Çöü
öüşü
ışıığı
Çöüü
şüü
char =
200; //−
56olur!intyanlis_sonuc =
yanlis_char; //−
56
Ğ
char =
200; //200kalırintdogru_sonuc =
dogru_char; //200
şı
ast <
int >
(yanlis_char) <<
endl; cout <<
"int'edönüşüm : " << yanlis_sonuc << endl;
Ğ
şı
ast <
int >
(dogru_char) <<
endl; cout <<
"int'edönüşüm : " << dogru_sonuc << endl;
üç
çöü
ğ
ığıı
ışı
char =
static_cast <
unsignedchar >
(value); returnstatic_cast <
int >
(safe_char); ;
üç
öüşü
ış
öüşüüü
öüşü
şüğğ
sonuc =
static_cast <
int >
(rakam); //55(ASCIIdeğeri)
Ğ
ğ
sonuc =
rakam−'
0'; //7(sayısaldeğer)
şü
sonuc <<
endl; cout <<
"DOĞRU(Sayısal) : " << dogru_sonuc << endl;
Ç
ı
dizisi[] =
"123";
şü
ı
ç
şü
Ç
öüşüüü
dizisi[i]! ='
'; i+
+)intascii_val = static_cast < int > (rakam_dizisi[i]); cout << "'" << rakam_dizisi[i] << "'− > " << ascii_val << endl;
Ğ
ı
ğ
ş
Ğ
Ç
öüşüüü
ayi =
0; for(inti =
0; rakam_dizisi[i]! ='
'; i+
+)intdigit = rakam_dizisi[i] −'0'; toplam_ayi = toplam_ayi * 10 + digit; cout << "Adım" << i + 1 << ".' " << rakam

$_c ases[i] <<$
$"æ"; if(success) cout << result << "æ"; else cout << "HATAoır"; cout << endl;$
$ğıı$
$ı$
$üç$
$_c hars[] =$
$"_cğıösüÇĞIİÖÜŞ";$
$ı$
$ış$
$_c hars[i]! ='$
$'; i+$
$+) charch = turkce_c hars[i]; boolascii_a lpha = (ch >=' A' ch <=' Z') || (ch >=' a' ch <=' z');$
$_c ast <$
$int >$
$(static_c ast <$
$unsignedchar >$
$(ch)) <<$
$")- >$
$Harf?" <<$
$(ascii_a lpha?"Evet" :$
$"Hayır") <<$
$endl;$
$ğıı$
$çöü$
$ğıı$
$_A LL, "tr_T R.UTF-$
$8"); //Türkçelocale$
$_c hars[i]! ='$
$'; i+$
$+) unsignedcharch = static_c ast < unsignedchar > (turkce_c hars[i]); boollocale_a lpha = isalpha(ch);$
$_c hars[i] <<$
$"'- >$
$Harf?" <<$
$(locale_a lpha?"Evet" :$
$"Hayır") <<$
$endl;$
$üş$
$_m ixed[] =$
$"Hello123cğABC"; for(inti = 0; test_m ixed[i]! = ' '; i + +) charch = test_m ixed[i]; cout << "'" << ch << "'- > D$

KaynakHedefOtomatikVeri Kaybı

		ı	ı
		ı	ü
			ğ

$öüşü$
$öüşü$
$ı$

CharASCIICharASCIICharASCII

$ı$ Ö

KarakterASCIIEscape SequenceAçıklama

	\		
	\		
	\		ı
			ş
			Ü
	\		Çı
$			
			ü
			ş

Ö
$öüşü$

Dönüşüm YöntemiPerformansGüvenlikÖnerilen Kullanım

ıı			öüşü
ıı		ü	ı
ıı		üşü	ğ
ıı			

$öüşü$
$ö$
$şışıı$
$şş$
$şş$
$Iış$
$şş$
$öüşü$
$ş$
$_t) wchar_t wide_c hars[] =$
$L"Acğıöüşş ÇĞIİÖÜŞ123";$
$ş$
$_c hars[i]! =$
$L''; i+$
$+) wchar_t wch = wide_c hars[i]; intunicode_v al = static_c ast < int > (wch);$
$ığı$
$ı$
$_v al <=$

