Встраивыемые функции

При вызове функции создаётся новая область в стеке. В эту область копируются аргументы, переданные функции, выделяется место для локальных переменных...
Однако эти действия иногда являются ненужными, они лишь уменьшают скорость работы программы. В качестве примера рассмотрим функцию, выполняющую простейшую математическую операцию:
```
float mean(float a,float b,float c){
   return (a+b+c)/3;
}
```
При нахождении среднего не является необходимым создавать копии переменных. Ведь они никак не меняются по ходу работы функции.

Таким образом, вместо вызова функции `mean` эффективнее просто считать среднее значение переменных. Но писать математическое выражение самостоятельно на протяжении всей программы неудобно. Это усложняет читаемость кода, увеличивает размер текстового файла программы, и отнимает у нас(программистов) много времени.

Для решения этой проблемы можно использовать концепцию встраиваемых функций:
```
inline float mean(float a,float b,float c){
   return (a+b+c)/3;
}
```
При компилляции все вызовы функции `mean` заменяются на выражения для нахождения среднего значения по трём переменным:
```
k = mean(a,b,c);
```
`===`
```
k = (a+b+c)/3;
```

Тут стоит заметить, что встраиваемые функции изначально небезопасная концепция. Например - создадим внутри встраиваемой функции переменную `b` типа `int`. И вызовем её из функции `main`, где уже ранее определена переменная `b` типа `float`.
Таким образом получаем ошибку, ведь переменная `b` уже была определена, и имела другой тип данных.

Однако в случае использования ключевого слова `inline` ошибки не возникнет:
```
#include <iostream>

inline float mean(float a,float b,float c){
   float alpha = 0;
   return (a+b+c)/3;
}


int main(){
   int alpha = 50;
   mean(5,2,8);

   return 0;
}
```

Компиллятор сам решает будет ли являться данная функция встраиваемой или нет. Таким образом, **использование ключевого слова `inline` для создания встраиваемых функций безопасно**.

Какие аргументы функции могут иметь значения по умолчанию

Думаю, что любой аргумент может иметь своё значение по умолчанию.(возможно это утверждение ошибочно, но информации о противном я не нашел).

Стоит лишь объявлять аргументы в правильном порядке. Аргументы, не имеющие значения по умолчанию, должны объявляться раньше аргументов, имеющих значение по умолчанию:
```
T func(t1 alpha,t2 betta,t3 gamma = zetta...)
```

На основании чего разрешается выбор перегруженной функции

Функция называется **перегруженной**, если она имеет несколько реализаций, и принимает различные типы аргументов.

Говоря проще, **перегруженная функция меняет своё поведение в зависимости от переданного типа данных**.

Выбор конкретной функции с данным именем производится по переданным в неё типам данных.
Тут стоит понимать, что иногда функцию нельзя выбрать однозначно. Рассмотрим пример:
```
#include <iostream>

void function(int alpha){
   std::cout<<"Integer function";
}
void function(float alpha){
   std::cout<<"Float function";
}

int main(){
   function(5);
   return 0;
}
```
При компилляции данной программы возникнет ошибка. `5` может принадлежать как типу `float`, так и `int`.

Чуть изменим код, и передадим в `function` целочисленную переменную `alpha`.
```
int alpha = 5;
function(alpha);
```

Программа скомпиллируется, и при выполнении выведет в консоль
> Integer function

Если явно перевести `alpha` к типу `float`, получим сообщение `Float function`.

Таким образом, если обе функции могут принять аргумент данного типа, выбирается та, которая лучше всего подходит по типу переданных параметров.