Додаткові способи введення та виведення даних

Лекція №10

Під час роботи з масивами доводить вводити досить багато вхідної інформації. Особливо ця проблема виникає під час багатократного виконання однієї і тієї самої програми.

Саме тому необхідні інші способи введення вхідних даних.

Константне задання елементів масиву

Значення вхідного масиву можна задавати у константному вигляді:

const <ім’я масива>[к-ть ел-в]…[к-ть ел-в]={<зн.1, зн.2,…>};

Особливості використання константного задання елементів масиву:

його значення неможливо змінити!

Приклад 1.

```
#include <stdio.h>
#include <stdlib.h>

 main()
{
  const int a[10]={1,2,3,4,5,6,7,8,9,10};
// const a[10]={1,2,3,4,5,6,7,8,9,10};
  int i;
  for (i=0;i<10;i++)
      printf("a[%d]=%d; ",i,a[i]);
  printf("\n");
    }
  system("PAUSE");
  return 0;
}
```

Результат виконання:

a[0]=1; a[1]=2; a[2]=3; a[3]=4; a[4]=5; a[5]=6; a[6]=7; a[7]=8; a[8]=9; a[9]=10;

Приклад 2.

```
#include <stdio.h>
#include <stdlib.h>

 main()
{
 const a[2][5]={{1,2,3,4,5},
                {11,12,13,14,15}};
//   const int a[2][5]={1,2,3,4,5,
                        11,12,13,14,15};
 int i,j;
 for (i=0;i<2;i++)
  {
    for(j=0;j<5;j++) printf("a[%d,%d]=%d; ",i,j,a[i][j]);
    printf("\n");
  }
 system("PAUSE");
 return 0;
}
```

Результат виконання:

a[0][0]=1; a[0][1]=2; a[0][2]=3; a[0][3]=4; a[0][4]=5;
a[1][0]=11; a[1][1]=12; a[1][2]=13; a[1][3]=14; a[1][4]=15;

Приклад 3.

```
#include <stdio.h>
#include <stdlib.h>

 main()
{
 // const a[5][2]={{1,2},{3,4},{5,11},{12,13},{14,15}};
//   const int a[2][5]={1,2,3,4,5,
                        11,12,13,14,15};
 int i,j;
 for (i=0;i<5;i++)
   {
     for(j=0;j<2;j++) printf("a[%d,%d]=%d; ",i,j,a[i][j]);
     printf("\n");
   }
 system("PAUSE");
 return 0;
}
```

Результат виконання:

```
a[0][0]=1; a[0][1]=2;
a[1][0]=3; a[1][1]=4;
a[2][0]=5; a[2][1]=11;
a[3][0]=12; a[3][1]=13;
a[4][0]=14; a[4][1]=15;
```

Приклад 4.

```
#include <stdio.h>
#include <stdlib.h>

 main()
{
  const a[5][2]={{1,2,3,4,5},
                 {11,12,13,14,15}};
//   const a[5][2]={1,2,3,4,5,
//                  11,12,13,14,15};
  int i,j;
  for (i=0;i<10;i++)
    {
      for(j=0;j<2;j++) printf("a[%d,%d]=%d; ",i,j,a[i][j]);
      printf("\n");
    }
  system("PAUSE");
  return 0;
}
```

Результат виконання:

```
a[0][0]=1; a[0][1]=2;
a[1][0]=11; a[1][1]=12
a[2][0]=0; a[2][1]=0;
a[3][0]=0; a[3][1]=0;
a[4][0]=0; a[4][1]=0;
```

Ініціалізація масивів

Надання значень елементам масиву одночасно їх описом:

<тип> <ім’я масиву> [<к-ть ел.>]…[<к-ть ел.>]={зн.1,зн.2,..};

Особливості використання константного задання елементів масиву:

його значення можна змінювати!

