**Визначення структурних змінних**

***Структура*** об'єднує логічно пов'язані дані різних типів. Структурний тип даних визначається наступним описом:

struct ім’я\_структури {

Опис\_елементів

};

Приклад:

struct dinner {

char \*place;

float cost;

struct dinner \*next;

};

Структурна змінна описується за допомогою змінної структурного типу.

Приклади:

struct dinner week\_days [7]; /\* масив структур \*/

struct dinner best\_one; /\* одна структурна змінна \*/

struct dinner \*p; /\* покажчик на структурну змінну \*/

**Структура**, запис в термінології мови Паскаль і Ада, - це складений об'єкт, в який входять компоненти будь-яких типів, за винятком функцій. На відміну від масиву, який є однорідним об'єктом, структура може бути неоднорідною. Тип структури вказується записом виду

struct {

список описів

} имя;

У структурі повинен бути вказаний принаймні один компонент. Покажчик типу структури використовується для визначення структур. Визначення структур мають такий вигляд:

тип-даних описателі;

де тип-данных вказує тип структури для об'єктів, що визначаються в описі. У своїй простій формі описателі представляють собою звичайні імена змінних, масивів, покажчиків і функцій. Наприклад, за допомогою визначення

struct {

double x,y;

} a,b,c[9];

змінні a і b визначаються як структури, кожна з яких складається з двох компонентів - x і y. *Змінна* c визначається як масив з дев'яти таких структур.

З визначення

struct {

int year;

short int month, day;

} date1,date2;

виходить, що кожна з двох змінних date1, date2 складається з трьох компонентів: year, month, day.

З типом структури може бути асоційоване ім'я, яке задається описом типу в формі

typedef struct {

список описів

} ім’я-типу-структури;

*Специфікатор* typedef (Визначає клас пам'яті) дозволяє нам створити своє власне ім'я типу. Це нагадує директиву #define, але з наступними змінами:

1. На відміну від #define специфікатор typedef дає символічні імена, але обмежується тільки типами даних.
2. Специфікатор typedef виконується компілятором, а не препроцесором.
3. У своїх межах специфікатор typedef більш гнучкий, ніж #define.

Надалі ці імена можуть використовуватися для визначення структур. Нижче наведено приклад опису типу структури з ім'ям employee:

typedef struct {

char name[30];

int id;

dept d;

family f;

} employee;

де слова dept, family вказують типи, а саме типи структур, попередньо визначені користувачем. Тип структури employee може бути використаний для визначення змінних. Наприклад, визначення

employee chairperson, president, e1, e2;

описує змінні chairperson, president, e1, e2 як структури типу employee.

Існує й інший спосіб асоціювання імені з типом структури. Цей спосіб заснований на застосуванні міток структури. Мітки структури аналогічно міткам перераховуємого типу. Мітка структури описується наступним чином:

struct метка{

список описів

}

де мітка є ідентифікатором. У наведеному нижче прикладі слово student описується як мітка структури:

struct student {

char name[25];

int id,age;

char sex;

};

*Мітки структури використовуються для визначення структур записом виду*

struct мітка список-ідентифікаторів;

Використання міток структури необхідно для опису рекурсивних структур, так як одного тільки оператора typedef недостатньо. У наведеному нижче прикладі опису рекурсивної мітки структури

struct node {

int data;

struct node \*next;

};

мітка структури node дійсно є рекурсивною, так як вона використовується в своєму власному описі, тобто в описі покажчика next. Через наявність знака \* змінна next описана як покажчик на об'єкти типу node. Структури не можуть бути прямо рекурсивними. Структура типу S не може містити компонент, який є структурою типу S. Однак структура типу S може містити компонент, який вказує на структуру типуS.

### Доступ до компонентів структури

Такий доступ здійснюється за допомогою спеціального позначення для виділеного компонента, що має такий вигляд:

s.c

де s є ім'ям структури або значенням структури з компонентом c. s може бути виразом, що дає в результаті значення структури. Наприклад, s може бути викликом функції зі структурою як її значення. До компонентів певної вище структури date1 можна звернутися, вказавши їх позначення:

date1.year

date1.month

date1.day

### Поля бітів в структурах

***Поле бітів -*** це елемент структури, певний як деяке число біт, зазвичай менше, ніж число біт в цілому числі. Поля бітів призначені для економного розміщення в пам'яті даних невеликого діапазону.

