

Увод в програмирането

Функции в C++

2017-2018 г.

ФМИ, специалност „Софтуерно инженерство“

Съдържание

- Понятие за функция в C++
- Деклариране, дефиниране и извикване на функции
- Представяне в паметта. Стекови рамки.
- Понятие за указател (само понятие)
- Формални и фактически параметри
- Подаване на параметри по стойност, указател и псевдоним.
- Предефиниране на функции
- Контрол на входните данни

Функция

- Какво означава функция
- В езика C++, функциите са основен конструктивен елемент на програмите
- Дават възможност за многократна употреба
 - Всяка функция се състои от множество от оператори, оформени подходящо за да се използват като обобщено действие или операция.
 - След като една функция бъде дефинирана, тя може да бъде изпълнявана многократно за различни входни данни.

Функции

- Разделяй и владей
 - Позволяват да се създават програми от комбиниране на по-малки „парчета код“ (т.е. Функции)
 - Всяко такова парче е по-ясно и лесно за тестване и модифициране
- Избягва се многократното повтаряне на едни и същи програмни фрагменти. Те се дефинират еднократно като функции, след което могат да бъдат изпълнявани произволен брой пъти.
- Постига се икономия на памет, тъй като кодът на функцията се съхранява само на едно място в паметта, независимо от броя на нейните изпълнения.

Дефиниране на функция в C++

- <сигнатура> { <тяло> }
- <сигнатура> ::= [<тип_результат> | void]
 <идентификатор> (<формални_параметри>)
 - void = празен тип, не връща резултат
- <формални_параметри> ::= <без параметри> | void |
 <параметър> {, <параметър> }
- <параметър> ::= <тип> [<идентификатор>]
- <тяло> ::= { <оператор> }

Извикване на функция

- Със стартиране на програмата се извиква главната функция `main()`
- След това всяка функция, включително и `main()` може да извиква произволен брой други функции
- Извикването на функция е операция с много висок приоритет
- Извикващата функция продължава изпълнението само след като извиканата функция завърши
 - Всяка функция завършва при срещане на `}` , която маркира край на тялото ѝ, или при изпълнение на оператора `return;`

Извикване на функция

- $\langle \text{име} \rangle (\langle \text{фактически_параметри} \rangle)$
- $\langle \text{фактически_параметри} \rangle ::= \langle \text{празно} \rangle \mid \text{void} \mid \langle \text{израз} \rangle \{, \langle \text{израз} \rangle \}$
- $\langle \text{формален_параметър} \rangle = \langle \text{фактически_параметър} \rangle$
 - ако се налага, прави се преобразуване на типовете
 - типът на фактическия параметър се съпоставя с типа на съответния формален параметър
- Фактическите параметри се наричат и атрибути

Примери за функции в C++

- Да се намери абсолютната стойност на дадено цяло число

```
int AbsoluteValue(int number) {  
    int retValue;  
    retValue = (number >= 0) ? number : -number;  
    return retValue;  
}
```

- Извикване на функцията

```
int main() {  
    int aNumber;  
    cout << "Input a number: "; cin >> aNumber;  
    aNumber = AbsoluteValue(aNumber);  
    cout << "The absolute value is: " << aNumber << endl;  
}
```


Алтернативи

```
int AbsoluteValue(int number) {  
    return (number >= 0) ? number : -number;  
}
```

```
int main() {  
    int aNumber;  
    cout << "Input a number: "; cin >> aNumber;  
  
    cout << "The absolute value is: " << AbsoluteValue(aNumber) << endl;  
}
```

