**УДК 681.3.06**
**Мельник В.М.**
**Луцький національний технічний університет**

# ЗАСТОСУВАННЯ ФУНКЦІЙ МОВИ ПРОГРАМУВАННЯ BORLAND C++ ДЛЯ ЇХ ПРОГРАМНОЇ РЕАЛІЗАЦІЇ В СЕРЕДОВИЩІ STAY 2.0

*У статті розглядаються практичні можливості функціонального застосування Borland C++ в програмному середовищі STAY 2.0. Також аналізуються можливості параметричного передання даних в ході їх функціональних обробок. Розглянено переваги та недостатки параметричного передання даних за допомогою функцій Borland C++ в середовищі STAY 2.0.*

***Ключові слова:*** *функція, параметричне передання даних, стандартні параметри, посилання, перевантаження (функцій).*

**Вступ.** На сьогоднішній день розроблено досить багато автономних програмних середовищ для розробки програмних продуктів. В залежності від мети їх створення та призначення, вони облегшують розробку програмних додатків та реалізацію підходів втілення різноманітних ідей для відповідних напрямків програмування. Поява великої їх кількості бере початок з сучасної удосконаленої мови C++, що являється багатогранним і невичерпним інструментом створення прикладних програм, включаючи і мови програмування базового призначення. Однією з найбільш гнучких та універсальних версій мови C++ являється версія Borland, яка стикується і підтримує дуже велику кількість оболонок і прикладних програм з можливістю активізації та залучення їх власних розроблених ресурсів. Проте, з іншої сторони, вона являється нелегким джерелом програмування з досить складними вимогами і підходами моделювання та реалізації розробки програмних додатків. То ж, наприклад, створення програмних інтерфейсів може проходити за допомогою таких програмних продуктів, як прикладної програми EXE, Macromedia Authorware 7, AutoPlay Media Studio 6, які, в свою чергу, також були створені в своєму первинному варіанті існування на мові C++.

Сьогодні також відома велика кількість систем управління базами даних (СУБД), що розрізняються за своїми можливостями або такими, що володіють зразково рівними можливостями. Так конкуруючими СУБД являються Paradox, dBase, FoxPro, Oracle, InterBase, Sybase і багато інших, які також беруть свій початок з C++.

Одним із таких прикладних середовищ є недавно створена версія STAY 2.0 для програмної реалізації законів і задач Міністерства праці і соціальної політики України, розробка якої почала впроваджуватися в основному на початку 2001 року, і яка відіграє важливу роль при роботі з базами даних в тому випадку, коли самі дані не можуть бути упорядковані за допомогою звичайної реляційної бази даних та розміщені в її електронних таблицях. Така постановка задачі з'являється тоді, коли, наприклад, для кожного об'єкта інформація повинна заноситись у вигляді не одного, а двох і більше електронних записів в одній і тій же електронній таблиці (базі). Завдання ускладнюється при послідовній роботі з головною і додатковими базами даних, що зв'язані первинними і вторинними ключами, і в яких для кожного об'єкту інформація також формується у вигляді двох і більше електронних записів. Слід зазначити, що така постановка завдання та програмна її реалізація стоїть перед розробниками серверних баз даних в STAY 2.0, що завжди проявляє себе при призначенні видів соціальної допомоги та пенсії з урахуванням їх змін та внесенням поправок до статей закону України про пенсію та соціальне забезпечення різних верств населення. Вищезгадана програма дозволяє створювати серверні бази даних подібного типу та в паралельній роботі з мовою програмування Borland C++ і її компілятором здійснювати їх адміністрування. В комплекс адміністрування таких баз даних входить не тільки контроль за достовірністю внесеної інформації в електронних записах для кожного об'єкта бази, але проведення перерахунків пенсій та соціальної допомоги, пов'язані з підвищенням чи пониженням суми виплат, що базуються на зміні статей закону про соціальний захист населення та пенсію, індексації, вибірка людей зі специфікою соціального забезпечення, поверненнями назад соціальних нарахувань від дат їх призначення, які також регламентуються поправками до відповідних статей Закону.