Приклад 1.

```
{
Int i,j, a[2][5]={{1,2,3,4,5},
                  {11,12,13,14,15}};
 for (i=0;i<2;i++)
  {
    for(j=0;j<5;j++) printf("a[%d,%d]=%d; ",i,j,a[i][j]);
    printf("\n");
  }
 a[0][0]=100;
 printf("%d\n",a[0][0]);
 system("PAUSE");
 return 0;
}
```

Результат виконання:

```
a[0][0]=1; a[0][1]=2;
a[1][0]=3; a[1][1]=4;
a[2][0]=5; a[2][1]=11;
a[3][0]=12; a[3][1]=13;
a[4][0]=14; a[4][1]=15;
100
```

Генератор випадкових чисел

Щоб в С згенерувати випадкове число, необхідно скористатися функцією **rand**() бібліотеки **stdlib.h**. Ця функція генерує натуральні випадкові числа.

Приклад 1.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

main()
{
int a[10],i;
for (i=0;i<10;i++)
   {
      a[i]=rand(); printf("%d ",a[i]);
   }
  printf("\n");
  system("PAUSE");
  return 0;
}
```

Результат виконання:

41 18467 6334 26500 19169 15724 11478 29358 26962 24464

Ряд випадкових чисел є незмінним для будь-якого запуску програми (41 18467 6334 26500 19169 15724 11478 29358 26962 24464…).

Проблема полягає у тому, що використовується одна й таж початкова точка входу для отримання псевдовипадкового числа.

Щоб уникнути цього, необхідно скористатися функцією **srand** (m). Саме вона встановлює початкову точку для отримання випадкового числа.

Однак, наприклад, **srand** (10) даватиме так само одне й те ж випадкове число 71.

Для того, щоб початкова точка також була випадковою, можна скористатися функцією **time** () з бібліотеки **time.h.** Тепер можна генерувати випадкові числа в постійно змінним потоці системного часу з почтаковою точкою 1 січня 1970 року. У якості параметра функції **time** () необхідно використати **NULL**:

```
srand (time (NULL));
```

Під час генерування псевдовипадкових чисел отримуються цілі числа в діапазоні від от 0 до 32767.

Приклад 2.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

 main()
{
  int a[10],i;
  srand(time(NULL));
  for (i=0;i<10;i++)
    {
      a[i]=rand(); printf("%d ",a[i]);
    }
  printf("\n");
  system("PAUSE");
  return 0;
}
```

Результат виконання:

1) 26169 21822 14724 23328 9136 14274 4216 27911 29433 24372
2) 26472 5616 4924 225 18120 19012 31530 17818 24915 9657

Однак, не завжди необхідні числа з такого великого діапазона. Щоб отримати ціле число з діапазона [0, b), необхідно застосувати ділення націло:

```
int k = rand ()%b;
```

Щоб отримати числа з діапазона [a,b), неохідно скористатися формулою:

```
k= rand () %b + a;
```

Щоб отримати випадкові дійсні числа з діапазона [a,b], необхідо використати формулу в такому вигляді:

```
float k= (float )rand () * (b-a)/RAND_MAX +a;
```

де **RAND_MAX**– це межа діапазона в 32767.