C++ Veri Tipi Dönüşümleri Kılavuzu

Programlama Referans Kılavuzu

20 Eylül 2025

İçindekiler

1 Giriş

C++ programlama dilinde veri tipi dönüşümleri, farklı veri tipleri arasında değer aktarımı yapmak için kullanılan temel işlemlerdir. Bu kılavuz, C++'da mevcut olan tüm dönüşüm türlerini detaylı bir şekilde açıklamaktadır.

2 Temel Veri Tipleri

C++'da başlıca veri tipleri şunlardır:

- `int` - Tam sayılar
- `float` - Tek duyarlık ondalık sayılar
- `double` - Çift duyarlık ondalık sayılar
- `char` - Karakterler
- `bool` - Boolean değerler
- `long`, `short`, `unsigned` varyantları

3 Dönüşüm Türleri

3.1 Otomatik (Implicit) Dönüşüm

Otomatik dönüşüm, derleyici tarafından otomatik olarak yapılan dönüşümlerdir.

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

int main() {
    // int'ten double'a otomatik d  n     m
    int sayi = 42;
    double ondalik = sayi;  // Otomatik d  n     m

    // char'dan int'e otomatik d  n     m
    char karakter = 'A';
    int ascii_degeri = karakter;  // ASCII de eri: 65

    // bool'dan int'e otomatik d  n     m
    bool dogru = true;
    int sayi_degeri = dogru;  // 1 olur

    cout << "Ondal k: " << ondalik << endl;
    cout << "ASCII: " << ascii_degeri << endl;
    cout << "Bool de eri: " << sayi_degeri << endl;

    return 0;
}
```

Listing 1: Otomatik Dönüşüm Örnekleri

4 Char'dan Integer'a Dönüşüm - Detaylı İnceleme

4.1 ASCII Karakter Sistemi ve Char Veri Tipi

C++'da `char` veri tipi aslında 8-bit'lik bir tam sayı türüdür. ASCII (American Standard Code for Information Interchange) tablosuna göre her karakter bir sayısal değere karşılık gelir.

4.1.1 ASCII Değer Aralıkları

- **0-31**: Kontrol karakterleri (null, tab, newline vb.)
- **32-47**: Boşluk ve özel karakterler
- **48-57**: Rakamlar ('0'-'9')
- **65-90**: Büyük harfler ('A'-'Z')
- **97-122**: Küçük harfler ('a'-'z')
- **123-127**: Diğer özel karakterler

4.2 Char'dan Int'e Dönüşüm Yöntemleri

4.2.1 Otomatik Dönüşüm (Implicit Conversion)

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

int main() {
    // Temel harfler
    char buyuk_harf = 'A';
    char kucuk_harf = 'a';
    char rakam_char = '5';

    // Otomatik d  n     m
    int ascii_A = buyuk_harf;     // 65
    int ascii_a = kucuk_harf;     // 97
    int ascii_5 = rakam_char;     // 53

    cout << "Karakter 'A' -> ASCII: " << ascii_A << endl;
    cout << "Karakter 'a' -> ASCII: " << ascii_a << endl;
    cout << "Karakter '5' -> ASCII: " << ascii_5 << endl;

    //  zel  karakterler
    char ozel_karakterler[] = {'!', '@', '#', '

\subsection{Performans Kar  la t rmas }

\begin{lstlisting}[caption=Char D n    mlerinde Performans
   Testi]
```

```cpp
#include <iostream>
#include <chrono>
#include <vector>
using namespace std;
using namespace std::chrono;

int main() {
    const int TEST_SIZE = 1000000;
    vector<char> test_chars;

    // Test verisi haz rlama
    for (int i = 0; i < TEST_SIZE; i++) {
        test_chars.push_back('A' + (i % 26));
    }

    // Otomatik d  n     m testi
    auto start1 = high_resolution_clock::now();
    volatile int sum1 = 0;
    for (char ch : test_chars) {
        sum1 += ch; // Otomatik d  n     m
    }
    auto end1 = high_resolution_clock::now();

    // static_cast testi
    auto start2 = high_resolution_clock::now();
    volatile int sum2 = 0;
    for (char ch : test_chars) {
        sum2 += static_cast<int>(ch);
    }
    auto end2 = high_resolution_clock::now();

    // C tarz  d  n     m testi
    auto start3 = high_resolution_clock::now();
    volatile int sum3 = 0;
    for (char ch : test_chars) {
        sum3 += (int)ch;
    }
    auto end3 = high_resolution_clock::now();

    cout << "Performans Test Sonu lar  (" << TEST_SIZE << "
       i lem):" << endl;
    cout << "Otomatik d  n     m: "
         << duration_cast<microseconds>(end1 - start1).count()
         << " mikrosaniye" << endl;
    cout << "static_cast: "
         << duration_cast<microseconds>(end2 - start2).count()
         << " mikrosaniye" << endl;
    cout << "C tarz  cast: "
         << duration_cast<microseconds>(end3 - start3).count()
         << " mikrosaniye" << endl;