**Постановка проблеми.** Програма STAY 2.0 через власні функціональні підходи та сукупність функцій Borland C++ дає можливість розробляти і адмініструвати бази даних не тільки в поєднанні з операційною системою DOC чи WINDOWS, але і в поєднанні з версіями Visual C++ при організації процесів їх паралельно організованої роботи. Новостворений інструмент STAY 2.0 дозволяє зі значною легкістю створювати електронні бази даних вищеописаного характеру (їх вікна інтерфейсів, вікна баз та поля, задання їх типів тощо), задавати їх атрибути (первинні та вторинні ключі, права доступу і ін.) та типи зв'язків (по ключах чи конкретних полях), між електронними таблицями, які ініціалізуються службовим словом BASE і зберігаються в одному або декількох файлах. Особливо велику увагу заслуговують функціональні можливості передачі та прийому даних для їх обробки як самого середовища програмування STAY 2.0 так і реалізація багаторесурсних функціональних підходів прийому, передачі і обробки даних самого середовища Borland C++.

**Метою статті** послужило з'ясування особливостей, підходів та програмного втілення функціональних можливостей багаторесурсного середовища Borland C++ в практичному їх застосуванні для програмного середовища STAY 2.0.

**Виклад основного матеріалу**. Одним із широкого кола функцій програмної оболонки Borland C++ (надалі скорочено C++) являються *вбудовані функції*. Функції даного типу мають досить специфічно напрямлене значення при їх використанні. Наприклад, якщо оголосити в програмному середовищі STAY 2.0 функцію з використанням ключового слова inline, як це зроблено в наведеному нижче прикладі, компілятор спробує замінити всі її виклики фактичним кодом функції.

```
inline int FuncA (int X)
{
    // код функції ...
}
```

Заміна викликів функції копією коду функції називається макророзширенням (inline expansion) або просто вбудовуванням (inlineing). Відмітимо, що директива inline, еквівалентна специфічній директиві Microsoft __inline, не гарантує, що функція буде реалізована як вбудована в програмному середовищі STAY 2.0. В деяких випадках компілятор C++ може згенерувати звичайний виклик функції, наприклад, якщо вона являється рекурсивною, або викликається вказівником функції. За допомогою компілятора можна виконати вбудовування функцій (якщо це можливо), використовуючи директиву Microsoft __forceinline [1].

Визначення вбудованої функції (тобто її оголошення, що містить відображення повного коду функції) в програмному середовищі STAY 2.0 має бути представлено в кожному вихідному файлі, в якому викликається дана функція. Для виконання вбудовування компілятор повинен мати прямий доступ до коду функції. Щоб у кожному модулі, що містить виклики вбудованої функції, були доступні ідентичні копії її визначення, необхідно оголосити її у файлі заголовків, що викликається кожним вихідним файлом. При зміні вбудованої функції всі вихідні файли, що містять її виклики, повинні бути перекомпільовані.

Заміна типу функції на inline в програмному середовищі STAY 2.0 як і в Borland C++ не змінює її сенсу, але збільшує швидкість виконання програми і розмір результуючого коду. Тому бажано використовувати специфікатор inline при визначенні маленької функції, що викликається з невеликого числа місць в коді, особливо у разі її багаторазового виклику в циклі.

Механізм вбудовування в програмному середовищі STAY 2.0 нагадує роботу макросів, визначених з використанням директиви препроцесора # define. Наприклад, наступна функція, призначена для повернення абсолютної величини цілого числа:

```
inline int Abs (int N) {return N <0? -N: N;}
```

подібна макросу

```
#define ABS (N) ((N) <0? - (N): (N)).
```

Але є і відмінність. Виклики вбудованої функції обробляються компілятором, а макроси – препроцесором, що виконує просту текстову підстановку. Отже, вбудована функція має дві важливі переваги перед макросом. По-перше, при виклику функції компілятор перевіряє тип переданих їй параметрів, щоб переконатися в тому, що вони цілочисельні або їх значення можна перетворити на цілочисельні. По-друге, якщо функції передається вираз, то вона обчислюється всього один раз. Якщо ж вираз передається в макрос, то він обчислюється двічі, що може призвести до несподіваних результатів [1]. Хоча відмітимо, що у вбудовану функцію також можна

передати аргумент будь-якого числового типу. Зразу ж тип значення буде перетворено в int, і функція також поверне значення типу int, в результаті чого може бути втрачена точність. Однак, як буде показано далі, в мові C++ ці обмеження долаються шляхом визначення кількох варіантів функції, по одному для кожного типу або для безлічі типів параметрів, що передаються функції.