// Една функция може и да не връща нищо

```
void PrintLine(unsigned int Size)
```

```
{
```

```
    for(int i=0; i < Size; i++)
```

```
    {
```

```
        cout << '-';
```

```
    }
```

```
}
```

```
int main()
```

```
{
```

```
    PrintLine(10); // ОК
```

```
    int x = PrintLine(20); // Грешка!
```

```
}
```

```
int MyFunction(int a)
{
    // Ако функцията има тип (в случая – int), то
    // всички пътища на изпълнение трябва да връщат
    // стойност от този тип!
    if (a > 0)
    {
        cout << "a is positive\n";
        return a;
    }
    else
    {
        cout << "a is negative\n";
    }
}
```

```
// Предполагаме, че ReadPositiveIntFromFile()  
// е функция, която прочита положително число от  
// някакъв файл и връща -1 ако не успее.
```

```
void MyFunction()  
{  
    int Data = ReadPositiveIntFromFile();  
  
    if (Data == -1)  
    {  
        cout << "ERROR: Cannot read from file!\n";  
        return;  
    }  
  
    // Тук използваме прочетените стойности ...  
}
```

```
int f();  
int g(int, int); // В декларацията можем да пропуснем  
                // имената на параметрите
```

```
int main()  
{  
    int a = f();  
    int b = g(5, 10);  
}
```

```
int f()  
{  
    return 10;  
}
```

```
int g(int a, int b)  
{  
    return a + b * 100;  
}
```

Дефиниция и декларация на функция

- Декларацията на функцията представлява нейната сигнатура, последвана от ;
 - Не е задължителна
 - Може да се каже, че декларацията е своеобразно „обещание“ за дефиниране на функцията по-късно в кода на програмата
- Дефиницията на функцията описва и нейното тяло
- Декларацията на дадена функция задължително трябва да предшества извикването и в друга функция
- Защо се налага да правим разлика между декларация и дефиниция

```
int f()                // Дефиниция на f()
{
    return 10;
}
```

```
int main()
{
    int a = f();        // ОК – f() е дефинирана по-горе
    int b = g(5, 10);   // Грешка! Не е ясно какво е g?
}
```

```
int g(int a, int b)    // Дефиниция на g()
{
    return a + b * 100;
}
```

```
int f()                // Дефиниция на f()
{
    return 10;
}
```

```
int g(int a, int b)    // Дефиниция на g()
{
    return a + b * 100;
}
```

```
int main()
{
    int a = f();        // ОК – f() е дефинирана по-горе
    int b = g(5, 10);   // ОК – g() е дефинирана по-горе
}
```



```
int f();           // Декларация на f()
int g(int a, int b); // Декларация на g()

int main()
{
    int a = f();      // ОК – функцията f() е декларирана по-горе
    int b = g(5, 10); // ОК – функцията g() е декларирана по-горе
}

int f()           // Дефиниция на f()
{
    return 10;
}

int g(int a, int b) // Дефиниция на g()
{
    return a + b * 100;
}
```

```
int f();           // Декларация на f()
int g(int a, int b); // Декларация на g()

int main()
{
    int a = f();    // ОК – функцията f() е декларирана по-горе
}

int f()            // Дефиниция на f()
{
    return g(10, 2); // ОК – функцията g() е декларирана по-горе
}

int g(int a, int b) // Дефиниция на g()
{
    return a + b * 100;
}
```

Област на видимост

```
void f()  
{  
    int b = 1000;  
    a = 2000; // Грешка!  
}
```

Област на видимост на b

```
int main()  
{  
    int a = 1;  
    f();  
    b = 2; // Грешка!  
    return 0;  
}
```

Област на видимост на a

```
#include <iostream>
using namespace std;

int c;

void f() {
    c = 200;
}

int main() {
    c = 100;
    f();
    cout << c << endl;
    return 0;
}
```

**Област на
видимост на c**

```
#include <iostream>
using namespace std;
```

```
int a = 0;
```

```
void f()
```

```
{
```

```
    int a = 10;
```

```
    cout << a << "-" << ::a << endl; // Извежда 10-0
```

```
}
```

```
int main()
```

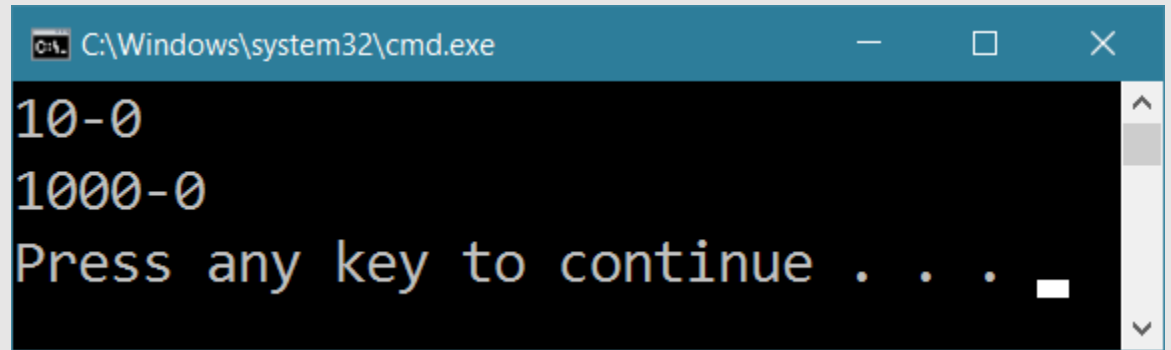
```
{
```

```
    int a = 1000;
```

```
    f();
```

```
    cout << a << "-" << ::a << endl; // Извежда 1000-0
```

```
}
```