```

```
76      return 0;
77  }
```

Listing 2: Char'dan Int'e Otomatik Dönüşüm

4.3 Pratik Kullanım Örnekleri

4.3.1 Büyük/Küçük Harf Dönüşümü

```
1  #include <iostream>
2  #include <cctype>
3  using namespace std;
4
5  // Manuel b y k harf d n   t rme
6  char toBigLetter(char ch) {
7      if (ch >= 'a' && ch <= 'z') {
8          return ch - ('a' - 'A'); // ASCII fark : 32
9      }
10     return ch;
11 }
12
13 // Manuel k     k harf d n   t rme
14 char toSmallLetter(char ch) {
15     if (ch >= 'A' && ch <= 'Z') {
16         return ch + ('a' - 'A'); // ASCII fark : 32
17     }
18     return ch;
19 }
20
21 int main() {
22     char test_chars[] = "AbC123dEf!";
23
24     cout << "Orijinal: " << test_chars << endl;
25     cout << "Manuel B y k Harf: ";
26     for (int i = 0; test_chars[i] != '\0'; i++) {
27         cout << toBigLetter(test_chars[i]);
28     }
29     cout << endl;
30
31     cout << "Manuel K     k Harf: ";
32     for (int i = 0; test_chars[i] != '\0'; i++) {
33         cout << toSmallLetter(test_chars[i]);
34     }
35     cout << endl;
36
37     // Standart k t phane fonksiyonlar ile kar   la t rma
38     cout << "Std B y k Harf: ";
39     for (int i = 0; test_chars[i] != '\0'; i++) {
40         cout << static_cast<char>(toupper(test_chars[i]));
41     }
42     cout << endl;
```

```
43
44      cout << "Std  K      k  Harf: ";
45      for (int i = 0; test_chars[i] != '\0'; i++) {
46          cout << static_cast<char>(tolower(test_chars[i]));
47      }
48      cout << endl;
49
50      // ASCII de erlerini  g ster
51      cout << "\nASCII Fark Analizi:" << endl;
52      cout << "'A' ASCII: " << static_cast<int>('A') << endl;
53      cout << "'a' ASCII: " << static_cast<int>('a') << endl;
54      cout << "Fark: " << static_cast<int>('a') -
           static_cast<int>('A') << endl;
55
56      return 0;
57  }
```

