Основы программирования на C, часть 4

Наумов Д.А., доц. каф. КТ

Операционные системы и системное программное обеспечение, 2019

Содержание лекции

1. Интерпретация сложных деклараций
2. Структуры и объединения
3. Работа с динамической памятью

Указатели на функции

В языке Си можно определять указатели на функции, которые ничем не отличаются от обычных указателей.
Например: double (*fp)(double);
Указатель на функцию можно:

- присваивать,
- размещать в массиве,
- передавать в функцию в качестве параметра...

```
double x=1.57,y;
    fp=sin;
    y=fp(x); /* y=sin(x) - эквивалентно */
```

Передача функции как параметра

```
#include<stdio.h>
#include<string.h>
#include<math.h>
typedef double (*tf)(double);
main(int argc,char * argv[])
{ char *str[4]={"sin","cos","exp","sqrt"};
  tf m[4]={sin,cos,exp,sqrt};
  double x;
  int i;
  for(i=0;i<4;i++)
     if(!strcmp(argv[1],str[i])) break;
  if(i==4) {
     printf("Имя функции задано неверно\n");
     return(-1);
  }
  x=atof(argv[2]);
  printf("%s(%f)=%f\n",str[i], x, m[i](x));
}
```

В декларациях обычно используется имя (идентификатор) и один из модификаторов *, [ ] и ( ), причем разрешается использовать более одного модификатора в одной декларации.

Для раскрытия этих деклараций применяются следующие правила:

1. Чем ближе модификатор стоит к идентификатору, тем выше его приоритет.
2. Приоритет ( ) и [ ] выше, чем приоритет *.
3. Приоритет повышается заключением в скобки ().

`int matrix[10][10];`	matrix – массив массивов типа int
`char **argv;`	argv – указатель на указатель на char
`int (*ip)[10];`	ip – указатель на массив из 10 элементов типа int
`int *ip[10];`	ip - 10-элементный массив указателей на int
`int *ipp[3][4];`	ipp - 3-элементный массив указателей на 4-элементный массив типа int
`int (*ipp)[3][4];`	ipp – указатель на 3-элементный массив, каждый элемент которого - 4-элементный массив типа int

`int *f();`	f – функция, возвращающая указатель на int
`int (*pf)();`	pf – указатель на функцию, возвращающую int
`char (*(*x())[])();`	x – функция, возвращающая указатель на массив указателей на функцию, возвращающую char
`char (*(*x[3])())[5];`	x – массив из 3 указателей на функцию, возвращающую указатель на массив из 5 элементов типа char

Оператор typedef

Для упрощения прочтения сложных деклараций, а также для именования, типам данных можно задавать новые имена с помощью оператора typedef.

- typedef double (*PFD)() - определяет тип PFD как указатель на функцию, возвращающую double.
- оператор typedef не создает новый тип, а декларирует новое имя (синоним) уже существующего типа.

После ключевого слова typedef следует конструкция, синтаксически аналогичная блоку описания переменных, с той лишь разницей, что вводимое ею новое имя или имена являются не именами переменных, а новыми именами типов.

Пример: $typedefintnumber, *num\_pointer;$

- *number* - синоним типа *int*;
- num_pointer - синоним указателя на *int*.

Декларация структуры

Структура – это тип данных, позволяющий сгруппировать несколько переменных (возможно различного типа) под одним именем.
В общем случае декларация структуры имеет следующий вид:
struct[<тег структуры>]<список деклараций полей>;

```
struct point {
        int x;
        int y;
} a,b;
```

Язык Си поддерживает именную, а не структурную, типизацию, что означает, что два неименованных структурных типа, пусть и содержащие совершенно идентичные списки деклараций полей, будут считаться различными и не будут совместимы по присваиванию.

Декларация структуры

- typedef struct point int x; int y; sp;
- инициализация полей - struct point k = 3,5;
- доступ к полям - k.x, k.y
- p=&k; p->x=2;

```
struct stud {
    char fio[15];/* фамилия студента*/
    struct data { int year;
                  int mon;
                  int day;
    } d;          /* дата рождения */
    int m[3]; /* оценки в сессию */
    };
```

В Си разрешается присваивать и копировать структуры, что позволяет передавать их в функцию в качестве аргумента и передавать из функции в качестве возвращаемого значения (в отличие от массивов, структуры при этом копируются целиком), но структуры нельзя сравнивать.

