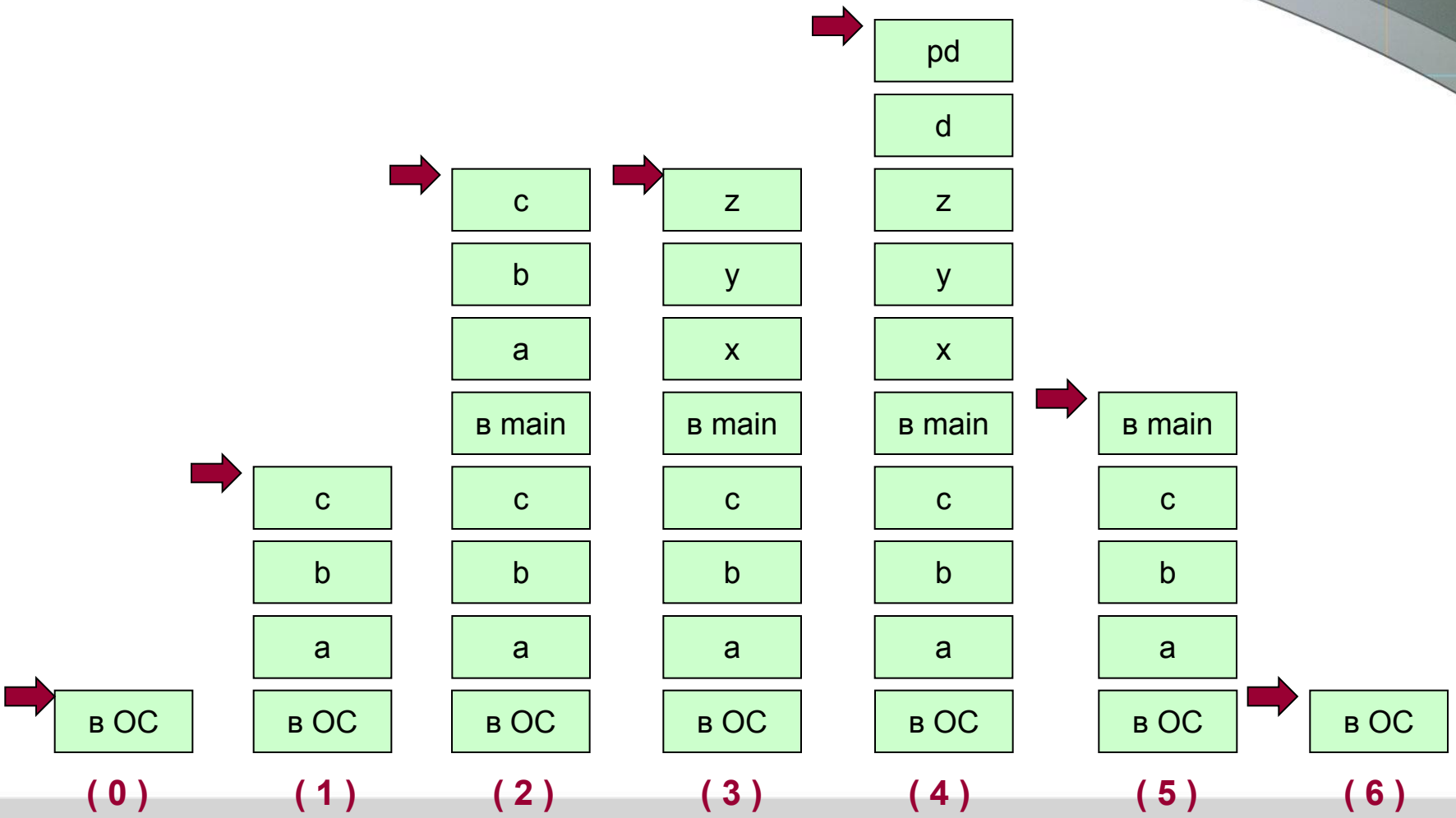
Рассмотрим следующий фрагмент кода:

```
( 0 ) void main ( void )          // определение функции main
      {
( 1 ) long a;                     // три переменные
      char *b, c;                 // с автоматическим классом памяти
      ……….
( 2 ) f ( a, b, c );              // вызов некоторой функции f
      ……….
( 6 ) }                     // конец определения функции main

( 3 ) void f ( long x, char *y, char z )// определение функции f
      {
( 4 ) char d, *pd;                // две переменные
      ……….                        // с автоматическим классом памяти
( 5 ) }                     // конец определения функции f
```

Структуры, препроцессор, динамическая память

Рисунок, поясняющий работу программного стека:



Структуры, препроцессор, динамическая память

Несколько причудливых описаний

Пусть имеется такое описание:

```
struct {
    int x;
    char *y;
} *p;
```

Интерпретируйте следующие выражения в соответствии с правилами языка Си:

```
++p -> x
p++ -> x
*p -> y
*p -> y++
*p -> ++y
*p++ -> y
*++p -> y
p+++y
```

Студенты, допустившие не более 1 ошибки, получают 5 баллов за активность.