Велику перевагу в створенні програмних додатків та обробці даних в програмному середовищі STAY 2.0 відіграють *стандартні параметри функцій*, які зручно визначати при оголошенні функцій. Наприклад, стандартні значення другого і третього параметру нижченаведеної функції задають таке оголошення:

```
void ShowMessage (char * Text, int Length = –1, int Color = 0);
```

У цій гіпотетичній функції значення параметра Length, рівне –1, змушує функцію обчислювати довжину тексту, а значення параметра Color, рівне 0, задає відображення тексту чорними літерами.

Якщо для деякого параметра задано стандартне значення, необхідно визначити стандартні значення для наступних параметрів (тобто для всіх параметрів, записаних праворуч від нього). Не можна оголосити функцію так:

```
void ShowMessage (char * Text, int Length = –1, int Color);
    // ПОМИЛКА: пропущено стандартне значення для 3-го параметра.
```

При виконанні функції стандартний параметр працює наступним чином: при заданому значенні фактичного параметра компілятор передає це значення у функцію; при опущеному фактичному параметрі компілятор передає стандартне значення параметра. При виконанні функції ShowMessage можна вказати один, два або три параметри.

```
ShowMessage ("Hello");      // те ж, що ShowMessage ("Hello", –l, 0);
ShowMessage ("Hello", 5);   // те ж, що ShowMessage ("Hello", 5, 0);
ShowMessage ("Hello", 5, 8);
```

Якщо параметр із заданим стандартним значенням при виконанні функції опущений, то необхідно опустити всі параметри, що знаходяться праворуч від нього і мають стандартні значення. Функцію ShowMessage не можна викликати наступним чином:

```
ShowMessage ("Hello", , 8); // ПОМИЛКА: синтаксична помилка
```

При визначенні стандартних значень можна використовувати вираз, що містить глобальні змінні або виклики функцій (вирази не можуть містити локальних змінних). Припустимо таке оголошення (передбачається, що код поміщений поза функції):

```
    // на глобальному рівні
int Palette = 1;
int GetColor  (int Pal);
void ShowMessage  (char  *Text,  int Length =  –1, int Color = GetColor
    (Palette));
```

Стандартні значення параметрів можна задати в оголошенні або визначенні функції. Але після визначення стандартного значення параметра його вже не можна перевизначити в наступних оголошеннях або визначеннях функції в попередній області видимості (навіть для присвоєння йому такого ж значення). Наприклад, наступний фрагмент програми генерує помилку.

```
  void ShowMessage  (char  *Text,  int Length =  –1,  int Color  =  0);
void main ()
{
    // код функції ...
}
void ShowMessage (char * Text, int Length = –1, int Color = 0)
    // ПОМИЛКА: повторне призначення стандартних
    // значень параметрам 2 і 3
{
    // код функції ...
}
```

Однак, як показано нижче, в наступні оголошення або визначення функції в попередній області видимості можна додати одне або більше стандартних значень параметрів.

```
void ShowMessage (char *Text, int Length = –1, int Color = 0);
    // далі у вихідному файлі:
void ShowMessage (char *Text = "", int Length, int Color);
```

```
// OK: Додається значення параметра за замовчуванням
```

Велику роль в програмному середовищі STAY 2.0 також відіграють *параметричні посилання в функціях*. Параметри функції і тип повернення функцією значення можна оголосити за допомогою саме них. Наприклад, наступна функція має параметр, який посилається на змінну типу int:

```
void FuncA (int & Parm);
```

При виклику цієї функції їй передається змінна типу int, яка використовується для ініціалізації посилання на параметр Parm. Всередині коду функції параметр Parm є псевдонімом змінної, переданої при виклику, і будь-які зміни Parm також впливають і на саму змінну. Таким чином, контрольний параметр надає спосіб передачі параметра в функцію за посиланням, а не за значенням (а в мовах С і С++ непосилальні параметри передаються за значенням, тобто функція отримує копію оригіналу змінної). Наступний код показує відмінність між посилальним і непосилальним параметрами:

```
void FuncA (int &Parm) // посилальний параметр
{
    ++Parm;
}
void FuncB (int Parm) // не посилальний параметр
{
    ++Parm;
}
void main ()
{
    int N = 0;
    FuncA (N);      // N передається по посиланню
                    // тут N дорівнює 1
    FuncB (N);      // передається через значення
                    // тут N все ще дорівнює 1
}
```