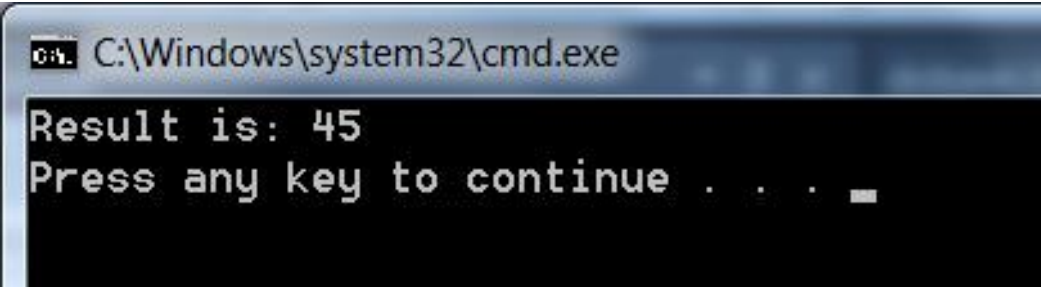
A screenshot of a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The window has a black background and white text. It displays the output of the C++ program: "10-0" on the first line, "1000-0" on the second line, and "Press any key to continue . . ." on the third line, followed by a small white cursor bar.

Ред на оценяване на параметрите

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

int main()
{
    int num = 10;