Приклад 3.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

 main()
{
  int i,b=-5,c=10; float a[10];
  srand(time(NULL));
  for (i=0;i<10;i++)
    {
      a[i]=(float)rand()*(c-b)/RAND_MAX +b;
      printf("%.2f ",a[i]);
    }
  printf("\n");
  system("PAUSE");
  return 0;
}
```

Результат виконання:

1) -4.95 0.90 8.00 -0.83 6.59 9.02 2.68 -3.60 1.11 7.01
2) a[i]=rand()*(c-b)/RAND_MAX +b; :
   -5.00 9.00 3.00 9.00 0.00 5.00 3.00 8.00 9.00 -5.00

Основи роботи з файлами

Найоптимальнішим способом збереження вхідних і вихідних даних є файли. В термінології ОС – це об’єм однотипної інформації, розміщеної в певному місці дискового простору.

Файли мають свої параметри:

- ім’я та розширення;
- шлях до місця розташування.

Розглянемо текстові файли, що містять вхідну інформацію.

За замовчуванням інформація вводиться з клавіатури та виводиться на екран монітора.

Для того, щоб змінити напрям потоку вхідної та вихідної інформації – читання з вхідного файлу та виведення у вихідний файл, необхідно “повідомити” про це програмі.

Розглянемо ще один тип змінних **file**.

Ім’я змінної цього типу вказує на те, що при його вказанні буде здійснюватись звернення до вмісту цього файлу.

Загальний вигляд опису:

file *<логічне ім’я файлу>;

Наприклад:

file *f_in,f_out;

Для того, щоб “зв’язати” логічне ім’я зі справжнім файлом на диску, відкривши доступ до нього, використовується функція

fopen(“<ім’я файла на диску>”, ”<спосіб використання файла>”);

Наприклад:

f_in=**fopen**(“in.dat”,”r”);

Найуживаніші способи використання файлів:

r – відкрити існуючий файл для читання;

w - строворити новий файл для запису

(якщо файл с вказаним ім’ям існує, то ві буде переписаний);

r+ - відкрити існуючий файл для читання і запису;

w+ - строворити новий для читання і запису;

rt - відкрити існуючий текстовий файл для читання;

wt – створити новий текстовий файл для запису;

r+t - відкрити існуючий текстовий файл для читання і запису;

На завершення роботи з файлом необхідно його закрити, використавши функцію **fclose**();

Наприклад:

fclose(f_in);

Після успішного завершення операції закриття вказаного файлу функція **fclose**( ) повертає значення нуль. Будь-яке інше значення свідчить про помилкове завершення цієї операції.

Для читання інформації з текстового файлу використовується функція:

fscanf (<логічне ім’я файлу>,<формат введення>,<список змінних>);

Наприклад:

fscanf (f_in,”%d”,&n);

Для запису інформації у текстовий файл використовується функція:

fprintf (<логічне ім’я файлу>,<формат виведення>,<список змінних>);

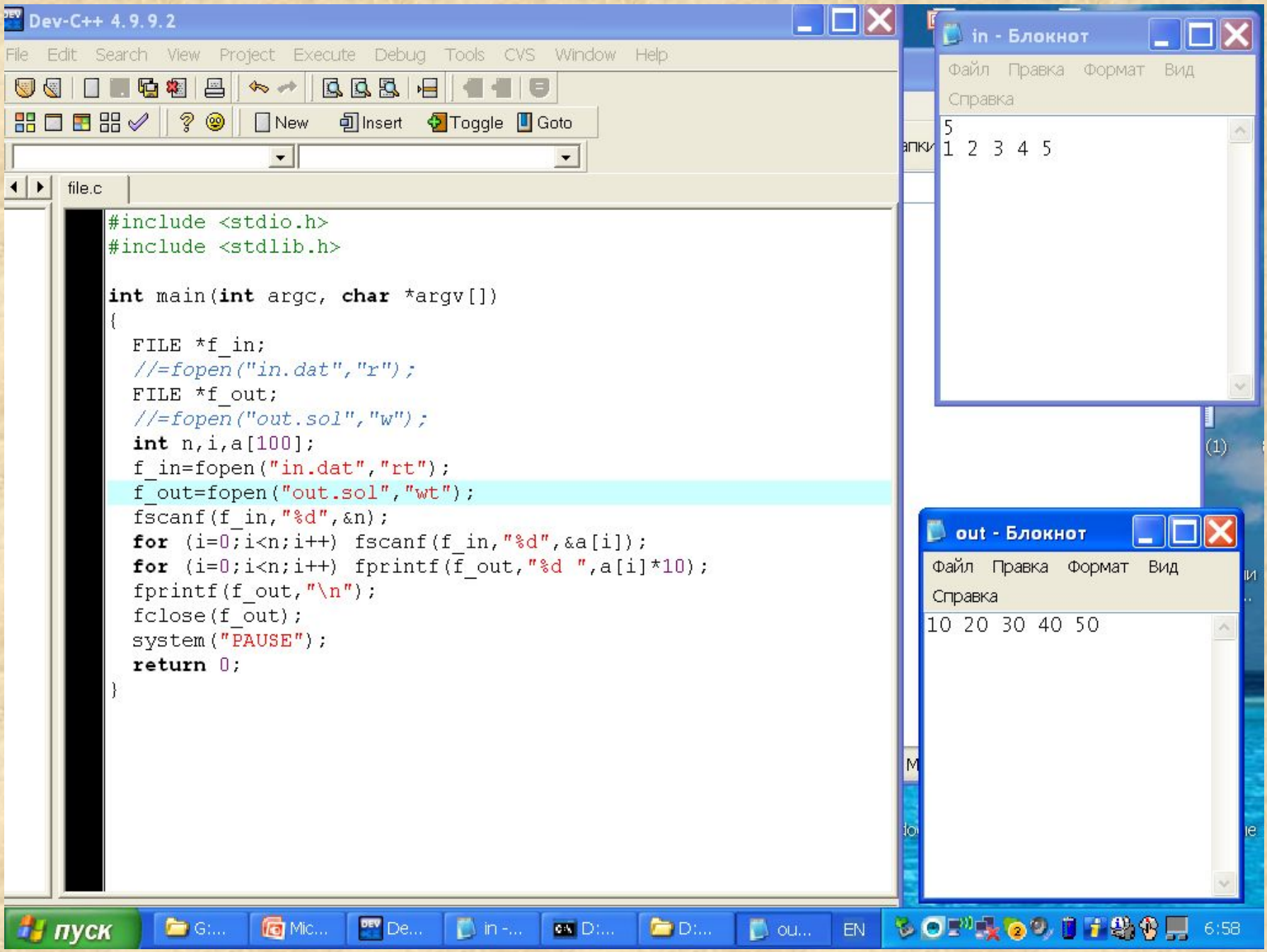
Наприклад:

fprintf (f_out,”%d”,a+b);

Приклад 1.

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
  FILE *f_in;    //=fopen(“f_in.txt","r");
  FILE *f_out;  //=fopen(“f_out.txt","w");
  int n,i,a[100];
  f_in=fopen(“f_in.txt","rt");
  f_out=fopen(“f_out.txt","wt");
  fscanf(f_in,"%d",&n);
  for (i=0;i<n;i++) fscanf(f_in,"%d",&a[i]);
  for (i=0;i<n;i++) fprintf(f_out,"%d ",a[i]*10);
  fprintf(f_out,"\n");
  fclose(f_out);
//  system("PAUSE");
  return 0;
}
```