Приклад:

struct bfeg {

unsigned int bf\_flg1 : 10;

unsigned int bf\_flg2 : 6;

};

Дана структура описує 10-бітове поле, яке для обчислень перетворюється в значення типу unsigned int, і 6-бітове поле, яке обробляється як значення типу unsigned int.

### Об’єднання

***Об'єднання*** описує змінну, яка може мати будь-який тип з деякої безлічі типів.

Визначення об'єднаного типу даних аналогічно визначенню структурного типу даних:

union ім’я\_об’єднання {

Опис\_елементів

};

Приклад:

union bigword {

long bg\_long;

char \*bg\_char [4];

};

Дані типу union bigword займають пам'ять, необхідну для розміщення найбільшого зі своїх елементів, і вирівнюються в пам'яті до кордону, пристосованих обмежень по адресації як для типу long, так і для типу char \*[4].

*Опис змінної об'єднаного типу*:

Приклад:

union bigword x;

union bigword \*p;

union bigword a[100];

**Перерахування**

*Дані перерахованого типу відносяться до деякої обмеженої множини даних.*

*Визначення перерахованого типу даних*:

enum ім’я\_перерахованого\_типу {

Список\_значень

};

Кожне значення даного перерахованого типу задається ідентифікатором.

приклад:

enum color {

red, green, yellow

};

*Опис змінної перерахованого типу*:

enum color chair;

enum color suite [40];

Використання змінної перерахованого типу в вираженні.

приклад:

chair = red;

suite[5] != yellow;

### Змінні структури

У мовах, таких як Ада і Паскаль, є тип даних, званих змінний запис, об'єкти якого містять набір одних і тих же компонентів плюс компоненти, які не є загальними для всіх інших об'єктів. У мові Сі також є тип даних, подібний змінному запису, який наз. змінною структурою, яка може бути реалізована з використанням комбінації структури і об'єднання. У загальному випадку змінні структури будуть складатися з трьох частин: набору загальних компонентів, мітки активного компонента і частини з мінливими компонентами. Загальна форма змінної структури має такий вигляд:

struct {

загальні компоненти;

мітка активного компонента;

union {

опис компоненту 1

опис компоненту 2

...

Опис компоненту n

} ідентифікатор;

}

Нижче наведено приклад визначення змінної структури health\_record:

struct {

/\* загальна інформація \*/

char name[25];

int age;

char sex;

/\* мітка активного компонента \*/

marital\_status ms;

/\* змінна частина \*/

union {

/\* самотній \*/

/\* немає компонентів \*/

/\* одружений \*/

struct {

char marriage\_date[8];

char spouse\_name[25];

int no\_children;

}

/\* розлучений \*/

char date\_divorced[8];

} marital\_info;

} health\_record;

де тип marital\_status, тобто тип мітки активного компонента ms, описаний як

typedef enum {SINGLE, MARRIED, DIVORCED}

marital\_status;

Нижче наведено кілька прикладів посилання на компоненти змінної структури:

health\_record.name

health\_record.ms

health\_record.marital\_info.marriage\_date

### Покажчики і структури

Розглянемо мітку структури student, опис якої було дано вище як

struct student {

char name[25];

int id, age;

char sex;

}

*Покажчик*  new\_student визначений як

struct student \*new\_student;

Припустимо, що пам'ять виділена таким чином, щоб new\_student вкажував на об’єкт student. Тоді на компоненти цього об'єкта можна посилатися в такий спосіб:

(\*new\_student).name

(\*new\_student).id

(\*new\_student).age

(\*new\_student).sex

Оскільки покажчики часто використовуються для вказівки на структури, в мові Сі спеціально для посилань на компоненти таких структур введений оператор вибору стрілка вправо ->. Наприклад, посилання на вищенаведені компоненти структури можна записати з використанням оператора стрілки вправо -> як:

new\_student->name

new\_student->id

new\_student->age

new\_student->sex

### Масив структур

Процес опису масиву структур абсолютно аналогічний опису будь-якого іншого типу масиву:

struct book libry[MAXBKS];

Цей оператор оголошує libry масивом, що складається з MAXBKS -елементів. Кожен елемент масиву представляє собою структуру типу book. Таким чином, libry [0] є book -структурою, libry [1] - другий book -структурою і т.д.