Listing 3: Char Dönüşümleriyle Harf Dönüştürme

4.3.2 Hex Karakter Dönüşümü

```
1   #include <iostream>
2   #include <iomanip>
3   using namespace std;
4
5   // Hex karakterini  say ya  d   n    t   r
6   int hexCharToInt(char hex_char) {
7       if (hex_char >= '0' && hex_char <= '9') {
8           return hex_char - '0';
9       }
10      else if (hex_char >= 'A' && hex_char <= 'F') {
11          return hex_char - 'A' + 10;
12      }
13      else if (hex_char >= 'a' && hex_char <= 'f') {
14          return hex_char - 'a' + 10;
15      }
16      return -1; // Ge ersiz  karakter
17  }
18
19  // Say  y    hex karakterine  d   n    t   r
20  char intToHexChar(int value) {
21      if (value >= 0 && value <= 9) {
22          return '0' + value;
23      }
24      else if (value >= 10 && value <= 15) {
25          return 'A' + (value - 10);
26      }
27      return '?'; // Ge ersiz  de er
28  }
29
30  int main() {
```

```
    char hex_chars[] = {'0', '5', 'A', 'F', 'a', 'f', 'G', '9'};

    cout << "Hex Karakter D  n     m Tablosu:" << endl;
    cout << "Hex Char\tInt De er\tGeri  D  n     m" << endl;
    cout << "--------\t---------\t-----------" << endl;

    for (int i = 0; i < 8; i++) {
        char hex_ch = hex_chars[i];
        int int_val = hexCharToInt(hex_ch);
        char back_to_hex = (int_val != -1) ?
           intToHexChar(int_val) : '?';

        cout << "'" << hex_ch << "'\t\t";
        if (int_val != -1) {
            cout << int_val << "\t\t'" << back_to_hex << "'";
        } else {
            cout << "Ge ersiz\t'?'";
        }
        cout << endl;
    }

    // Hex string'i integer'a d  n   t  rme
    char hex_string[] = "1A2B";
    int hex_value = 0;

    cout << "\nHex String  D  n     m  : \"" << hex_string <<
       "\"" << endl;
    for (int i = 0; hex_string[i] != '\0'; i++) {
        int digit_val = hexCharToInt(hex_string[i]);
        if (digit_val != -1) {
            hex_value = hex_value * 16 + digit_val;
            cout << "Ad m " << i+1 << ": '" << hex_string[i]
                 << "' -> " << digit_val
                 << ", Toplam: " << hex_value << endl;
        }
    }

    cout << "Final Sonu : " << hex_value
         << " (Decimal), 0x" << hex << hex_value << " (Hex)" <<
            endl;

    return 0;
}
```

