Увод в програмирането

Лекция 3, част 2:

If, ?:, switch

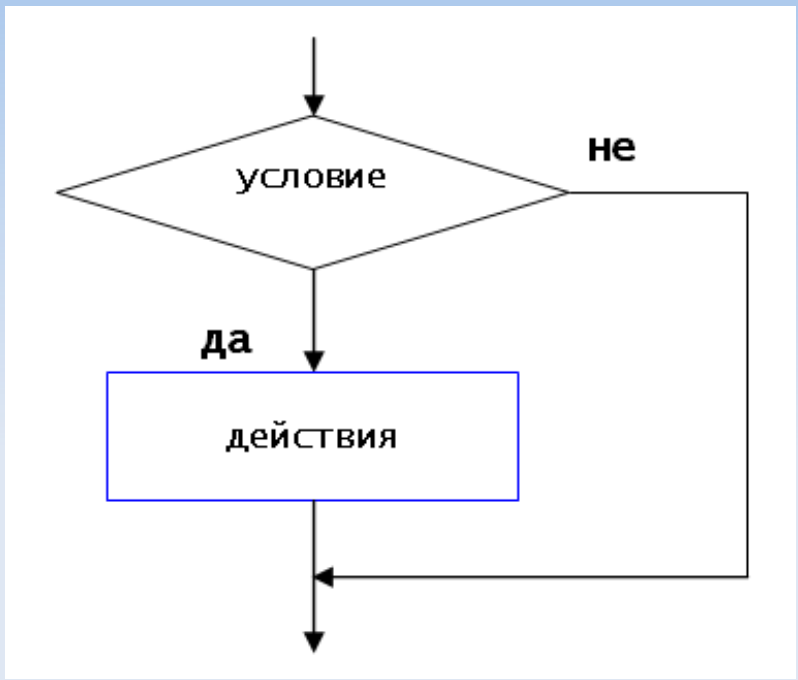
Първа част: if

Условен оператор - преговор

- Непълен синтаксис:

  if (<условие>) <оператор>

- Пример: абсолютна стойност на дадено число:




```
double x;
cin >> x;
double y = x;
if (y < 0)
    y = -y;
cout << y;
```

- (естествено, понеже е често използвано действие, има вградена функция – fabs в библиотеката cmath)

If - подробен синтаксис

- if (<условие>) <оператор$_1$>
  [else <оператор$_2$>]
- Изчислява се условието, зададено с булев израз
- Ако стойността на този израз е true, се изпълнява оператор$_1$
- Ако е зададена клауза else и стойността на условния израз е false, се изпълнява оператор$_2$

If – пример (1)

- Средна цена на продукт при покупка за *p* лева и *n* на брой продукта
  - Извършва се валидация на входните данни
- ```
  double price;
  int count;
  cin >> price >> count;
  if (count <= 0)
      cout << "Error: " <<
          "Number must be positive."
          << endl;
  else
      cout << "Average: "
          << price / count
          << " BGN" << endl;
  ```
  ```

If – пример (2)

- Най-голямото измежду три числа:

```
int a, b, c;
cin >> a >> b >> c;
int max = a;
if (b > max) max = b; // няма else
if (c > max) max = c;
cout << "max(" << a << "," << b
     << "," << c << ") = " << max
     << endl;
```

Вложени условни оператори (1)

- Тъй като цялата конструкция if – else е оператор, условните оператори могат да се влагат един в друг
- Често условните оператори се влагат по следния начин:

  if (<условие$_1$>) <оператор$_1$>
  { else if (<условие$_i$>) <оператор$_i$> }
  [else <оператор>]

- Където конструкцията от втория ред може да се повтаря много пъти

```
if (c1)
    s1
else if (c2)
    s2
else if (c3)
    s3
else
    s4
```

- Ако условие$_1$ е вярно, се изпълнява оператор$_1$ и се спира дотук
- В противен случай се проверява условие$_2$, ако не е вярно – условие$_3$ и т.н. Ако е вярно *i*-то условие, изпълнява се *i*-ти оператор и се спира
- Ако нито едно от условията не е вярно, се изпълнява последната клауза else (ако има такава)

If - else if - else – пример

- Спомени от училище: линейно уравнение $ax + b = 0$
- Не е ли просто
  `cout << -b / a;`

Разглеждане на случаи

- Както видяхме в примера с валидацията, една качествена програма проверява всички възможни ситуации
  - Например грешно въведени от потребителя стойности
  - В случая с линейното уравнение – имаме цели 3 случая
- Ако в програмната логика не сме предвидили обработка на всички ситуации, нашата програма ще се държи неправилно в непредвидените ситуации
  - Грешни изчисления, съобщения за грешка, син екран, отворена врата за хакери ...

If - else if - else – пример (прод.)

- Решаване на линейно уравнение:

```
double a, b;
cin >> a >> b;
if (a != 0)
    cout << "x = " << -b / a << endl;
else if (b != 0) // a == 0, няма
    // нужда да го пишем в условието
    cout << "No solution" << endl;
else // вече a == 0 и b == 0
    cout <<
        "Every number is a solution"
        << endl;
```

Блок

- if и else изискват след тях да стои точно един оператор (statement)
- Почти винаги искаме да се изпълнява по-сложна логика след if (или else)
- За целта ограждаме желания код (поредица от оператори) в големи скоби { } - получава се *блок*
- Играе ролята на единичен оператор, когато ни е нужен такъв
- Можем да пишем вътре всичко, каквото досега сме писали в main
  - Включително и други блокове

if и блок – пример

```
int x; cin >> x;
if (x > 1)
{
    double y = sqrt(x – 1);
    cout << y * y;
}
else if (x == 0)
    cout << "something" << endl;
else
{
    cout << "something else" << endl;
}
```

- При един оператор може блок, може и без

Празен оператор

- ;
- Не извършва никакви действия
- Използва се, когато синтаксисът на някакъв оператор изисква присъствието на поне един оператор, а логиката на програмата не изисква такъв
- ; е като {} (т.е. като празен блок)

Въпрос

```
bool iAmTheSmartestPerson = false;
int iq = 50;
if (iAmTheSmartestPerson);
    iq = 160;
cout << "My IQ is " << iq << endl;
```

- Какво ще се отпечата на екрана?
- Защо?

Вложени условни оператори (2)

- След като в един блок можем да пишем всякакви оператори, в частност можем да напишем и друг условен оператор

```
if (…) {
      …
      if (…) {
            …
      } else …
} else {
      if (…) … else {
            if (…) …
      }
}
```

Въпрос

- Правилни ли са следните твърдения за еквивалентност?
- `if (A) X;` ⇔ `if (A) X; else;`
- `if (!A) X; else Y;` ⇔ `if (A) Y; else X;`
- `if (A && B) X; else Y;` ⇔
  `if (A) if (B) X; else Y; else Y;`
- `if (A || B) X; else Y;` ⇔
  `if (A) X; else if (B) X; else Y;`

Live Demo

- Пресмятане на размера на thumbnail на дадена снимка
- Thumbnail-ът трябва да се вмести в поле с размери 100 на 100 пиксела
- Не във всеки сайт се пресмята правилно

Втора част: условен израз ,,?:”

Условен израз ?:

- <условие> ? <израз_при_истина> : <израз_при_неистина>
- **Израз**, при изчисляването на който се проверява дадено условие и в зависимост от неговата стойност (true / false) целият израз приема стойността на точно един от двата израза – <израз_при_истина> или <израз_при_неистина>
- След като горната конструкция е израз, тя може да участва в други, по-сложни изрази

Пример за ?:

- Пример: да се намери удвоената стойност на по-голямото от две числа

```
int a = 5, b = 6;
int doubleMax = 2 * (a > b ? a : b);
```

- Може да декларираме doubleMax като const
- Или:

```
int a = 5, b = 6;
int doubleMax;
if (a > b) doubleMax = 2 * a;
else doubleMax = 2 * b;
```

Една задача – много решения

- Когато решаваме една задача, можем да напишем правилно решение по много начини
- В случая с if и ?: езикът ни дава две средства с донякъде застъпващи се възможности, но всяко от тях е по-подходящо в различни случаи – изборът е наш
- При компилиране кодът е възможно да се преобразува до едни и същи инструкции на процесора (независимо, че сме използвали различни средства като if и ?:)

Пример за ?: (2)

- Безопасно деление на две числа – ако искаме при знаменател 0 частното също да бъде 0, може да използваме следния израз:

```
b != 0 ? a / b : 0
```

или

```
b ? a / b : 0
```

Пример за ?: (3)

- Да се намери знакът на дробно число *x*, т.е.:

$$\text{sign}x = \begin{cases} +1, & x > 0 \\ 0, & x = 0 \\ -1, & x < 0 \end{cases}$$

- ```
  double x; cin >> x;
  int signx = x > 0 ? 1 : (x < 0 ? -1 : 0);
  ```
- Или: `signx = x < 0 ? -1 : (x > 0)`
- Или:
  ```
  if (x > 0) signx = 1;
  else if (x < 0) signx = -1;
  else signx = 0;
  ```
  ```

Въпрос

Кои от следните изрази са еквивалентни на A && B и кои не са?

