**Объявление объектов и типов.**

Все переменные в Си должны быть объявлены. Объявление происходит с помощью ключевого слова – спецификатора типа, идентификатора и модификаторов.

|  |  |
| --- | --- |
| **Модификатор** | **Определяемый тип** |
| \* | указатель |
| [] | массив |
| () | функция |

Можно использовать одновременно более одного модификатора, что позволяет создавать множество сложных описателей типов.

**Замечание:**  Некоторые комбинации недопустимы, например:

1. элементами массива не могут быть функции:
2. функции не могут возвращать функции.

При интерпретации сложных описателей квадратные и круглые скобки (справа от идентификатора) имеют приоритет перед звездочкой (слева от идентификатора).

Квадратные или круглые скобки имеют один и тот же приоритет и связываются слева направо.

Спецификатор типа рассматривается на последнем шаге, когда описатель уже полностью проинтерпретирован. Можно использовать круглые скобки, чтобы изменить порядок интерпретации.

Для интерпретации можно использовать простое правило: «изнутри наружу», состоящее из 4-х шагов.

1. Начать интерпретацию с идентификатора и посмотреть вправо, есть ли квадратные или круглые скобки.
2. Если они есть, то проинтерпретировать эту часть описателя и затем посмотреть налево в поисках звездочки.
3. Если на любой стадии справа встретится закрывающая круглая скобка, то вначале необходимо применить все эти правила внутри круглых скобок, а затем продолжить интерпретацию.
4. Интерпретировать спецификатор типа.

**Пример:**

**int \*(\*comp[10])()**

**6 5 3 1 2 4**

В примере объявляется переменная **comp (1)** как массив (**2**) из 10 указателей (**3**) на функции (**4**), возвращающие указатели (**5**) на целые значения(**6**).

/\* Что напечатает программа ? \*/

#include <stdio.h>

int main()

{

char note[]="You are welcome!";

char \*ptr;

ptr=note;

puts(ptr); // You are welcome!

puts(++ptr; // ou are welcome!

note[7]='\0';

puts(note); // You are

puts(++ptr); // u are

return 0;

}

**// Примеры объявления с использованием модификаторов**

int board [8][8]; // массив массивов типа int

int \*\*ptr; // указатель на указатель на тип int

int \*risk[10]; // 10-элементный массив указателей на тип int

int (\*wis)[10]; // указатель на 10-элементный массив типа int

int (\*oop)[3][4]; // указатель на массив 3x4 типа int

char \*fun(); // функция, возвращающая указатель на тип char

char (\*fun)(); // указатель на функцию, возвращающую тип char

char (\*fun())[3]; // функция, возвращающая указатель на 3-эл. массив har\*/

**Пример: /\* Многоуровневые ссылки \*/**

#include <stdio.h>

char \*m[]={"January","February","March","April","May","June","July",

"August","September","October","November","December"};

char \*\*mp[]={m+11,m+10,m+9,m+8,m+7,m+6,m+5,m+4,m+3,m+2,m+1,m};

char \*\*\*mpp[]={mp,mp+9,mp+6,mp+3};

char \*\*\*\*mppp=mpp;

int main()

{

int i;

printf("%s\n",\*\*\*++mppp);

printf("%s\n",\*\*\*--mppp);

for(i=0;i<12;i++)

printf("%s%c",mppp[0][0][i-11],(i==6)?'\n':' ');

printf("\n");

for(i=0;i<12;i++)

printf("%s%c",mppp[1][2][11-i],(i==5)?'\n':' ');

printf("\n");

printf("%s\n",\*mppp[3][1]);

printf("%s\n",\*\*mppp[1]+3);

printf("%s\n",mppp[2][-1][3]+2);

for(i=0;i<8;i++)

printf("%s\n",mppp[3][-2][-9]+i);

return 0;

}

**Результаты выполнения программы:**

March

December

January February March April May June July

August September October November December

December November October September August July

June May April March February January

August

ch

tober

February

ebruary

bruary

ruary

uary

ary

ry

y

**Пример:**

#include <stdio.h>

char \*c[]={"TEST1 ","NOP ","FILE ","FIRST "};

char \*\*cp[]={c+3,c+2,c+1,c};

char \*\*\*cpp=cp;

int main()

{

printf("%s ",\*\*++cpp);

printf("%s ",\*--\*++cpp+3);

printf("%s",\*cpp[-2]+3);

printf("%s\n",cpp[-1][-1]+1);

return 0;

}

**Результаты выполнения программы:**

FILE T1 STOP

**Пример:/\* нахождение определителя матрицы \*/**

#include<stdio.h>

#include<windows.h>

#define n 4

void main()

{

float a[n][n],koef,det=1;

int i,j,k;

char cStr[]="Введите матрицу n на n";