    cout << "Result is: "
         << Sum(num++, num+10, ++num)
         << endl;
    return 0;
}
```

A screenshot of a Windows command prompt window. The title bar shows the path 'C:\Windows\system32\cmd.exe'. The window contains the text 'Result is: 45' on the first line and 'Press any key to continue' on the second line, followed by a cursor.

Редът на оценяване на параметрите е неопределен и зависи от желанието на компилатора да оптимизира кода

Представяне в паметта

Разпределение на Оперативната Памет (ОП)

- Най-общо се състои от:
 - програмен код,
 - област на статичните данни,
 - област на динамичните данни
 - програмен стек

Програмен стек
Динамична памет
Статична памет
Програмен код
ОП за работа с ОС

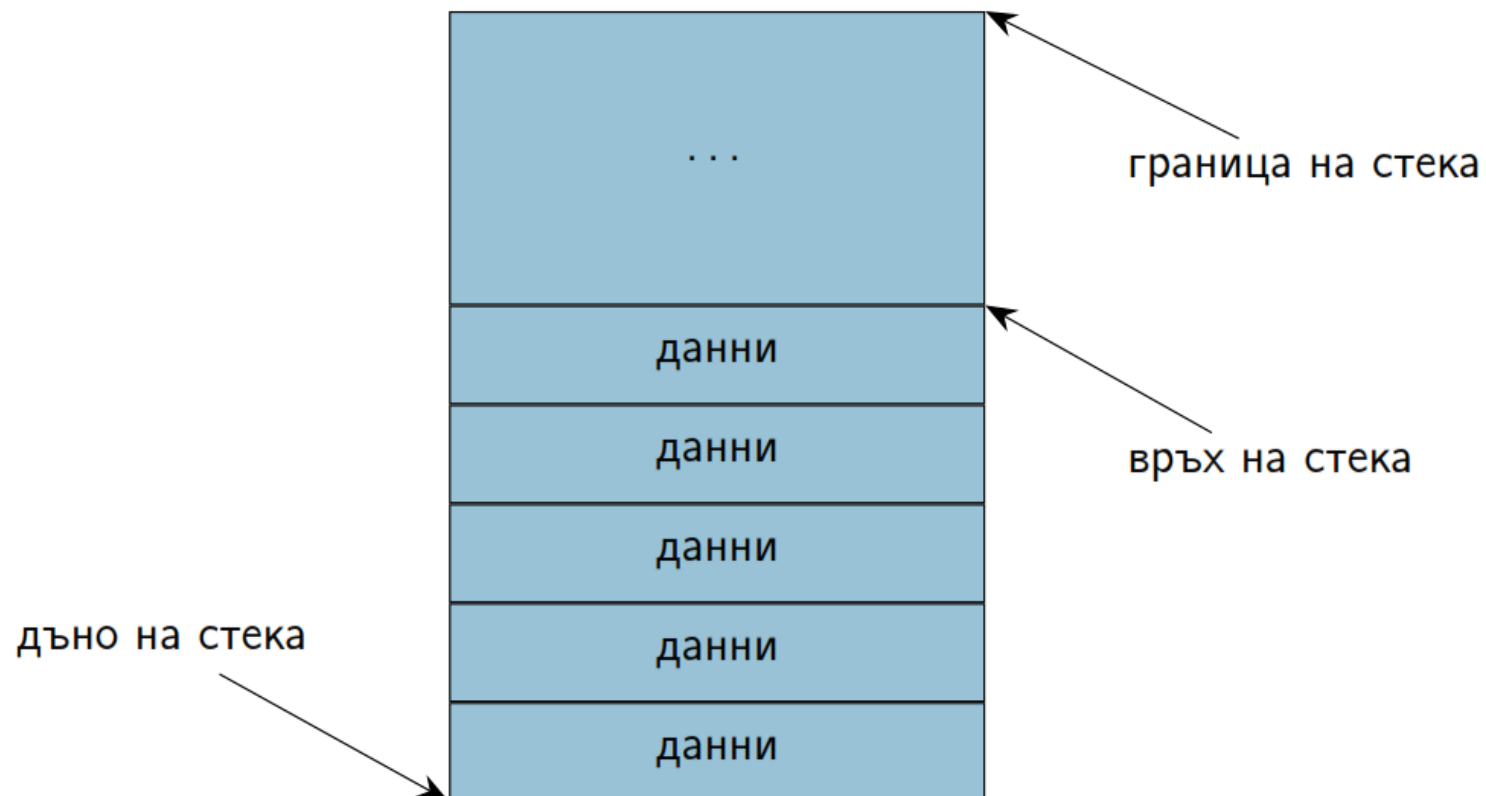
Разпределение на ОП за изпълнима програма ...

- Програмен код
 - В тази част е записан изпълнимият код на всички функции, изграждащи потребителската програма.
- Област на статичните данни
 - В нея са записани глобалните обекти (в широкия смисъл на думата) на програмата.
- Област на динамичните данни
 - Използва се за заделяне и освобождаване памет в процеса на изпълнение на програмата, а не преди това (при компилирането ѝ).
 - Служи за реализиране на динамични структури от данни (списъци, дървета, графи, ...)

Разпределение на ОП за изпълнима програма ...

- Програмен стек
 - Този вид памет съхранява данните на функциите на програмата.
 - Динамична структура, организирана по правилото “последен влязъл – пръв излязъл”.
- Стекови рамки
 - Това са елементите на програмния стек, своеобразни “блокове” от памет, които съхраняват данните, дефинирани в функциите

Програмен стек



Стекова рамка

- В дъното на стека е стековата рамка на `main()`.
- На върха на стека е стековата рамка на функцията, която се обработва в момента.
- Когато изпълнението на една функция завърши, нейната стекова рамка се отстранява от стека.

Стекова рамка

- Видът на стековата рамка зависи от реализацията. С точност до наредба, тя има вида:

Формални параметри
Адрес за връщане
Адрес на предходна рамка на стека
Локални променливи

Извикване на функция