Listing 4: Hexadecimal Karakter Dönüşümleri

4.4 Char Dönüşümlerinde Güvenlik Önlemleri

4.4.1 Buffer Overflow Koruması

```
#include <iostream>
```

```cpp
#include <cstring>
#include <climits>
using namespace std;

class SafeCharToIntConverter {
private:
    static const int MAX_DIGITS = 10; // int i in maksimum digit say s

public:
    static bool isValidDigitChar(char ch) {
        return (ch >= '0' && ch <= '9');
    }

    static bool safeConvert(const char* str, int max_len, int& result) {
        if (str == nullptr || max_len <= 0) {
            return false;
        }

        result = 0;
        int sign = 1;
        int i = 0;

        //   aret   kontrol
        if (str[0] == '-') {
            sign = -1;
            i = 1;
        } else if (str[0] == '+') {
            i = 1;
        }

        // Uzunluk kontrol
        int digit_count = 0;
        while (i < max_len && str[i] != '\0' && digit_count < MAX_DIGITS) {
            if (!isValidDigitChar(str[i])) {
                return false; // Ge ersiz karakter
            }

            int digit = str[i] - '0';

            // Ta ma kontrol
            if (result > (INT_MAX - digit) / 10) {
                return false; // Ta ma riski
            }

            result = result * 10 + digit;
            i++;
            digit_count++;
        }
```

```
50
51          result *= sign;
52          return true;
53      }
54  };
55
56  int main() {
57      const int BUFFER_SIZE = 50;
58      char test_inputs[][BUFFER_SIZE] = {
59          "12345",        // Normal
60          "-6789",        // Negatif
61          "2147483647",   // INT_MAX
62          "2147483648",   // Ta ma
63          "123abc",       // Ge ersiz karakter
64          "",             // Bo  string
65          "+999"          // Pozitif i aret
66      };
67
68      cout << "G venli Char to Int D n     m Testi:" << endl;
69      cout << "Giri \t\t\tSonu \t\tBa ar l ?" << endl;
70      cout << "-----\t\t\t-----\t\t--------" << endl;
71
72      for (int i = 0; i < 7; i++) {
73          int result;
74          bool success = SafeCharToIntConverter::safeConvert(
75              test_inputs[i], BUFFER_SIZE, result);
76
77          cout << "\"" << test_inputs[i] << "\"\t\t";
78          if (success) {
79              cout << result << "\t\tEvet";
80          } else {
81              cout << "N/A\t\tHay r";
82          }
83          cout << endl;
84      }
85
86      return 0;
87  }
```

Listing 5: Güvenli Char Array İşlemleri

4.4.2 C Tarzı Dönüşüm (Type Casting)

```
1  #include <iostream>
2  using namespace std;
3
4  int main() {
5      double pi = 3.14159;
6
7      // C tarz   d  n     m
8      int tam_sayi = (int)pi;  // 3 olur (kesir k s m  kaybolur)
```

```
9
10      // Farkl   yaz m   ekli
11      int tam_sayi2 = int(pi);
12
13      cout << "Orijinal: " << pi << endl;
14      cout << " D   n    t   r   l m     : " << tam_sayi << endl;
15
16      return 0;
17  }
```

Listing 6: C Tarzı Dönüşüm

4.4.3 C++ Tarzı Dönüşümler

C++ dört farklı açık dönüşüm operatörü sunar:

static_cast En yaygın kullanılan ve güvenli dönüşüm yöntemidir.

```
1  #include <iostream>
2  using namespace std;
3
4  int main() {
5      // Say sal  d   n      m ler
6      double d = 3.14;
7      int i = static_cast<int>(d);
8
9      // Pointer  d   n      m leri
10     int sayi = 100;
11     void* void_ptr = static_cast<void*>(&sayi);
12     int* int_ptr = static_cast<int*>(void_ptr);
13
14     cout << "Double: " << d << ", Int: " << i << endl;
15     cout << "Pointer de eri: " << *int_ptr << endl;
16
17     return 0;
18 }
```

Listing 7: static_cast Kullanımı

dynamic_cast Polimorfik sınıflar için runtime tip kontrolü yapar.

```
1  #include <iostream>
2  using namespace std;
3
4  class Base {
5  public:
6      virtual ~Base() {}
7  };
8
9  class Derived : public Base {
10 public:
11     void ozel_fonksiyon() {
```

```
        cout << "Derived s n f n n zel fonksiyonu" << endl;
    }
};

int main() {
    Base* base_ptr = new Derived();

    // G venli downcast
    Derived* derived_ptr = dynamic_cast<Derived*>(base_ptr);

    if (derived_ptr != nullptr) {
        derived_ptr->ozel_fonksiyon();
    } else {
        cout << " D n m ba ar s z!" << endl;
    }

    delete base_ptr;
    return 0;
}
```

Listing 8: dynamic_cast Kullanımı

const_cast const niteleyicisini eklemek veya kaldırmak için kullanılır.