CharToOem(cStr,cStr);

printf("%s\n",cStr);

for(i=0;i<n;i++)

for(j=0;j<n;j++)

scanf("%f",&a[i][j]);

// Прямой ход

for(i=0;i<n-1;i++){ // идем по строкам

for(j=i+1;j<n;j++){ //по соотв. столбцу

koef=a[j][i]/a[i][i];

for(k=i;k<n;k++) //по столбцам

a[j][k]-=a[i][k]\*koef;}}

// Вывод результатов

for(i=0;i<n;i++){

for(j=0;j<n;j++)

printf("a[%d][%d]=%5.2f\t",i,j,a[i][j]);

printf("\n");}

for(i=0;i<n;i++)

det\*=a[i][i];

printf("det=%5.2f\n",det);

}

**Пример: /\* решение СЛАУ методом Гаусса \*/**

#include<stdio.h>

#include<windows.h>

#define n 4

void main()

{

float a[n][n+1],x[n],koef,det=1,y=0;

int i,j,k;

char cStr[]="Введите матрицу n на n+1";

CharToOem(cStr,cStr);

printf("%s\n",cStr);

for(i=0;i<n;i++)

for(j=0;j<n+1;j++)

scanf("%f",&a[i][j]);

// Прямой ход метода Гаусса

for(i=0;i<n;i++){ // идем по строкам

for(j=i+1;j<n;j++){ // по соотв. столбцу

koef=a[j][i]/a[i][i];

for(k=i;k<n+1;k++) // по столбцам

a[j][k]-=a[i][k]\*koef;}}

// Вывод результатов

for(i=0;i<n;i++){

for(j=0;j<n+1;j++)

printf("a[%d][%d]=%5.2f\t",i,j,a[i][j]);

printf("\n");}

for(i=0;i<n;i++)

det\*=a[i][i];

printf("det=%f\n",det);

// Обратный ход.

x[n-1]=a[n-1][n]/a[n-1][n-1];

for(i=n-2;i>=0;i--){

y=0;

for(j=n-1;j>i;j--)

y+=a[i][j]\*x[j];

x[i]=(a[i][n]-y)/a[i][i];}

for(i=0;i<n;i++)

printf("x[%d]=%5.2f\n",i+1,x[i]);

}

**Структуры**

Структура – тип данных, который объединяет несколько переменных, в общем случае разных типов. Переменные, которые объединены структурой, называются элементами, полями, членами структуры.

**Пример: /\* объявления структуры \*/**

struct student{char name[30];

int course;

char group;

int scholarship;};

Объявление структуры является оператором, то есть в конце должна быть “;”.

Таким образом определяется шаблон структуры.

**struct** – тип структуры, **student** – имя структуры. Все вместе это определяет шаблон структуры, память не выделяется.

Определим переменные типа **struct student:**

**struct student stud1,stud2;**

Компилятор автоматически выделяет под них место в памяти компьютера. Под каждую переменную выделяется непрерывный участок памяти.

Задание шаблона и объявление переменных можно производить в одном операторе:

struct student{char name[30];

int course;

char group;

int scholarship;} stud1, stud2;

Доступ к конкретному элементу структуры осуществляется с помощью операции “точка”.

**Пример:**

strcpy(stud1.name,”Степанов А.В.”);

printf(“%s\n”,stud2.group);

**Массивы структур**

1. Задаем шаблон структуры.
2. Объявляем массив.

**Пример:**

**struct student studcourse[200];**

Доступ к полю name 101- го элемента массива: **studcourse[100].name**

studcourse[100].name[7] – 8 - й элемент (буква) поля name 101-го элемента массива studcourse.

Если объявлены две переменные типа структуры с одним и тем же шаблоном, то можно использовать операцию присваивания:

stud1=stud2;

**Замечание:** нельзя использовать операцию присваивания переменных типа структуры, шаблоны которых описаны под разными именами.

Пример:

second=first; // неправильно

second.a=first.a; // правильно

Можно создать указатель на структуру:

struct student \*pStru; // pStru – переменная типа указатель на структуру student

Операция стрелка (->) употребляется вместо операции точка (.), когда хотят получить значение элемента структуры с применением переменной типа указатель.

**Пример: /\* комплексные числа \*/**

#include <stdio.h>

#include <stdlib.h>

int main()

{

struct complex {

float x;

float y;

} z;

struct complex \*p; // объявление указателя

p = &z;

(\*p).x = 1;

p->y = 2;

printf("Re z = %f\tIm z = %f\n", z.x, z.y);

system("pause");

return 0;

}

В качестве элементов структуры можно использовать массивы, структуры и массивы структур.