У цьому прикладі функції FuncA і FuncB збільшують значення параметра Parm. Однак тільки функція FuncA, яка отримує параметр як посилання, змінює значення змінної N, що передається з викликаючої функції. Не можна передати константу (наприклад, 5) як параметр, якщо останній не оголошений як посилання на тип const [1]. Наприклад:

```
void FuncA (int & RInt);
void main ()
{
    FuncA (5);      // ПОМИЛКА: посилання не може ініціалізуватися
                    // значенням константи
    // ...
}
```

Якщо у функцію передається змінна великого розміру (наприклад, велика структура), то використання посилального параметра зробить виклик функції більш ефективним. При непосилальному параметрі весь вміст змінної копіюється в параметр. При посилальному параметрі він тільки ініціалізується, посилаючись безпосередньо на оригінал змінної. Окрема копія змінної при цьому не створюється.

Щоб використовувати переваги, надані посилальним параметром, в програмному середовищі STAY 2.0, як і в мовах С та С++, можна використовувати як параметр вказівник (зокрема, можливість зміни значення змінної, що передається викликом програми, і усунення операції копіювання). Однак посилальними параметрами можна маніпулювати за допомогою більш простого синтаксису, подібного до застосування при роботі зі звичайними змінними.

Оператором & в STAY 2.0 можна оголосити функцію, що повертає посилання. Наприклад:

```
int & GetIndex ();
```

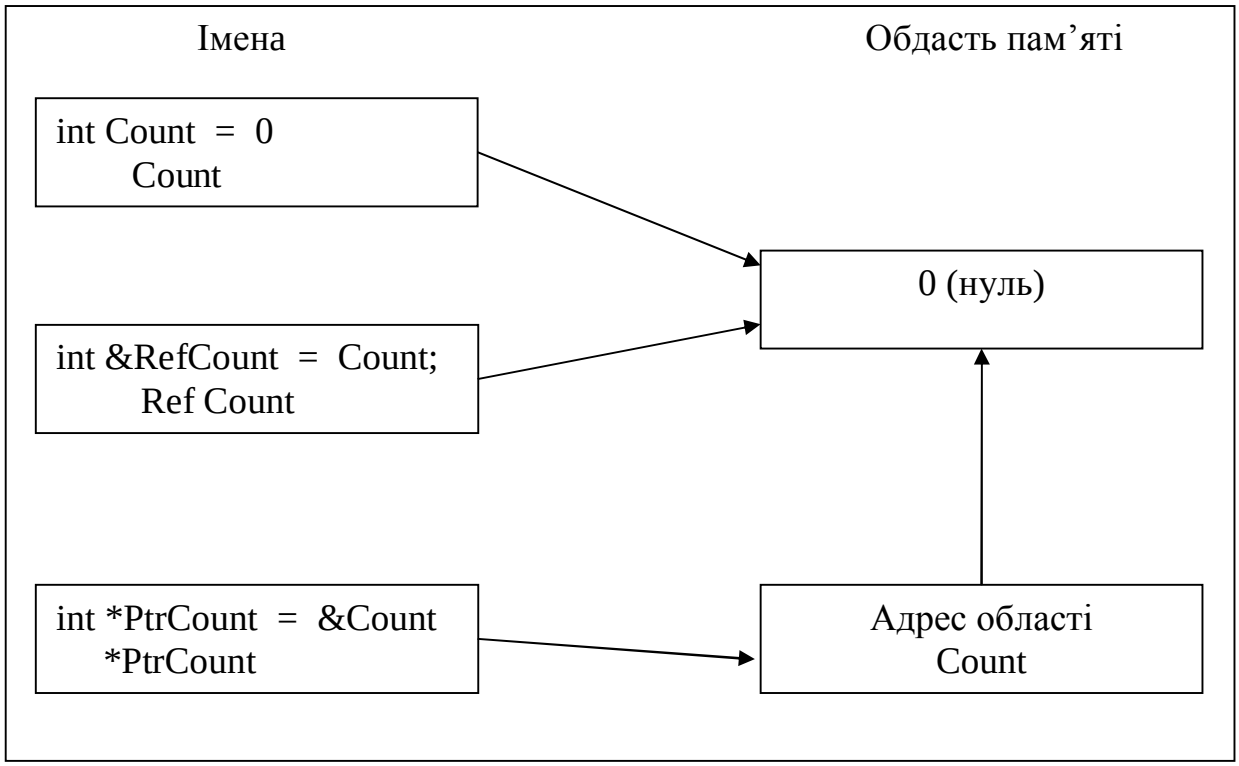
Це оголошення означає, що при виконанні функції GetIndex буде створений тимчасовий псевдонім для деякої цілочисельної змінної (обумовленої оператором return всередині функції). Отже, виклик функції GetIndex може знаходитися в будь-якому місці вираження, що містить дану цілочисельну змінну, наприклад:

```
int N;
N = GetIndex ();  // копіює значення змінної типу int
```

```
    GetIndex O = 5;   // присвоює 5 змінної типу int
++GetIndex ();     // збільшує значення змінної типу int.
```