- `A ? B : false`
- `A ? true : B`
- `(bool)(A * B)`
- `A && B ? true : false`
- `!(!A || !B)`
- `A && true && B`

Трета част: switch

Мотивация

- Харесва ли ни този код?

```
if (day == 1)
    cout << "Monday";
else if (day == 2)
    cout << "Tuesday";
else if (day == 3)
    cout << "Wednesday";
...
else if (day == 7)
    cout << "Sunday";
else
    cout << "Invalid day";
```

- Ако се запознаем с негови алтернативи, ще видим, че може и по-добре

Синтаксис на switch

switch (<израз>)
{
    case <израз$_1$>: <редица_от_оператори$_1$>
    case <израз$_2$>: <редица_от_оператори$_2$>
    …
    case <израз$_{n-1}$>: <редица_от_оператори$_{n-1}$>
    [default: <редица_от_оператори$_n$>]
}

- Почти винаги последният оператор в редица от оператори е break

Семантика на switch

- Изчислява се изразът в скобите след switch
- Търси се етикет израз$_i$ (измежду указаните след case), който има същата стойност
- Ако се намери – изпълнява се редицата от оператори след съответния етикет
- Ако не се намери, но има default: – изпълняват се операторите след default:
- Внимание: изпълняват се и всички следващи оператори (след други етикети) до стигане на края на switch или до срещане на break или return

Примерът за ден от седмицата

```
switch (day)
{
    case 1: cout << "Monday"; break;
    case 2: cout << "Tuesday"; break;
    …
    case 7: cout << "Sunday"; break;
    default: cout << "Invalid day";
}
```

- Ако забравим break на всички места, при например day=5 ще се отпечата ”FridaySaturdaySundayInvalid day”

Допълнителен материал

- След като switch е опасен (ако пропуснем break), защо е по-хубав от поредица от if-else?
- Защото при if-else условията се проверяват последователно едно след друго, докато при switch компилаторът може да генерира по-ефективен код

Пример: брой дни в месец – две решения

```
int month, days; cin >> month;
```

- ```
  switch (month) {
      case 4: case 6: case 9: case 11:
          days = 30; break;
      case 2: days = 28; break;
      default: days = 31; break;
  }
  ```
- ```
  if (month == 4 || month == 6 ||
     month == 9 || month == 11) days = 30;
  else if (month == 2) days = 28;
  else days = 31;
  ```
- Възможни подобрения:
  - Да се проверява дали month е от 1 до 12
  - Да се въвежда и година и да се прави проверка дали е високосна

Четвърта част: добри практики

Конвенции за оформлението

- {} - по-добре да се слагат винаги, дори и да не са необходими
- След if се поставя интервал
- В скобите около условния израз - не
- Кодът в блок е отместен надясно (с 4 интервала / с табулация / …) – *indentation*
- Отварящата скоба може да бъде на нов ред или на същия, след интервал

```
if (x < 0)
{
    x = -x;
    cout << x;
}
```

Debugging

- Бъг: грешка в програмата
  - За разлика от компилационната грешка, се проявява при изпълнение на програмата и компилаторът не може да го засече
  - Предупрежденията на компилатора (Warnings) понякога предсказват наличието на бъгове
- Когато установим бъг, може да се наложи да проследим внимателно изпълнението на програмата
  - Един възможен начин е временно да поставим cout на повече места в програмата, с цел да проследим стойностите на изразите, които ни интересуват
  - По-добър начин: с инструментите на IDE-то

Debugging (2)

- Демонстрация на debugging в среда за разработка (IDE)
  - (breakpoints, изпълнение ред по ред и т.н.)

Идеи за допълнителни задачи

- Валидация на дата – въвеждат се три стойности – за година, месец и ден, след което се проверява дали образуват валидна дата (месецът да е от 1 до 12 и т.н.)
- Дадени са две 4-цифрени числа – колко бика и колко крави има

Въпроси