**Пример: /\* использование в качестве элемента структуры другой структуры \*/**

struct address{

char city[30];

char street[30];

int house;

};

struct fulladdress{

struct address addr; //

int room;

char name[30];

}AddrStud;

…

AddrStud.addr.house=15; // присвоение значения house структуры address переменной AddrStud

**Пример:**

struct date{

int day;

int month;

int year;

};

struct student{

char name[20];

char surname[20];

struct date birthday;

int group;

int age;

};

struct student students[21];

…

students[4].group=119;

students[4].birthday.day=3;

students[4].birthday.month=8;

students[4].birthday.year=1988;

**Инициализация**

struct date boy={3,8,1988};

/\* Что напечатает программа? \*/

#include <stdio.h>

#include <stdlib.h>

struct house {

float ds;

int rooms;

int stories;

char \*address;

};

int main()

{

struct house fr = { 25000, 6, 5, "Baker street" };

struct house \*sign;

sign = &fr;

printf("%d %d\n", fr.rooms, sign->stories);

printf("%s\n", fr.address);

printf("%c %c\n", sign->address[7], fr.address[10]);

system("pause");

return 0;

}

**Объединения (union)**

Существует еще один тип данных для размещения в памяти нескольких переменных разного типа. Объявляется объединение так же, как и структура.

union u {

int iVar;

char cStr;

long int lVar;

};

Это шаблон объединения.

**Пример**:

union u alfa,beta; // можно с шаблоном

В отличие от структуры для переменной типа union места в памяти выделяется ровно столько, сколько надо элементу объединения, имеющему наибольший размер в байтах.

Остальные переменные будут располагаться в том же месте памяти. Синтаксис использования элементов объединения такой же, как и для структуры:

u.cStr=’5’;

Для объединений также разрешена операция ->, если мы обращаемся к объединению с помощью указателя.

#include <stdio.h>

#include <stdlib.h>

int main()

{

union u {

int iVar;

char cStr;

long int lVar;

double fVar;

};

union u alfa; //

union u \*ptr;

ptr = &alfa;

ptr->iVar = 5;

printf("d=%d\n", sizeof(alfa));

printf("%p\tiVar=%d\n", &alfa.iVar, alfa.iVar);

ptr->fVar = 5.6;

printf("%p\tfVar=%f\n", &alfa.fVar, alfa.fVar);

system("pause");

return 0;

}

**Перечисления**

Перечислимый тип (enumeration).

Этот тип – множество поименованных целых констант.

Перечислимый тип определяет все допустимые значения, которые могут иметь переменные этого типа.

Формат:

enum имя\_типа {список\_названий} список переменных;

**Пример:**

enum seasons {win,spr,sum,aut};

enum seasons s;

Каждое из имен win,spr,sum,aut представляет собой целую величину. По умолчанию они соответственно равны 0,1,2,3.

Оператор

printf(“%d\t%d”,win,aut);

выдаст числа 0 и 3.

Во время объявления типа можно одному или нескольким именам присвоить другие значения:

enum value {one=1,two,three,ten=10,thausand=1000,next};

Если распечатать

printf(“%d\t%d\t%d\t%d\t%d\t%d\n”,one,two,three,ten,thausand,next);

то на экране появятся числа 1 2 3 10 1000 1001, то есть каждый следующий символ увеличивается на 1 по сравнению с предыдущим, если нет другого присвоения.

С переменными перечислимого типа можно производить следующие операции:

1. Присваивать переменную типа ***enum*** другой переменной того же типа.
2. Провести сравнение с целью выяснения равенства или неравенства.
3. Арифметические операции с константами типа ***enum*** (i=win-aut);

Нельзя использовать арифметические операции и операции ++ и -- для переменных типа ***enum***.

Основная причина использования перечислимого типа – это улучшение читаемости программ.

#include <stdio.h>

#include <stdlib.h>

int main()

{

enum seasons { win, spr, sum, aut } s, s1;

printf("%d\t%d\t%d\t%d\t", win, spr, sum, aut);

s = win;

s1 = aut;

int i = spr;

// sum=3; //ошибка

printf("\ns=%d\ts1=%d\ti=%d\n", s, s1, i);

system("pause");

return 0;

}

1. **Пример:**  **/\* Вычисление e x** \*/



Написать программу вычисления экспоненты с точностью 1E-6.

1. **Числа Фибоначчи. Золотое сечение.**

Числа Фибоначчи (Леонардо Пизанский [1180 - 1240]) – числовая последовательность [1228], каждый член которой равен сумме двух его предыдущих членов.

Рекуррентная формула: 

Золотое сечение – гармоническое деление отрезка длины точкой  на две части в соотношении: (\*\*) . Решение этого уравнения: (\*) . Решение также приближенно равно  где  ­− числа последовательности Фибоначчи.

Написать программу, в которой запрашивается количество членов последовательности и далее находится значение Золотого сечения. Вывести решение (\*) при Проверить для (\*\*).

Замечание: для использования математических функций необходимо использовать math.h

1. **Комплексные числа**

Используя структуры, написать сложение, вычитание, умножение и деление комплексных чисел.