```
int Compute(int a, int b)
{
    return a + b * 100;
}

int main()
{
    int x, y;

    x = 10; y = 25;
    cout << "Result is: " << Compute(x, y);

    return 0;
}
```

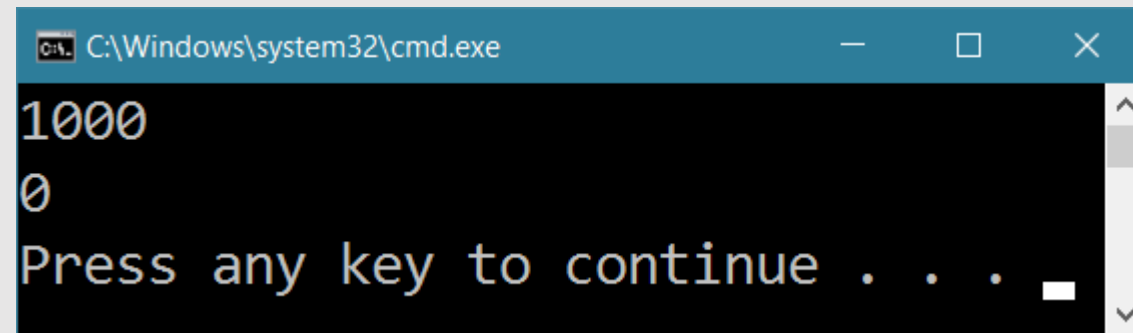


- Пресмята се стойността на фактическия параметър
- В стековата рамка на функцията се създава копие на стойността
- Всяка промяна на стойността остава локална за функцията
- При завършване на функцията, предадената стойност и всички промени над нея се загубват

// Предаване на параметри по стойност
// Функцията работи върху копие на оригиналните данни!

```
void f(int a)
{
    a = 1000;
    cout << a << endl;
}
```

```
int main()
{
    int a = 0;
    f(a);
    cout << a << endl;
    return 0;
}
```



A screenshot of a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The window has a black background with white text. It displays the output of the program: "1000" on the first line, "0" on the second line, and "Press any key to continue . . ." on the third line. A small white cursor is visible at the end of the third line.

Тип указатель и тип псевдоним

- Тип указатель

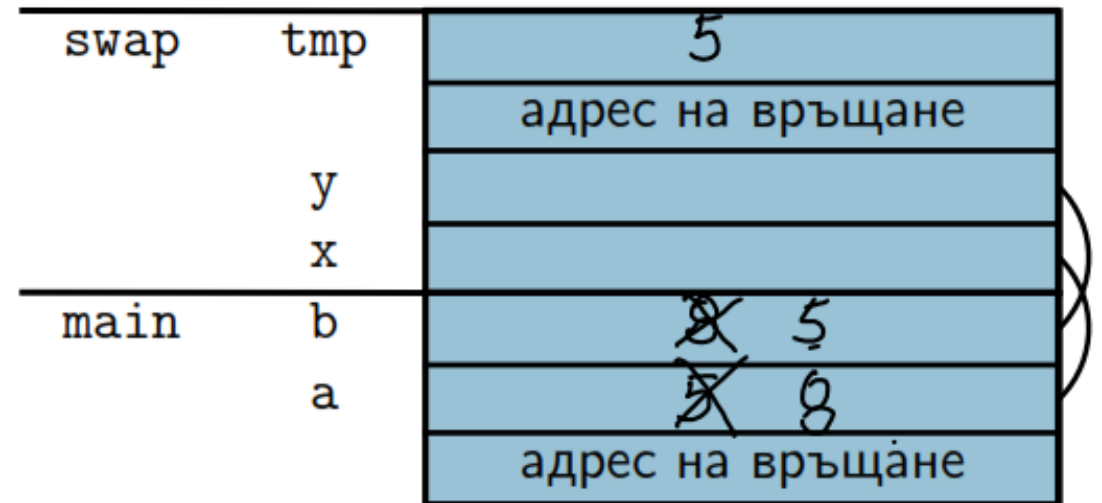
```
int a = 10;  
int *pa;
```

- Тип псевдоним (reference)

```
int a;  
int &ra = a;
```

Предаване на аргументи по псевдоним

```
void swap(int& x, int& y) {  
    int tmp = x;  
    x = y;  
    y = tmp;  
}  
  
int main() {  
    int a = 5, b = 8;  
    swap(a, b);  
    cout << a << ' ' << b << endl;  
}
```



Предаване на аргументи по адрес

```
void swap(int* p, int* q) {  
    int tmp = *p;  
    *p = *q;  
    *q = tmp;  
}  
  
int main() {  
    int a = 5, b = 8;  
    swap(&a, &b);  
    cout << a << ' ' << b << endl;  
}
```

