Функции

- Парче код, което изпълнява някакво действие
- Може да връща даден резултат
- Един и същ код може да се използва многократно
- Освобождава място в main()

Синтаксис

- <сигнатура> <идентификатор> ([<формални_параметри>]){ <тяло> }

- <сигнатура> ::= [<тип_резултат> | void]
- void = празен тип, не връща резултат

- Ако типът на резултата се пропусне, подразбира се int
- Ако функция със сигнатура различна от void не връща стойност се получава грешка при компилация

Синтаксис

- <сигнатура> <идентификатор> ([<формални_параметри>]){ <тяло> }

- Ако има формални параметри, то трябва да се специфицира типът им
- Ако параметър е примитивен тип данна или някакъв обект, то се създава нов обект в scope-а на функцията!

Извикване на фунцкия

- <име>([<фактически_параметри>]);

- Извикването на функция всъщност е операция с много висок приоритет
- Типът на фактическия параметър се съпоставя с типа на съответния формален параметър
- Ако се налага, прави се преобразуване на типовете <формален_параметър> = <фактически_параметър>

Връщане на резултат

- return [<израз>];
- Оператор за връщане на резултат на функция
- Типът на <израз> се съпоставя с типа на резултата на функцията ако се налага, прави се преобразуване на типовете
- Работата на функцията се прекратява незабавно
- При сигнатура void, return не връща нищо, а просто прекъсва функцията(не е задължителен)

Примери

- Функция намираща сбора на 5 числа

```
int Sum (int a, int b, int c, int d, int e)
{
	return (a+b+c+d+e);
}
```

или

```
int Sum (int a, int b, int c, int d, int e)
{
	int temp = a+b+c+d+e;
	return temp;
}
```

Примери

```
bool ValidateData(int a)
{
	if(a>=1000)
	{
		return true;
	}
	if (a%2 != 0)
	{
		return false;
	}
}
```

//Грешка при компилиране(undefined behaviour), защото не всички възможни изходи връщат стойност

Q&A

- Q: Примерите дотук изглеждат тривиални и прекалено лесни, за да се наложи да използваме функция. Какво ще стане ако просто си ги въвеждам всеки път?

- A1: Кодът ти ще е претрупан с повтарящи се фрагменти, а това намалява качеството на кода. Некачественият код е за други ВУЗ-ове.
- A2: Ако решиш да промениш нещо ще трябва да пренаписваш кода навсякъде. Това е загуба на време, а и може да е източник на грешки.
- A3: Виж в следващия слайд.

Пример

- Алгоритъм за проверка дали едно число е просто

```
bool IsPrime(int a)
{
        if(a<2)
        return false;
        if(a==2)
                return true;
        for(unsigned i = 3; i*i<a; i+=2)
        {
                if(a%i == 0)
                        return false;
        }
        return true;
}
```

Защо не ползвам else if и else?

Пояснение относно параметрите

- Не са задължителни
- Създават се нови обекти, като при примитивните типове данни това е много бърза операция, но при някои други данни може да е много бавно
- Новите обекти са равни на оригиналните, но не са свързани с тях
- Ако промените временен обект във функцията, оригиналният не се променя

Пояснение относно параметрите

- Подредбата им е от значение
- Добра практика е да са константни ако нямаме намерение да ги променяме
- В тялото на функцията не може да се създават нови променливи с имена на параметри
- Ако вече сте забравили
- Новите обекти са равни на оригиналните, но не са свързани с тях

Функция разменяща стойностите на 2 променливи

```
void swap(double a, double b)
{
        double c = a;
        a = b;
        b = c;
}
```

- Нищо няма да се случи, защото a и b са нови временни обекти.
- Тези променливи не са свързани с променливите, които сме подали като параметри!!!

Overloading

- Възможно ли е да имам функция, която да прави повече от 1 действие в зависимост от подадените параметри
  - Пример Sum(1,2,3), Sum(1,2,3,4), Sum(1.55,1.2)

- Отговор: да, това се нарича function overloading

Декларация на функция

- <декларация_на_функция> ::= <сигнатура>;
- Декларацията е “обещание” за дефиниция на функция
- Декларацията не е задължителна
- Една функция може да бъде декларирана няколко пъти, но може да бъде дефинирана само веднъж
- Неизпълнените обещания водят до проблеми...
  - ...освен когато никой не разчита на тях

Декларация на функция - пример

- int abs(int); // името на параметъра не е задължително щом не го
  //ползвате без дефиниране, но не е грешка ако го има

```
…….код……
int abs(int a)
{
        return (a>0) ? a : -a;
}
```

