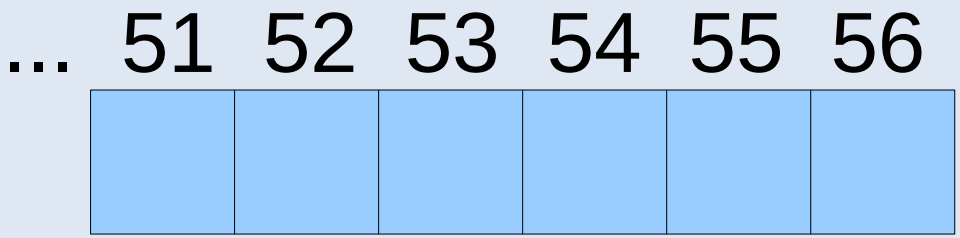
Увод в програмирането

Лекция 7:

Указатели

Памет

- При изпълнение на една програма всички променливи и константи, както и програмният й код, се зареждат в оперативната памет
- Линейна последователност от байтове
- Всеки байт има пореден номер – *адрес*



Представяне на променливи (и константи) в паметта

- Както вече беше споменато, променливите от тип int заемат по 4 байта, double – 8 байта, bool – 1 байт и т.н.
  - Това може да се провери с оператора sizeof
- Пример:

```
double weight;
float prices[8];
// общо 8 + 32 = 40 байта
```

- (Допълнителен материал) друг пример: ако искаме да заредим 4000x3000 (12 MPix) цветно битмап изображение като двумерен масив, обикновено ни трябват 36 MB памет

Намиране на адреса на променлива

- С оператор ”&” (амперсанд) можем да получим адреса на една променлива
  - По-пълно: С унарния (имащ един параметър) оператор ”&” можем да получим адреса на една променлива (или константа, но не и литерал):

```
int a = 5, b = 48950348;
cout << &a << endl << &b << endl;
// грешно: &7, &(a + 1)
```

  - По-конкретно & връща адреса на първия байт на променливата

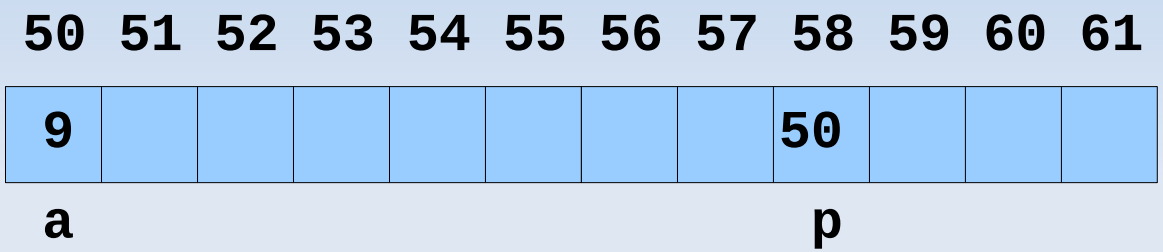
Тип указател

- Типът на резултата, получен с оператора &, е указател
- Въпреки че вътрешно се представя като естествено число, не е unsigned int
- Променлива от тип указател може да съхранява даден адрес в паметта или NULL – специална стойност, показваща, че указателят не сочи към нищо
  - NULL има стойност 0
  - В C++11 има нова константа nullptr, за да не става объркване с числото 0

- Типът указател е параметризиран тип – напр. ”указател към int”, ”указател към double” и т.н.
- Така се разбира как да се интерпретира съдържанието, към което сочи, колко байта да се прочетат и т.н.
- Синтаксис на деклариране на указатели:
  <тип> *<идентификатор> {, *<идентификатор>}

- Пример:

```
int a = 9; //в нашия пример нека a има адрес 50
int* p = &a;
```




```
int* b, c; // b е указател, c е ”обикновен” int
cout << p; // 0x00000032 в нашия пример
           // това е числото 50 в 16-ична бр.с.
// Грешно: cin >> p; както и p = 100;
p = &c; // p вече сочи променливата c
int *q = nullptr; // q не сочи нищо
```

Достъпване на стойността, сочена от указателя

- Унарен оператор ”*”
- С *<указател> можем да правим всичко, каквото можем да правим и с обикновена променлива от съответния тип

```
p = &a;
a = 7; cout << *p; // 7
*p = 8; cout << a; // 8
cin >> *p;
```