```
void swap(int& x, int& y) {  
    int tmp = x;  
    x = y;  
    y = tmp;  
}  
  
int main() {  
    int a = 5, b = 8;  
    swap(a, b);  
    cout << a << ' ' << b << endl;  
}
```

// Предефиниране на функция (overload)

```
void MyFunction(int a)
{
    cout << "MyFunction (int)\n";
}
```

```
void MyFunction(double a)
{
    cout << "MyFunction (double)\n";
}
```

```
double MyFunction(int a, double b)
{
    cout << "MyFunction (int, double)\n";
    return (double)a * 10.0 + b;
}
```

```
int a = 5;  
double b = 10;
```

```
MyFunction(a);           // MyFunction(int)  
MyFunction(b);           // MyFunction(double)  
MyFunction(a, b);        // MyFunction(int, double)
```

```
MyFunction(10);          // MyFunction(int)
```

```
MyFunction(10.0);        // MyFunction(double)  
MyFunction((double)10);  // MyFunction(double)
```

```
MyFunction(10, 10.0);    // MyFunction(int, double)  
MyFunction(10.0, 10);    // MyFunction(double), WARNING!
```

// Функция НЕ МОЖЕ да се предефинира само по нейния тип.
// Трябва да има и разлика в броя или вида на аргументите.

```
int MyFunction(int a)
{
    cout << "MyFunction(int)\n";
}
```



```
double MyFunction(int a)
{
    cout << "MyFunction(double)\n";
}
```

```
int main()
{
    MyFunction(10); // Един пример защо това не е позволено.
                  // Коя версия бихме извикали тук?
}
```

Параметри по подразбиране

```
int MyFunction(int a, int b = 0)
{
    return a + b;
}
```

```
int main()
{
    int x = MyFunction(10, 20); // x = 30;
    int y = MyFunction(10);      // y = 10;
    int z = MyFunction();        // Грешка!
}
```




// Подразбиращите се параметри трябва винаги
// да бъдат в края!

```
int MyFunction(int a = 0, int b) // Грешка!  
{  
    return a + b;  
}
```

```
int MyFunction(int a = 10, int b = 10)
{
    return a + b;
}
```



```
int MyFunction()
{
    return 0;
}
```

```
int main()
{
    int x = MyFunction(); // Грешка!
}
```

Inline функции

- Записването на аргументите на функцията в стека и неговото управление след това отнемат време.
- Това може да се избегне ако функцията се дефинира като `inline`
- Ключовата дума `inline` указва на компилатора, че е препоръчително тялото на функцията да се замести директно на мястото на извикването ѝ.
 - Това не е задължително, тъй като зависи от възможностите и „желанието“ на компилатора да оптимизира кода
- Обикновено се прави за прости функции

Пример

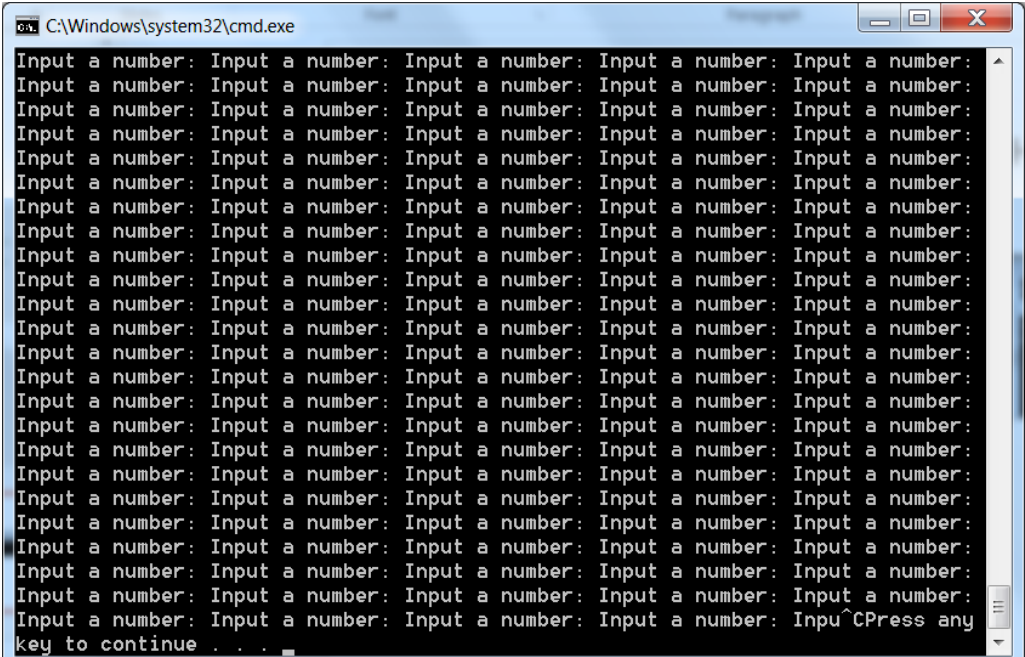
```
inline int AbsoluteValue(int number) {  
    return (number >= 0) ? number : -number;  
}
```

Контрол на входните данни