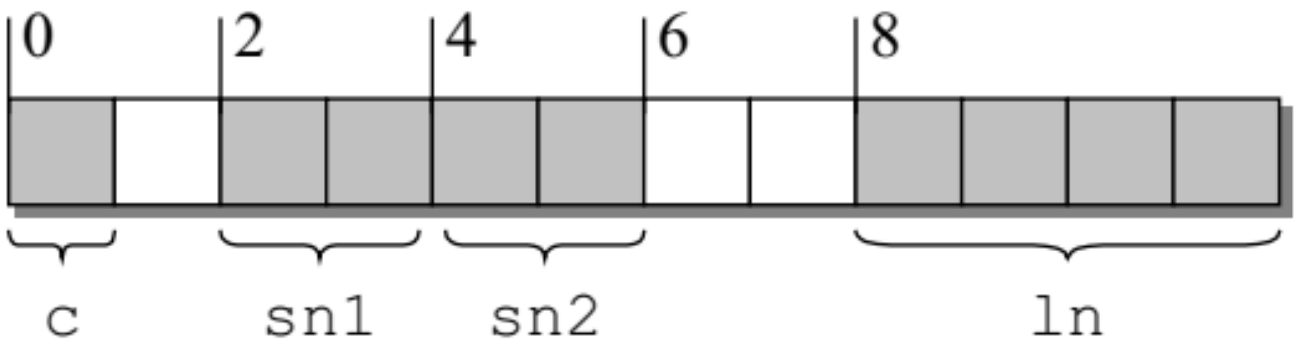
```
struct stud s={“Ivanov”,{1980,6,30},{5,3,4}};
struct data ss;
ss=s.d; /*ss.year=1980; ss.mon=6; ss.day=30;*/
if(ss==s.d) {…} /* ошибка */
```

Написать функцию, параметрами которой являются массив анкет студентов (struct stud) и их количество. Функция печатает фамилии отличников и даты рождения.

```
void f(struct stud g[],int n)
{ int i;
  for(i=0;i<n;i++) {
  if(g[i].m[0]==5 && g[i].m[1]==5 && g[i].m[2]==5)
     printf("%s %d.%d.%d\n",
          g[i].fio,g[i].d.day,g[i].d.mon,g[i].d.year)
  }
}
```

Размещение структуры в памяти

```
struct {
   char   c;
   short  sn1;
   short  sn2;
   long   ln;
}
```

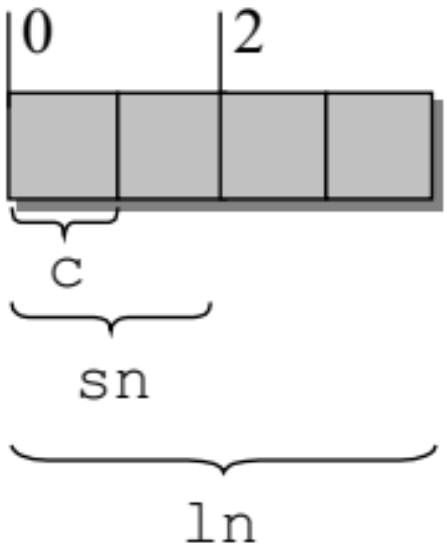


Определить размер структуры - sizeof()

Объединения

Объединение (union) – это тип данных, позволяющий хранить разнородные данные (поля) в одной и той же области памяти.

```
union {
   char   c;
   short  sn;
   long   ln;
}
```




```
union cu{
      char c;
      int I;
      double d;
} a = 'z';
```

Выделение памяти

Для выделения памяти используется функция malloc из стандартной библиотеки.

```
#include <stdlib.h>
void * malloc(size_t size);
```

- функция выделяет блок памяти указанного (в байтах) размера и возвращает указатель на него;
- тип size_t представляет собой один из базовых целочисленных типов, размера которого достаточно для представления любого допустимого в данном контексте значения.
- функция возвращает нетипизированный указатель, перед использованием его, как правило, необходимо привести к требуемому типу.
- в случае неудачи функция возвращает нулевой указатель.

Освобождение памяти

Для освобождения памяти используется функция free уиз стандартной библиотеки.

```
#include <stdlib.h>
void free (void *p);
```

- функции должен быть передан указатель на начало блока памяти, ранее выделенного с помощью malloc().
- После обращения к free() этот указатель становится недействительным.

Изменение размера блока памяти

Для освобождения памяти используется функция realloc уиз стандартной библиотеки.

```
#include <stdlib.h>
void * realloc(void *p, size_t newsize);
```

- функция при необходимости может выделить новый непрерывный блок динамической памяти требуемого размера, при этом она корректно скопирует туда содержимое старого блока и ос вободит старый блок памяти.
- функция возвращает новый указатель на блок измененного размера или, в случае неудачи, NULL (в последнем случае старый блок остается нетронутым).
- После обращения к realloc() старый указатель становится недействительным.

Написать фрагмент программы, размещающий в динамической памяти вводимые из стандартного входного потока вещественные числа. Количество вводимых чисел вводится первым.

```
... int k,i;
    double *p;
    . . .
    scanf("%d",&k);
    p=(double*)malloc(k*sizeof(double));
    for(i=0;i<k;i++) scanf("%lf",p+i);
```

Ввести строку из стандартного входного потока длиной не более ста символов и разместить ее в динамической памяти.

```
... char str[100],*p;
      ...
      if(gets(str)!=NULL) {
            p=(char*)malloc(strlen(str)+1);
            strcpy(p,str);
```

Ввести строку символов из стандартного входного потока и распечатать ее в обратном порядке, построив при этом в динамической памяти стек.

```
... int q;
    struct st { char c;
                struct st *s;
    } *p,*n;

    p=n=NULL;
    while((q=getchar())!='\n') {
        /* построение стека */
        n=(struct st*)malloc(sizeof(struct st));
        n->c=q; n->s=p;
        p=n;
    }
    while(n!=NULL) { /* печать строки */
        printf("%c",n->c);
        n=n->s;
    }
```