Dev-C++ 4.9.9.2
File Edit Search View Project Execute Debug Tools CVS Window Help
New Insert Toggle Goto
file.c
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
FILE *f_in;
//=fopen("in.dat","r");
FILE *f_out;
//=fopen("out.sol","w");
int n,i,a[100];
f_in=fopen("in.dat","rt");
f_out=fopen("out.sol","wt");
fscanf(f_in,"%d",&n);
for (i=0;i<n;i++) fscanf(f_in,"%d",&a[i]);
for (i=0;i<n;i++) fprintf(f_out,"%d ",a[i]*10);
fprintf(f_out,"\n");
fclose(f_out);
system("PAUSE");
return 0;
}
in - Блокнот
Файл Правка Формат Вид
Справка
5
1 2 3 4 5
out - Блокнот
Файл Правка Формат Вид
Справка
10 20 30 40 50
пуск
6:58

Вхідні та вихідні файли повинні знаходотись у тій самій теці, що й файл з кодом програми. Інакше необхідно вказувати повний шлях до цих файлів.

Оскільки вхідні та вихідні файли є текстовими, то їх можна створювати і переглядати у будь-якому середовищі, наприклад у “Блокнот”.

Будь-яке середовище програмування С також дозволяє створювати і переглядати такі файли.

Для роботи з файлами використовується бібліотека **stdio.h**.