- Този похват се нарича forward declaration, ще го разглеждате по ООП

Overloading

- Една функция може да има безброй много overloads
- При извикване на функцията, компилаторът се грижи да намери правилният overload на функцията
- Компилаторът може да направи преобразуване на данните ако се налага
- Ако не намери подходящ се получава грешка при компилиране
- Ако намери повече от 1 подходящ се получава грешка за двусмислие

Примери за overloading

1. void cout(char a){std::cout<<a;}
2. void cout(int a){std::cout<<a;}
3. void cout(char a, int b){std::cout<<a<<‘-’<<b;}
4. void cout(double a, char b){std::cout<<b<<‘-’<<a;}
5. void cout(bool a){std::cout<<a;}
6. void cout(char a, bool b, int c){std::cout<<a<<b<<c;}
7. void cout(const int a){std::cout<<a;}
8. void cout(char a, unsigned b){std::cout<<a<<‘-’<<b;}
9. char cout(char a){return a;}

Примери за overloading

1. void cout(char a){std::cout<<a;} //двусмислие с 9
2. void cout(int a){std::cout<<a;} //двусмислие със 7
3. void cout(char a, int b){std::cout<<a<<‘-’<<b;} //двусмислие с 8
4. void cout(double a, char b){std::cout<<b<<‘-’<<a;}
5. void cout(bool a){std::cout<<a;}
6. void cout(char a, bool b, int c){std::cout<<a<<b<<c;}
7. void cout(const int a){std::cout<<a;} //двусмислие с 2
8. void cout(char a, unsigned b){std::cout<<b<<‘-’<<a;} //двусмислие с 3
9. char cout(char a){return a;} //двусмислие с 1

Как може да се отстранят тези двусмислия

- void cout(char a, int b){std::cout<<a<<‘-’<<b;}
- void cout(char a, unsigned b){std::cout<<b<<‘-’<<a;}

- void cout(char a, int b){std::cout<<a<<‘-’<<b;}
- void cout(unsigned b, char a){std::cout<<b<<‘-’<<a;}

- Важно! За компилатора има значение подредбата на параметрите. Ако спрямо дадената подредба няма отговаряща функция се получава грешка при компилация

Как може да се отстранят тези двусмислия

- Другите 2 двусмислия няма как да се отстранят така
- Може да се промени името на някоя от функциите
- Може една от функциите да има нов параметър, който да не се използва

```
char cout(char a, bool useless){return a;} //лоша практика
```

Параметри по подразбиране

- Възможно е да имате програма, в която 90% от случаите подавате един и същ параметър на дадено място

- C++ позволява да имате стойност по подразбиране за 1 или повече параметри, които не се налага да уточнявате при извикване на функцията

Параметри по подразбиране

- Синтаксис: void Cout(int a, int b = 5){std::cout<<a<<‘ ‘<<b;}

Cout(4); //4 5
Cout(3,6); //3 6

- Параметрите по подразбиране трябва винаги да са в края!!!

- void Cout(int a){}; //ще се получи двусмислие

Параметри по подразбиране

- void Cout(int a, int b = 5, char c = ‘t’){std::cout<<a<<‘ ‘<<b<<‘ ‘<<c;}

Cout(4); //4 5 t

Cout(3,6); //3 6 t

Cout(3, ‘0’);//3 48 t

- ‘0’ има стойност 48 в ASCII => компилаторът го разглежда като int със стойност 48
- Параметрите по подразбиране винаги са в последователността, в която са дефинирани, не могат да се прескачат

Стекова памет

- Извикването на функция е с много висок приоритет
- Но какво става ако се извика функция в тялото на друга функция?
- Коя функция ще е с по-голям приоритет?
- Отговор: Тъй като извикването на функция е с много висок приоритет, ако в тялото на някоя функция извикаме друга, то втората ще е с по-голям приоритет и след като се изпълни ще се върнем в предната.

Стекова памет

- Какво е стек?
  - Съставна структура от данни, за която ви е още рано?
  - Информация за това какво е стек има включена в бонус материалите

- Как да си обясня стекова памет тогава?

Стековата памет на интуитивно ниво

- Майстор Тричко прави ремонт. Задачата му е да смени кранчето за студената вода.

1.Той започва да го сменя, но се обляга на мивката и я изкъртва.

2.Сега задачата му е първо да смени мивката, но докато го прави спуква тръба.

3.Сега задачата му е да оправи тръбата, но за да го направи трябва първо да спре течащата вода.

4.Той спира водата.

3.След това оправя тръбата.

2. После оправя мивката.

1.Накрая сменя и кранчето за студената вода.