- Не трябва да прилагаме оператор * върху нулев указател

```
p = nullptr;
// *p = 123; - грешка по време на изпълнение
if (p != nullptr) *p = 123; // или if (p)
```

- Не трябва да го прилагаме и върху неинициализиран указател
  -
```
int *p;
// *p = 123; - грешка по време на изпълнение
```

Сравняване на указатели

- Можем да използваме ==, !=, <, >, <= и >=

```
double a[] = {1, 2, 3};
double *p = &a[0], *q = &a[2];
if (p < q) cout << "Yes";
```

Адресна аритметика

- Към даден адрес може да се прибавя или изважда цяло число

```
int a[] = {1, 2, 3}; // нека a има адрес 100
int *p = &a[0]; // p сочи адрес 100
p += 1; // p вече сочи адрес 104, а не 101
cout << *p; // 2
```

  - Новият адрес не е 101, тъй като се прибавя `1 * sizeof(int)` – идеята е, че следващата стойност от тип int, която може да се сочи, е чак на адрес 104

- Два адреса не могат да се събират, понеже няма практически смисъл
- Два адреса обаче могат да се извадят:
  `p = &a[0]; int *q = &a[2]; cout << q - p; // 2`

Константи

- Константен указател към тип T
  - T* const <идентификатор>
  - Не може да се пренасочи към друг адрес
- Указател към константа от тип T
  - const T* <идентификатор> или
    T const * <идентификатор>
  - Съдържанието му, т.е. *<идентификатор>, не може да бъде променяно

Допълнителен материал: Указател към void

- Може да сочи променливи от произволен тип

```
int a = 10;
void* p = &a;
```

- Не може директно да се извлече сочената от него стойност, тъй като компилаторът не знае тя от какъв тип е
  - т.е. не знае колко байта да прочете и как да ги интерпретира

```
// грешка: cout << *p;
cout << *((int*) p); // преобразуваме p в
// указател към int и извличаме съдържанието му
double b = 3.14; p = &b; *((double*) p) = 2.5;
```

Връзка между указатели и масиви в C/C++ (1)

- Когато си говорихме за масиви, много неща трябваше да бъдат назубрени, без да е ясно защо са така, например:
  - Индексът на първия елемент е 0
  - При опит да отпечатаме масив със cout получаваме странно число
  - И др.

Връзка между указатели и масиви (2)

- Нека имаме `double array[] = {.5, 3.14, -2e2};`
- array е константен указател към първия елемент на масива

```
cout << *array; // 0.5
```

- `array + i` е адресът на i-тия елемент
  - Затова първият е с индекс 0
- *(array + i) е същото като array[i]
  - На всичкото отгоре е същото и като i[array]
- Понеже е константен указател, не можем да му присвоим за стойност друг масив:

```
// грешка: array = array2;
```

Връзка между указатели и масиви (3)

- При многомерните масиви принципите са същите, просто сложнотиите се натрупват все повече и повече:

```
a[i][j] == (*(a+i))[j] == *(*(a+i)+j)
```

- Затова – дано да е време за междучасие
- След това ще видим реални ползи от указателите

Малко терминология

- Формални (formal) и фактически (actual) параметри на функция:
  - Формални – идентификаторите на параметрите в декларацията, напр. ”a” във void f(int a)
  - Фактически – стойностите, които се подават при извикване на функцията

Предаване по стойност и по адрес

- ```
  void passByValue(int a) {a = 10;}

  void passByAddress(int *a) {*a = 10;}
  ```
- ```
  int b = 5;
  passByValue(b); // a получава стойност 5
      // a и b са съвсем различни променливи
  cout << b << endl; // 5
  passByAddress(&b);
  cout << b << endl; // 10
  ```

Едно приложение: връщане на няколко резултата

- С return можем да върнем само една стойност
- Ще използваме възможността с указател да променим подадени променливи
- ```
  void getTimeComponents(int totalSeconds,
      int *hours, int *minutes, int *seconds)
  {
      *hours = totalSeconds / 3600;
      *minutes = totalSeconds / 60 % 60;
      *seconds = totalSeconds % 60;
  }
  ```
- Демонстрация
  ```

Подаване на масиви като параметри на функции (1)

- **`void printArray(double array[], int length)`**
  - или: double *array - абсолютно същото
- И в двата случая при извикване се подава адресът на масива
- Понеже от този адрес няма как да се разбере колко елемента съдържа масивът, в общия случай подаваме още една променлива, указваща този брой
  - sizeof(array) в тялото на функцията няма да ни помогне – ще върне размера на указателя

Подаване на масиви като параметри на функции (2)