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

void degistir(int* ptr) {
    *ptr = 100;
}

int main() {
    const int sayi = 42;

    // const' kald rma (dikkatli kullan lmal !)
    int* degistirilebilir_ptr = const_cast<int*>(&sayi);

    cout << "Orijinal: " << sayi << endl;

    // Not: const de i keni de i tirmek tan ms z
       davran !
    // Bu rnek sadece syntax' g stermek i indir

    return 0;
}
```

Listing 9: const_cast Kullanımı

reinterpret_cast Düşük seviye bit düzeyinde dönüşümler için kullanılır.

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

```cpp

int main() {
    int sayi = 1234567890;

    // int'i char dizisi olarak yorumlama
    char* char_ptr = reinterpret_cast<char*>(&sayi);

    cout << "Say : " << sayi << endl;
    cout << "Byte'lar: ";
    for (int i = 0; i < sizeof(int); i++) {
        cout << static_cast<int>(char_ptr[i]) << " ";
    }
    cout << endl;

    return 0;
}
```

Listing 10: reinterpret_cast Kullanımı

5 String Dönüşümleri

5.1 Sayıdan String'e

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

int main() {
    int sayi = 123;
    double ondalik = 45.67;

    // C++11 ve sonras
    string str1 = to_string(sayi);
    string str2 = to_string(ondalik);

    cout << "Int to string: " << str1 << endl;
    cout << "Double to string: " << str2 << endl;

    return 0;
}
```

Listing 11: Sayıdan String'e Dönüşüm

5.2 String'den Sayıya

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

```

```cpp
int main() {
    string str_sayi = "123";
    string str_ondalik = "45.67";

    // C++11 ve sonras
    int sayi = stoi(str_sayi);
    double ondalik = stod(str_ondalik);

    cout << "String to int: " << sayi << endl;
    cout << "String to double: " << ondalik << endl;

    // Hata kontrol  ile
    try {
        int gecersiz = stoi("abc123");
    } catch (const invalid_argument& e) {
        cout << "Ge ersiz arg man hatas !" << endl;
    }

