Тема 5: Функции - допълнение. Референции.

1. Function overloading

Function overloading - 2 функции с едно и също време

Пример:
```
int max(int a, int b);
int max(int a, int b, int c);
```

За да се осъществи презаписването на функции трябва параметрите да са или с различен брой, или от различен тип.
Ако не е изпълнено едно от двете условия ще се получи двусмислие, защото компилаторът ще намери повече от една подходящи функции.

Примери за двусмислие:

```
void cout (char ch) {
 std::cout << ch << " ";
}

char cout (char ch){
   return ch;
}
```

Типът на връщане не е от значение. => Дори и да е различен има двусмислица, защото броят на параметрите е еднакъв и са от еднакъв тип.

Това, че е константен int в единия случай, а в другия не, няма значение.

```
void cout (int a) {
   std::cout << a << " ";
}

void cout (const int a) {
    std::cout << a << " " ;
}
```

Примери за коректен function overloading:

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

Тук параметрите са еднакви на брой, но са различни като тип.

Тук параметрите са от еднакъв тип, но са различен брой.

```
void cout(char ch1) { std::cout << ch1; }
void cout(char ch1,char ch2,char ch3) {std::cout << ch1;}
```

Ако имаме функция със същото име, същите по брой и тип параметри, но в различен ред => пак е валидно.

2. Параметри по подразбиране

- Когато ни се налага да подаваме един и същ параметър на дадено място в почти всички случаи.
- Можем да имаме такава стойност по подразбиране за един или повече параметри, които не се налага да уточняваме при извикване на функцията.
- Трябва винаги да бъдат в края и да са в последователността, в която са дефинирани.

```
void print(int a, int b = 5, char c = 't') {
    std::cout << a << " " << b << " " << c;
}
int main() {
    print(4); // 4 5 x
    print(3,6); // 3 6 x}
```

3. Декларация и дефиниция на функции

def: Декларация на функция - казва на компилатора за съществуването на функцията.

def: Дефиниция на функция - казва на компилатора какво прави тази функция.

Пример:

```
int findGCD(int a, int b);//declaration

int main(){
   std::cout << findGCD(6,18);
}

int findGCD(int a, int b){

  while(b!=0{
      int temp = a%b;
      a = b;
      b = temp;
  }
 return a;
}//definition
```

Можем да декларираме функция без да я дефинираме, но ако извикаме тази функция ще получим компилационна грешка.

4. Стек и стекова рамка

Стекова рамка - памет, в която пазим извикването на функциите

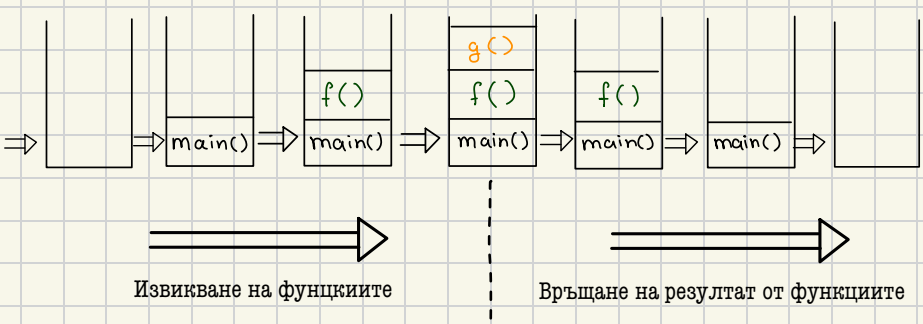
↳ на принципа LIFO(Last in first out)

Пример:

```
void g(){}

void f(){g();}

 int main(){
    f();
}
```

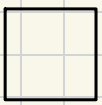


5. Референции

def: Референция - алтернативно име за вече съществуваща променлива

- Променлива може да бъде декларирана като референция чрез символа - & .

```
int a = 5;
int refOfA = &a;
```



a, refOfA

- Досега подавахме променливи във функции по копие.

По този начин функцията не може да промени директно стойността на променливата

- Вече можем да подаваме променливите по референция => функцията ще променя директно стойността на променливата.

Пример: (референция като параметър на функция)

```
void swapByCopy( int a, int b) {
  int  temp = a;
  a = b;
  b = temp;
}
```

VS

```
void swapByReference( int& a, int& b) {
  int  temp = a;
  a = b;
  b = temp;
}
```

```
int main() {
  int a = 2;
  int b = 5;

  swapByCopy(a,b);
  std:: cout << a << "  " << b << std::endl; //  2  5

  swapByReference(a,b);
  std:: cout << a << "  " << b << std::endl; // 5   2
```

- Референцията трябва да се инициализира още при дефиницията си.
- Типът на референцията и променливата трябва да е еднакъв.
- Референцията не може да се пренасочва.
- При инициализация на референцията не се копират данните на променливата, към която реферираме.

Функция връщаща референция:

- функцията не връща стойност на променливата, а цялата променлива
- трябва да сме сигурни, че НЕ връщаме локално създаден обект

Пример за връщане на локално създаден обект:

```
int& func() {
    int a = 10;
    return a;
}
```

Променливата a е създадена в scope-а на функцията и умира при излизане от scope-а. ==> Следователно не знаем какво ще върне функцията. (undefined behaviour)

lvalue vs. rvalue

Lvalue - изрази, които притежават адрес в паметта(променливи, функции, които връщат референция)

Rvalue - изрази, които не са Lvalue

Префиксен оператор ++:
(++a)

```
int& preffPlusPlus( int& a){
   a = a +1;
   return a;
}
```

Постфиксен оператор ++:
(a++)

```
int suffPlusPlus( int& a ) {
   a = a +1;
   return a - 1;
}
```