Наступний код показує, як можна реалізувати функцію GetIndex, і ілюструє ефект виклику функції.

```
int Index = 0;
int & GetIndex ()
{
    return Index;
}
void main ()
{
      int N;
      // тут Index дорівнює 0
      N = GetIndex ();
      // тут Index дорівнює 0 і N дорівнює 0
      GetIndex () = 5;
      // тут Index дорівнює 5
      ++GetIndex ();
      // тут Index дорівнює 6
}
```




Мал. 3.1. Змінна, посилання і покажчик на неї

Функція, що повертає значення яке є посиланням, повинна повертати змінну відповідного типу. Вона також може повертати константу, наприклад 5, тільки у випадку, якщо вона оголошена, як така що повертає посилання на тип const. Змінна повернення використовується для ініціалізації тимчасової посилальної змінної, яка створюється при виклику функції. Іншими словами, вважаючи, що Index і GetIndex визначені так, як показано вище, оператор:

```
++GetIndex (); // виклик функції призводить до створення
                // тимчасової посилальної змінної, що є псевдонімом
                // змінної Index еквівалентний коду
int &Temp = Index;   // оголошення і ініціалізація змінної-
                      // посилання дійсного типу на змінну Temp.
```

У цих прикладах значення змінної Index збільшується на 1.

Оскільки посилання, що створюється при виклику функції, використовується після повернення з функції, остання не може повернути посиланням на змінну, яка знищується після виходу з функції. Наприклад, не можна повернути посилання на автоматичну змінну або параметр, як показано у наступному "небезпечному" коді.

```
int &BadIdea1 ()
{
    int i;
    return i;
}
int &BadIdea2 (int Parm)
{
    return Parm;
}
```

Цей код не викликає помилок компілятора, однак результат використання посилання однією з цих функцій непередбачуваний. Функція, що повертає посилання, може "безпечно" повернути тільки глобальну змінну, змінну типу static або змінну, пам'ять для якої виділяється динамічно.

З іншого боку, функція, що повертає непосилальний тип, може виконати "безпечне" повернення автоматичної змінної або параметра, тому що при виклику функції створюється окрема копія вмісту змінної, а не просто генерується посилання на неї. Саме з цієї причини функція, що повертає значення, що не є посиланням, менш ефективна, ніж функція, що повертає посилання, особливо якщо вона повертає великий об'єкт даних.

Розглянемо *константи як параметри функцій* програмного середовища Borland C++. Параметри функції, як і тип значення повернення нею в програмному середовищі STAY 2.0 можна оголосити, використовуючи ключове слово const.