```
int main()
{
    int height; // Should be between 1 and 25
    do {
        cout << "Input a number: "; cin >> height;
    } while ((height < 1) || (height > 25));

    cout << "Height is: " << height << endl;

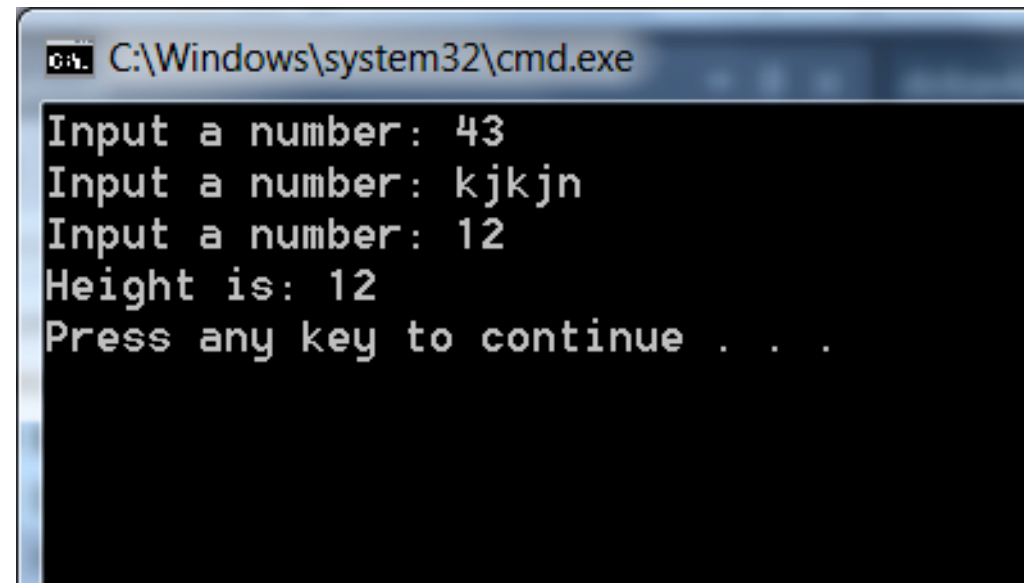
    return 0;
}
```



A screenshot of a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The window displays a program that repeatedly prompts the user with "Input a number:". The text is repeated many times, filling the window. At the bottom, it says "key to continue . . .".

Контрол на входните данни

```
int SafelyInputInteger(int lowerBound, int upperBound) {  
    int intNumber;  
  
    do {  
        cout << "Input a number: "; cin >> intNumber;  
  
        if (cin.fail()) {  
            cin.clear();  
            cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');  
            continue;  
        }  
    } while ((intNumber < lowerBound) || (intNumber > upperBound));  
  
    return intNumber;  
}  
  
int main()  
{  
    int height; // Should be between 1 and 25  
  
    height = SafelyInputInteger(1, 25);  
    cout << "Height is: " << height << endl;  
  
    return 0;  
}
```



```
C:\Windows\system32\cmd.exe  
Input a number: 43  
Input a number: kjkjn  
Input a number: 12  
Height is: 12  
Press any key to continue . . .
```

- За подготовката на тази презентация са използвани слайдове на:
 - Доц. Александър Григоров
 - Доц. Атанас Семерджиев
 - Доц. Трифон Трифонов