**Визначення елементів масиву структур.** При визначенні елементів масиву структур ми застосовуємо ті ж самі правила, які використовуються для окремих структур: супроводжуємо ім'я структури операцією отримання елемента і імені елемента:

libry[0].value value – перший елемент масиву

libry[4].title title – п’ятий елемент масиву

### Перейменування типів

Формат

typedef старий\_тип новий\_тип

Приклади:

typedef long large;

/\* визначення тип large, еквівалентний типу long \*/

typedef char \*string;

/\* тип string, еквівалентний типу char\* \*/

Перейменування типів використовується для введення осмислених або скорочених імен типів, що підвищує зрозумілість програм, і для поліпшення перенесення програм (імена одного типу даних можуть відрізнятися на різних ЕОМ).

Приклад:

/\* Реалізовано алгоритм, який дозволяє визначити

рядки матриць, що складаються з однакових цілих,

розташованих в різних стовпчиках. Використовуються

двовимірні масиви і структури. Спочатку виконується

сортування рядків за зростанням. Відсортовані

рядки порівнюються і виводяться на екран номера

однакових рядків \*/

#include <stdlib.h>

#include <string.h>

#include <stdio.h>

#include <conio.h>

#define n 4 /\*кількість рядків \*/

#define m 4 /\*кількість стовпців\*/

typedef struct mas{int i,i1;} mas;

int m1[n][m]; /\*первинний масив\*/

struct mas{int i,i1;};

mas a[n\*2];

/\*масив типу mas, де будуть лежати однакові

рядки, a[1].i и a[1].i1 – однакові рядки\*/

void main()

{

clrscr();

int i,j;

randomize();

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

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

m1[i][j]=random(2);

/\* випадковим чином в масив заносимо цілі \*/

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

printf("\n %d) ",i);

for(int j=0;j<m;j++)

printf(" %d",m1[i][j]);

}

int min, p;

/\* індекс мінімального елемента після s-го елемента

i-го рядка сортування рядків масиву по зростанню \*/

for(i=0;i<n;i++) { /\* i-сортування i-ого рядка \*/

for(int s=0;s<m-1;s++) {

min=m1[i][s+1];

for(int j=s;j<m;j++)

if(m1[i][j]<=min) {

min=m1[i][j];p=j;

}

/\* запам'ятовуємо мінімальний елемент в ряду після

s-го елемента \*/

if(m1[i][s]>=min) {

m1[i][p]=m1[i][s];m1[i][s]=min;

}

/\* міняємо місцями s-й і p-й елемент, якщо

s-й> p-го (мінімального)\*/

}

}

printf("\n");

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

printf("\n %d) ",i);

for(int j=0;j<m;j++)

printf(" %d",m1[i][j]);

/\* виводимо відсортований масив \*/

}

int s,k=0;

/\* скільки елементів в i-му рядку збігаються з елементами i1 рядки \*/

/\*скільки рядків співпали \*/

int i1;

for(i=0;i<n-1;i++) /\* верхній рядок i \*/

for(i1=i+1;i1<n;i1++) { /\* нижній рядок i1 \*/

s=0;

for(int j=0;j<m;j++)

/\* порівняння йде по j-му стовпцю \*/

if(m1[i][j]==m1[i1][j]) s++;

/\* якщо відповідні елементи в i-й і i1-го рядка збігаються то кількість збіглися збільшується на 1\*/

if(s==m) {a[k].i=i;a[k].i1=i1;k++;}

/\* якщо всі елементи i-й і i1-го рядка збіглися, то вони однакові \*/

}

printf("\n Співпадаючі рядки :");

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

printf("\n %d и %d",a[i].i,a[i].i1);

/\* друкуємо a[i].i-у и a[i].i1-у співпадаючий

рядок \*/

getch();

}