Оголошення параметра як константи означає, що функція не може змінити значення цього параметра, що стосується лише деталей реалізації функції. Тому при виклику на дану особливість можна не звертати уваги [5].

```
void FuncA (const int N); // FuncA не може змінити N; ну і що?
```

Якщо параметр є вказівником або посиланням, то функція може, зазвичай, змінювати значення переданої змінної.

```
FuncA (int *PInt);
FuncB (int &RInt);
void main ()
{
    int N = 1;
    FuncA (&N); // FuncA може змінювати значення N
    FuncB (N);   // FuncB може змінювати значення N
    // ...
}
```

Якщо ж параметр є вказівником або посиланням на тип const, то функція не може змінити значення переданої змінної. При виклику функції побічний ефект, викликаний зміною значення змінної, відсутній. Наприклад:

```
FuncA (const int *PInt);
FuncB (const int &RInt);
void main ()
{
    int N = 1;
    FuncA (&N); // FuncA НЕ МОЖЕ змінити значення N
    FuncB (N);    // FuncB НЕ МОЖЕ змінити значення N
    // ...
}
```

Крім того, якщо параметр є вказівником або посиланням на тип даних const, то функції можна передати константну змінну. Якщо ж параметр є вказівником або посиланням на неконстантний тип даних, то передача константи в програмному середовищі STAY 2.0 забороняється, так як це призводить до виникнення некоректної ситуації (зміни константи). Наприклад, якщо N оголошена як:

```
const int N = 1;
```

можна законно передати адресу N в FuncA або передати N в FuncB, як у наведеному вище прикладі.

Якщо параметр оголошений як посилання на константу, то можна передати функції константний вираз (неприпустимо для параметра, який є посиланням на неконстанту).

```
void FuncA (const int &RInt);
void main ()
{
    FuncA (5);  // дозволено
    //...
}
```

Якщо функція повертає значення базового типу (наприклад, int або double), додавання ключового слова const до специфікації значення, що повертається у визначенні функції, не має особливого значення. Змінити таке значення, що повертається, в програмному середовищі STAY 2.0 не можна ніяким способом (тобто це не адресується вираз).

Наприклад:

```
const int Func (); // повертається не адресується вираз,
                   // тому його ніяк не можна змінити
```

Однак, якщо функція повертає вказівник або посилання, додавання ключового слова const в програмному середовищі STAY 2.0 означає, що функція виклику не може використовувати значення повернення для зміни значення змінної, на яку вказує або посилається функція. Наприклад:

```
const int *FuncA ()
{
    static int Protected = 1;
    ++Protected;
    return &Protected;
}
const int &FuncB ()
{
    static int Safe = 100;
    --Safe;
    return Safe;
}
void main (int Parm)
{
    int N;
    N = *FuncA (); // дозволено: N приймає копію Protected
    N = FuncB ();   // дозволено: N приймає копію змінної Safe
    *FuncA () = 5; // ПОМИЛКА: спроба зміни значення типу const
    FuncB ();       // ПОМИЛКА: спроба зміни значення типу const
}
```

Зауважте, що самі функції FuncA і FuncB змінюють значення внутрішнього елемента даних. А тому, якщо в їхньому оголошенні в функції виклику є ключове слово const, проводити аналогічні зміни заборонено.

І на кінець розглянемо *перевантажені функції*. У програмуванні на мові C++, можна визначити декілька функцій з одним і тим же ім'ям, якщо функції з ідентичними іменами відрізняються за кількістю і типами параметрів. Наприклад, в програмному середовищі STAY 2.0 оголошення двох різних варіантів функції Abs, один для обчислення абсолютної величини типу int, а інший – для значення типу double. Код практичного застосування наведемо нижче:

```
int Abs (int N)
{
    return N < 0 ? -N : N;
}
double Abs (double N)
{
    return N < 0.0 ? -N : N;
}
```

Якщо в програмному середовищі STAY 2.0 такі функції, які названі ідентично, оголошені всередині однієї області видимості, то їх називають перевантаженими функціями, аналогічно як і в C++ [1,3]. Компілятор автоматично викликає відповідний варіант перевантаженої функції на підставі типу параметра або параметрів у фактичному виконанні функції, наприклад:

```
int Abs (int N); // обидва варіанти функції Abs оголошені в
double Abs (double N); // області видимості файлу
void main ()
{
    int I;
    double D;
    I = Abs (5);          // викликає 'int Abs (int N)'
    D = Abs (-2.5);       // викликає 'double Abs (double N)'
    // ...
}
```

Дві перевантажені функції повинні відрізнятися хоча б по одному з двох наступних правил:

• функції мають різне число параметрів;
• типи одного або більше параметрів різні.

