Увод в програмирането

Лекция 5:

Функции

Контролни на ИС

- Първо контролно на семинари
- След няколко седмици - първо контролно и на избираемия практикум
  - По план ще включва всичко до...
  - Ще бъде на компютри
    - За подготвените ще е по-лесно, за останалите – не, понеже ще блокират на първото съобщение за грешка

Контролни на Инф, М, ПМ, Стат

- Първо контролно
- По план ще включва всичко до ...

Мотивация (1)

- Съвременният софтуер може да бъде огромен
  - Windows NT от далечната 1993 има 4-5 млн. реда код, Windows XP – 45 млн. и т.н.
- С досегашните знания една по-сложна програма ще има много дълга main функция
- Как подхождаме в реалния свят, когато си имаме работа със сложна система?
  - Напр. автомобил или човешкото тяло

Мотивация (2)

- Опитваме се да разбием сложното на по-прости съставни части
  - Човешкото тяло се изучава система по система, орган по орган
- Отделните части си взаимодействат по някакъв начин, т.е. не са независими
- Дългата програма ще искаме да я разбием на отделни фрагменти, с които да се работи по-лесно

Модулност в програмирането

- В настоящия курс ще разгледаме едно средство за постигане на поставената цел – *функции*
- Функцията (в програмирането) е относително независима част от програмата, изпълняваща специфична задача
- Може да бъде изпълнявана многократно в различни моменти от живота на програмата
- По-късно ще разгледаме приликите с функциите в математиката

Ползи от функциите (1)

- Разбиване на една сложна задача на няколко по-лесни
- Избягване на повторение на код
  - Програмата става по-кратка
  - Ако искаме да променим нещо, ще ни бъде много по-лесно и безопасно, ако пипаме на едно място, а не да го търсим на сто места в програмата
  - Икономия на памет – по-малко процесорни инструкции

Ползи от функциите (2)

- Възможност за повторно използване на код
  - Напр. функциите в библиотеката cmath – би било голяма загуба на време, ако всеки път ги пишем от нулата
- Разпределение на работата
- Абстрахиране от реализацията
- Програмите са по-четливи, улеснява се поддръжката
  - По-лесно ще намерим къде се извършва дадено изчисление и т.н.
- Улеснява се тестването

Getting started (1)

```
void printHelp()
{
  cout << "Бикове и крави 1.0 (C) Пешо"
    << endl
    << "Въвеждате 4-цифрено число "
    << "с различни цифри...";
}
```

- Всяка функция има име
  - Идентификатор, т.е. може да съдържа букви, цифри, долно тире, но не може да започва с цифра

Getting started (2)

- ```
  void printHelp()
  {
    cout << "Бикове и крави 1.0 (C) Пешо"
      << endl
      << "Въвеждате 4-цифрено число "
      << "с различни цифри...";
  }
  ```
- Всяка функция има тяло
  - Блок – можем да пишем всичко в него, каквото и в main
  ```

- Функциите се дефинират извън main
- Засега ще ги дефинираме преди main, за да могат да се използват от main

Извикване на функция (1)

- За да се изпълни примерната функция, трябва:
  - Да напишем имѐто на функцията, последвано от кръгли скоби и накрая ”;”

    ```
    printHelp();
    ```
- Ще се изпълни тялото на функцията
  - В случая – ще се отпечата инструкция за използване на програма за играта ”Бикове и крави”

Извикване на функция (2)

- Демонстрация

```
void printHelp() {
    cout << "...";
}
int main() {
    cout << "Въведете 1 за помощ, 2 за начало";
    int choice;
    cin >> choice;
    if (choice == 1) printHelp();
    else if (choice == 2) {
        // ...
    }
    cout << "Край.";
    return 0;
}
```

Извикване на функция (3)

- (Въпреки че е очевидно) една функция може да се извика от:
  - Функцията main
  - Друга функция
  - Себе си (рекурсия, ще я учим скоро)

Параметри на функциите (1)

- Какво ще стане, ако в дадена програма на едно място искаме да отпечатаме квадратите на числата от 1 до 10, а на друго – на числата от 50 до 60?
- Кодът на двете места е почти еднакъв
- Решение: функциите могат да получават *параметри* (*аргументи*)
- При всяко извикване на функцията можем да имаме различни стойности на параметрите й
- Резултатът от изпълнението на функцията ще зависи от стойностите на параметрите

Параметри на функциите (2)

- Пример:

```
void printSquares(int start, int end)
{
    for (int i = start; i <= end; i++)
        cout << i * i << " ";
    cout << endl;
}
```

- Параметрите могат да бъдат от познатите ни типове – int, double, bool и т.н.
- Винаги се разделят със запетайки
- За всеки параметър се посочва типът, дори и да е еднакъв за всички

Извикване на функция с параметри (1)

- В скобите след името на функцията се изреждат стойности за всеки от параметрите
- Всяка стойности може да бъдат произволен израз от същия (или съвместим) тип като на съответния параметър
- Пример:

```
int n;
cin >> n;
printSquares(1, 2 * n);
```

  - start получава стойност 1, а end – удвоената стойност на n

Извикване на функция с параметри (2)

- Демонстрация с debugging

Връщане на стойност (1)

- Една функция може да върне някаква стойност на този, който я е извикал

Сравнение с математиката (1)

- Познатите от училище функции от вида $f(x)$=... (напр. $f(x)=x^2+2x+1$, sin x и т.н.)
  - приемат един параметър x от тип реално число (което донякъде прилича на double)
  - връщат друго реално число
- Функцията в математиката е съпоставяне на определена величина, наричана аргумент, на друга величина, наричана стойност, като на всеки аргумент се съпоставя точно една стойност
  - Напр. на 1 *f* ще съпостави 4, на 2 – 9 и т.н.
  - Аргументът и стойността могат да бъдат елементи на всякакво множество, не само числа

Сравнение с математиката (2)

- Функциите в програмирането приличат на математическите функции
  - Могат да приемат параметри
  - Могат да връщат стойност
- Резултатът от изпълнението на една функция в програмирането може да зависи не само от параметрите, които са указани при дефинирането й (напр. четене със cin)
- Резултатът може да не е само върнатата стойност (напр. може да има отпечатване)

Връщане на стойност (2)

- Вместо void се указва типът на стойността, която функцията ще върне при извикване
  - Отново можем да използваме познатите типове
  - Void ще използваме само тогава, когато искаме функцията да не връща нищо, напр. ако просто отпечатва нещо на екрана
- Пример:

```
double f(double x) {
    return x * x + 2 * x + 1;
}
```

Връщане на стойност (3)

- Върнатата стойност на една функция може да се използва във всякакви изрази от съвместим тип, например:

```
double y = f(1.23);
double z = f(0.3) * y / 2;
cout << sin(f(f(y)));
```

Оператор return

- Прекратява изпълнението на функцията
- Връща стойността на израза, указан след return, на този, който е извикал функцията
- Ако функцията е void, за прекратяване на изпълнението може да се използва **`return;`** (без израз)
- Ако функцията е от тип, различен от void, задължително трябва да има return
  - Не трябва да има ситуация, в която не се достига до return – напр. ако имаме return в if/switch/цикъл, трябва да сложим return и на други места, така че да покрием всички случаи

Връщане на стойност – пример

- Преобразуване на градуси от Фаренхайт до Целзий:

```
double fahrenheitToCelsius(double degrees)
{
    double celsius = (degrees - 32) * 5 / 9;
    return celsius;
}
```

- Код в main:

```
double fahrenheitDegrees;
cin >> fahrenheitDegrees;
double celsiusDegrees =
    fahrenheitToCelsius(fahrenheitDegrees);
if (celsiusDegrees >= 37) cout << "You're ill";
```

Обобщен синтаксис

- <сигнатура> <тяло>
- <сигнатура> ::= [inline] [<тип> | void] <идентификатор> (<формални_параметри>)
- <параметри> ::= <празно> | void |
  <параметър> {, <параметър> }
- <параметър> ::= <тип> [<идентификатор>]
- <тяло> ::= <блок>
- Допълнителен материал: inline функции

Малко по-сложен пример

- Намиране на първото просто число, по-голямо или равно на дадено *n*:

```
// проверката за просто число е хубаво да е
// в отделна функция
bool isPrime(int number) {
    if (number < 2) return false;
    for (int i = 2; i * i <= number; i++)
        if (number % i == 0) return false;
    return true; // няма else!
}
int findFirstPrime(int n) {
    int current = n;
    while (!isPrime(current))
        current++;
    return current;
}
```

Избор на параметри и върната стойност

- Основен момент при писането на функции е правилното разбиване на програмата на части, както и правилният избор на параметри и върнати стойности
- В противен случай няма да получим ползите, които разгледахме в началото
- Всяка функция е хубаво да решава своята си задача и да не се смесва с несвързани други задачи

Пример

- Функция, която проверява дали цифрите на двуцифрено число са еднакви
- Лошо решение:

```
void areDigitsEqual() {
    int number;
    cout << "Enter a two-digit number: ";
    cin >> number;
    bool result = number / 10 == number % 10;
    cout << (result ? "Equal" : "Not equal");
}
```

- Защо е лошо, нали извежда правилен резултат?

- Нека си спомним какво си говорихме в началото:
- Чрез използването на функции един и същ код може да се използва на много места
- Как тогава ще решим следната задача: да се намери първото двуцифрено число с еднакви цифри
- Хем имаме вече функция за проверка на дадено число, хем тя е неизползваема
  - Функцията чете числото със cin и така не можем да подадем стойност чрез програмен код

- Също: в момента пишем конзолни приложения, но освен тях има и такива с по-сложен графичен потребителски интерфейс (GUI)
- Там вместо cout/cin ще трябва друг код
- Най-добре е функцията да се занимава единствено с проверката на числото и да няма cout/cin
- Вместо това да се използват параметри и върната стойност

- Добро решение:

```
bool areDigitsEqual(int number) {
    return number / 10 == number % 10;
}
```

- Всички стойности, които са необходими за работата на функцията, се подават като параметри, а не се четат със cin
- Резултатът се връща с return, а не се отпечатва
- Така функцията е много по-универсална

Как да се упражняваме

- За упражнение може да вземем произволни задачи от решаваните до момента и да ги пренапишем с функции

Примери

- Функция за намиране на по-голямото от две цели числа
- Чрез горната функция да се намери най-голямото от три цели числа
- Да се напише програма, която намира разликата в секунди между два момента от денонощието. Да има и функции за валидиране на въведените от потребителя стойности

Област на променливите

- Както вече е споменато, всяка променлива е видима в блока, в който е декларирана (и във вложените в него други блокове)
  - Следователно ако в две функции има променливи с еднакви имена, те нямат абсолютно нищо общо
- Променливите в една функция се наричат *локални* за нея (това включва и параметрите)
- Променлива може да се декларира и извън функциите – *глобална променлива* (аналогично и константа)
  - Вижда се във всички функции, дефинирани след нея

Декларация на функция

- Дотук разгледахме дефиниции на функции
- Възможно е само да декларираме дадена функция, без да я реализираме веднага
  - Посочваме върнат тип, име и типове на параметрите (имената им не са задължителни)
  - Така можем да реализираме функцията и след main например
  - Можем и да дефинираме функции, които взаимно се извикват
- ```
  int f(int);
  void g(int x) { if (x > 0) cout << f(1); }
  int f(int a) { if (a < 0) g(a); }
  ```
  ```

Function overloading

- Възможно е да дефинираме функции с еднакво име, но с различни параметри (брой и тип)
- Компилаторът се ориентира коя функция да използва именно по броя и типа на параметрите
- Пример: може да дефинираме различни версии на функция abs за всеки тип:

```
double abs(double value) {
    return value < 0 ? -value : value; }
int abs(int value) {
    return value < 0 ? -value : value; }
```

- При извикване на abs(-10) се извиква втората функция, резултатът е int и няма нужда от преобразуване на типа

Добри практики

- Всяка функция трябва да извършва една добре дефинирана задача
- Името трябва да описва предназначението
  - Добри имена: calculatePrice, readName
  - Лоши имена: f, g1, process
- Не е добре функция да е по-дълга от страница/екран
- Глобални променливи не трябва да се използват, когато е възможна употребата на параметри и върнати стойности

- Две основни стратегии при разработване на по-големи програми:
  - Top-Down
  - Bottom-Up
    - Пример от реалния свят: Лего

Допълнителен материал (1)

Мистериозните параметри на функцията
`int main(int argc, char* argv[])`

- При извикване на програмата могат да се подадат параметри:
  - **`git commit -m "initial commit"`**
  - Програмата git получава масив от символни низове - "-m" и "initial commit"
  - При кликване на файл в Windows Explorer се извиква асоциираното приложение и като аргумент се подава пълният път до файла
- return в main връща *код за грешка*
  - Почти винаги 0 означава, че всичко е наред, а друго число е код на конкретна грешка
  - Който е извикал изпълнимия ни файл, може да разбере върнатия код и да реагира адекватно

Допълнителен материал (2)

- Статични променливи
  - Не са локални за функцията
  - Глобалните променливи също могат да се използват за целта, но статичните са видими само в дадена функция
  - static <тип> <идентификатор> = <израз>;

Допълнителен материал (3)

- Unit тестове
  - Написали сме функция abs
  - Можем да напишем код, който да извиква ф-ята с подходящи примерни данни и да сравнява получения резултат с предварително указан
    - Примерите обикновено са в краищата на областите, които се държат еднакво; в случая – 0, 1 и -1
  - Този тестов код можем да го пускаме всеки път, когато искаме да проверим дали програмата ни работи коректно
    - Правим промени в кода, работят, но чупим нещо друго, което преди е работело – трябва да го тестваме пак
    - Вместо да проверяваме ръчно
    - Естествено, тестовете не гарантират 100% коректност

Допълнителен материал (4)

- Документация към функция
  - JavaDoc

Допълнителен материал (5)

- Уеб приложения

- Обобщение
- Въпроси