    return 0;
}
```

Listing 12: String'den Sayıya Dönüşüm

6 Dönüşüm Tablosu

Kaynak	Hedef	Otomatik	Veri Kaybı
int	double	Evet	Hayır
double	int	Hayır	Evet
char	int	Evet	Hayır
int	char	Hayır	Evet (büyük değerler)
bool	int	Evet	Hayır
int	bool	Evet	Evet

Tablo 1: Temel Dönüşüm Tablosu

7 En İyi Uygulamalar

7.1 Char-Integer Dönüşümlerinde En İyi Uygulamalar

1. **Güvenli Dönüşüm**:
   - Rakam karakterleri için: `ch - '0'` yöntemini kullanın
   - ASCII değeri için: `static_cast<int>(ch)` kullanın
   - Taşma kontrolü yapın, özellikle signed char için
2. **Veri Kaybı Kontrolü**:
   - Int'den char'a dönüşümde değer aralığını kontrol edin (0-255 veya -128 ile 127)

- Büyük integer değerleri için uyarı verin

3. **Tip Güvenliği**:
   - Unsigned char kullanarak negatif değer problemlerini önleyin
   - static_cast kullanarak açık dönüşüm yapın
4. **Performans**:
   - Büyük dizilerde otomatik dönüşümü tercih edin
   - Gereksiz cast işlemlerinden kaçının
5. **Hata Yönetimi**:
   - Geçersiz karakter girişlerini kontrol edin
   - Buffer overflow koruması ekleyin
   - Return değerleri ile başarı durumunu bildirin

7.2 Char Dönüşümlerinde Tavsiye Edilen Kalıplar

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

// Kal p 1: G venli rakam karakteri d  n      m
inline int safeDigitToInt(char ch) {
    return (ch >= '0' && ch <= '9') ? (ch - '0') : -1;
}

// Kal p 2: G venli ASCII d  n      m
inline int safeCharToASCII(char ch) {
    return static_cast<unsigned char>(ch);
}

// Kal p 3: G venli int-to-char d  n      m
inline bool safeIntToChar(int value, char& result) {
    if (value >= 0 && value <= 255) {
        result = static_cast<char>(value);
        return true;
    }
    return false;
}

// Kal p 4: Harf kontrol  ile case conversion
inline char safeToUpper(char ch) {
    return (ch >= 'a' && ch <= 'z') ? (ch - 32) : ch;
}

inline char safeToLower(char ch) {
    return (ch >= 'A' && ch <= 'Z') ? (ch + 32) : ch;
}
```

```
32
33 // Kal p 5: Karakter  s  n  f   kontrol
34 inline bool isAlpha(char ch) {
35     return (ch >= 'A' && ch <= 'Z') || (ch >= 'a' && ch <= 'z');
36 }
37
38 inline bool isDigit(char ch) {
39     return (ch >= '0' && ch <= '9');
40 }
41
42 inline bool isAlphaNumeric(char ch) {
43     return isAlpha(ch) || isDigit(ch);
44 }
45
46 int main() {
47     // Test   rnekleri
48     char test_chars[] = {'5', 'A', 'z', '@', '150', '\n'};
49
50     cout << "G venli  D  n     m  Testleri:" << endl;
51     cout << "Karakter\tRakam?\tASCII\tB  y  k\tK     k\tAlfa?"
          << endl;
52     cout << "--------\t------\t-----\t-----\t-----\t-----" <<
          endl;
53
54     for (char ch : test_chars) {
55         if (ch == '\n') continue; // Newline karakterini atla
56
57         int digit = safeDigitToInt(ch);
58         int ascii = safeCharToASCII(ch);
59         char upper = safeToUpper(ch);
60         char lower = safeToLower(ch);
61         bool alpha = isAlpha(ch);
62
63         cout << "'" << ch << "'\t\t"
64              << (digit != -1 ? to_string(digit) : "N/A") << "\t"
65              << ascii << "\t"
66              << "'" << upper << "'\t"
67              << "'" << lower << "'\t"
68              << (alpha ? "Evet" : "Hay r") << endl;
69     }
70
71     return 0;
72 }
73 \end{lstlisting>
74
75 \section{Yayg n Hatalar ve    zmler   }
76
77 \subsection{Ta ma (Overflow) Hatas }
78
79 \begin{lstlisting}[caption=Ta ma Hatas  ve Kontrol ]
80 #include <iostream>
```

```cpp
#include <limits>
using namespace std;

int main() {
    long long buyuk_sayi = 2147483648LL; // int'in maksimum
       de erinden b y k

    // Kontrol yaparak d n m
    if (buyuk_sayi <= numeric_limits<int>::max()) {
        int kucuk_sayi = static_cast<int>(buyuk_sayi);
        cout << "G venli d n m : " << kucuk_sayi << endl;
    } else {
        cout << "D n m g venli de il , ta ma riski!" <<
           endl;
    }

    return 0;
}
```

Listing 13: Önerilen Dönüşüm Kalıpları

7.3 Hassasiyet Kaybı

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

int main() {
    double hassas_sayi = 3.14159265359;
    float az_hassas = static_cast<float>(hassas_sayi);

    double fark = abs(hassas_sayi -
       static_cast<double>(az_hassas));

    cout << "Orijinal: " << hassas_sayi << endl;
    cout << "Float: " << az_hassas << endl;
    cout << "Hassasiyet kayb : " << fark << endl;

    return 0;
}
```

Listing 14: Hassasiyet Kaybı Kontrolü

8 Sonuç

C++ veri tipi dönüşümleri, programcılara büyük esneklik sağlar ancak dikkatli kullanılmalıdır. Güvenli dönüşüm yöntemlerini tercih etmek, hata kontrolü yapmak ve veri kaybı risklerini göz önünde bulundurmak önemlidir.

Bu kılavuzda örneklenen tüm dönüşüm türlerini anlayarak, C++ programlarınızda daha etkili ve güvenli kod yazabilirsiniz.