Як бачимо в наведеному прикладі, перевантажені функції можуть повертати результати різних типів. Однак, як показано в наступному помилковому коді, дві перевантажені функції не можуть відрізнятися тільки типом, що повертається.

```
int Abs (int N)
{
    return N <0? -N: N;
}
double Abs (int N) // ПОМИЛКА: перевантажена функція, яка
                   // відрізняється типом значення, що повертається
{
    return (double N) (N <0.0?-N: N);
}
```

Якщо два параметри відрізняються тим, що один з них має тип const або є посиланням, то при визначенні перевантажених функцій в програмному середовищі STAY 2.0 як і в C++ [3,4] дані параметри будуть вважатися однотипними. Наприклад, не можна визначити перевантажені функції таким чином [5]:

```
int Abs (int N);
int Abs (const int N); // ПОМИЛКА: список параметрів занадто схожий.
```

У цьому випадку параметри типу int і const int будуть ініціалізовані з використанням однакового набору типів даних. При передачі будь-якого з цих типів після виклику функції компілятор не зможе визначити, яка саме перевантажена функція викликається. Аналогічно, в програмному середовищі STAY 2.0 не можна визначити перевантажені функції наступним чином:

```
int Abs (int N);
int Abs (int &N); // ПОМИЛКА: список параметрів занадто схожий,
```

так як в обидві функції буде передана змінна типу int.

Якщо передати в перевантажену функцію аргумент, який не збігається з типом аргументу, визначеного для будь-якого варіанту функції, то компілятор спробує перетворити аргумент до одного з певних типів. Буде виконано або стандартне перетворення (наприклад, int в long), або перетворення, визначене користувачем. Якщо перетворити типи неможливо, то компілятор згенерує помилку. В іншому випадку, тобто коли перетворення можливе, буде викликана функція, специфікація аргументів якої найбільш близька до отриманої. Коли функції подібні в рівній мірі, генерується помилка (виклик функції неоднозначний). Відмітимо, що докладно критерії порівняння типів параметрів перевантажених функцій в програмному середовищі Borland C++ розглянуті в розділах довідникової системи [2].

Параметри функції зі стандартними значеннями в програмному середовищі STAY 2.0 також можуть бути причиною неоднозначного виклику перевантажених функцій, наприклад:

```
void Display (char *Buffer);
void Display (char *Buffer, int Lenght = 32);
```

Наступний виклик згенерує повідомлення про помилку, тому що список параметрів відповідає обом перевантаженим функціям.

```
Display ("Hello"); // ПОМИЛКА: неоднозначний виклик!
```

Зверніть увагу: компілятор не генерує помилку при оголошенні функцій, що перевантажуються. Він сигналізує про помилку тільки при неоднозначному виклику.

**Висновок.** Одже, дослідження поведінки функцій та їх параметрів з багаторесурсного програмного середовища Borland C++ відіграє велику роль при створенні функціональних програмних кодів в програмному середовищі STAY 2.0. Знання поведінки функціональних параметричних величин передання як аргументів для послідовної обробки даних не тільки запобігає від помилкових ситуацій та від непередбачених результатів при створенні програмних продуктів, але дозволяє ефективно проектувати програмний код, збільшуючи швидкодію програми, ефективне використання оперативної пам'яті та організацію вигідної передачі даних між функціями.

Застосування різних типів функцій середовища Borland C++ при створенні програмних додатків в програмному середовищі STAY 2.0 дозволяє можливість програмного розв'язання найрізноманітніших завдань для розробників, що на сьогодні являється найціннішим в програмуванні і перетворює його в одне із високодосконалих програмних середовищ для розробки програмування нереляційних баз даних та їх адміністрування.

1. М.Вахтеров, С.Орлов. Четвертый Borland C++ и его окружение / Центр информационных технологий, електронное издательство, 1994.
2. The draft standard C++ library, P.J.Plauger, Prentice Hall, 1995
3. Х.Дейтел і П.Дейтел. Как програмировать на C/C++ – М.-К.-С.П:, 2006.– 908с.
4. Архангельский А.Я. Программирование в C++ Builder 5. М.:ООО "БИНОМ-ПРЕСС", 2003.- 1152 с.
5. Архангельский А.Я. Приемы программирования в Borland C++. М.:ООО "БИНОМ-ПРЕСС", 2003.- 784 с.