```
void printArray(const double array[], int count) {
    for (int i = 0; i < count; i++)
        cout << array[i] << " ";
    cout << endl;
}
```

- Извикване:

```
double numbers[] = {-.1, 2.34, 4.5, 0.1, 15};
printArray(numbers, 5);
```

- const не е задължителен, но без него на функция, която реално не променя масива, няма да можем да подадем константен масив

Трикове

- Можем да подадем само част от масив: нека имаме **`double array[5] = {9.9, 8.8, 7.7, 6.6, 5.5};`**
- **`printArray(array, 3);`**
  отпечатва първите 3 елемента
- **`printArray(array + 2, 3);`**
  последните 3 елемента
- **`printArray(array + 1, 3);`**
  елементите без първия и последния

Връщане на масиви от функции

- Грешен начин, заблуждаващ:

```
int *evilReturnArray() {
    int array[3] = {4, 5, 6};
    return array;
}
```

- Какво му е грешното, тествах го и работи?

```
int *array = evilReturnArray();
cout << array[0] + array[1] + array[2]; // отпечатва "15"
```

- Нека добавим още код, нямащ нищо общо:

```
double d[1000] = {0};
cout << array[0] + array[1] + array[2] << endl;
// отпечата се някакво странно число, а не сме пипали нищо
```

Връщане на масиви от функции

- След завършване на изпълнението на функцията паметта, в която се намира локалната й променлива array, се маркира като свободна
  - Все едно си купуваме къща, предвидена за събаряне – има я още, но само до време
- При дефиниране на масива d тази памет се използва и старото й съдържание се унищожава
  - (Идват багерите и ни събарят къщата)
- Поука: **никога не връщаме с return указател към локална променлива!!!**
  - Както видяхме от примера, сочената от указателя памет само временно съдържа правилни данни

Връщане на масиви от функции

- Правилен начин: подаваме масив, в който да запишем резултата:

```
int readPositiveNumbers(int result[]) {
    int count = 0;
    while (true) {
        int value;
        cin >> value;
        if (value <= 0)
            return count;
        result[count] = value;
        count++;
    }
}
```

Още един пример – входно-изходни параметри

- Функция, която сортира подадения й масив:

```
void swap(double *a, double *b) { // помощна
        // функция за размяна на стойности
    double c = *a;
    *a = *b;
    *b = c;
}
void sort(double *array, int count) {
    for (int i = 0; i < count - 1; i++) {
        int minIndex = i;
        for (int j = i + 1; j < count; j++)
            if (array[j] < array[minIndex])
                minIndex = j;
        swap(array + i, array + minIndex);
    }
}
```

Подаване/връщане на многомерни масиви

```
double sumMatrix(const double matrix[][3], int rows)
// за да може компилаторът да знае на кой адрес се
// намира (i,j)-ти елемент, трябва да знае колко
// елемента има всеки ред
{
    double sum = 0;
    for (int i = 0; i < rows; i++)
        for (int j = 0; j < 3; j++)
            sum += matrix[i][j];
    return sum;
}

// грешно: void f(int m[][], int rows, int columns)
```

Труден пример (само за Инф и ИС)

- Връщане на два указателя

```
void f(int a[], int **p1, int **p2)
{
    *p1 = a + 2;
    *p2 = a + 3;
}

int *a, *b, arr[] = {1, 2, 3, 4};
f(arr, &a, &b);
cout << *a; // 3
```

- Не е сложно – за да върнем int чрез параметър, пишем int*; за да върнем int* - пишем int**

Псевдоними

- Приличат на указателите, но:
- Не може да им се присвоява нова стойност след инициализация
- Няма стойност, аналогична на NULL
- Няма ”псевдонимена” аритметика
- Много по-лесен синтаксис:

```
int var = 5;
int &ref = var;
ref = 6;
cout << var << endl; // 6
```

Допълнителен материал

- Указател към функция
  - C++11: анонимни функции
- Примери: map, filter и др.

- Второ контролно на Информатика - кога?
- Обобщение
- Въпроси