Линейные алгоритмы

Двоичный дамп

```cpp
// C++ Standard Operators
// Binary dump

#include <iostream>
#include <bitset>

int main()
{
  using namespace std;

  int n(13);

  cout << bitset<32>(n) << '\n' << endl;

  cout << (n < 0 ? '1' : '0'); // 31
  n <<= 1;
  cout << (n < 0 ? '1' : '0'); // 30
  n <<= 1;
  cout << (n < 0 ? '1' : '0'); // 29
  n <<= 1;
  cout << (n < 0 ? '1' : '0'); // 28
  n <<= 1;
  cout << (n < 0 ? '1' : '0'); // 27
  n <<= 1;
  cout << (n < 0 ? '1' : '0'); // 26
  n <<= 1;
  cout << (n < 0 ? '1' : '0'); // 25
  n <<= 1;
  cout << (n < 0 ? '1' : '0'); // 24
  n <<= 1;
  cout << (n < 0 ? '1' : '0'); // 23
  n <<= 1;
  cout << (n < 0 ? '1' : '0'); // 22
  n <<= 1;
  cout << (n < 0 ? '1' : '0'); // 21
  n <<= 1;
  cout << (n < 0 ? '1' : '0'); // 20
  n <<= 1;
  cout << (n < 0 ? '1' : '0'); // 19
  n <<= 1;
  cout << (n < 0 ? '1' : '0'); // 18
  n <<= 1;
  cout << (n < 0 ? '1' : '0'); // 17
  n <<= 1;
  cout << (n < 0 ? '1' : '0'); // 16
  n <<= 1;
```

```
  cout << (n < 0 ? '1' : '0'); // 15
  n <<= 1;
  cout << (n < 0 ? '1' : '0'); // 14
  n <<= 1;
  cout << (n < 0 ? '1' : '0'); // 13
  n <<= 1;
  cout << (n < 0 ? '1' : '0'); // 12
  n <<= 1;
  cout << (n < 0 ? '1' : '0'); // 11
  n <<= 1;
  cout << (n < 0 ? '1' : '0'); // 10
  n <<= 1;
  cout << (n < 0 ? '1' : '0'); // 9
  n <<= 1;
  cout << (n < 0 ? '1' : '0'); // 8
  n <<= 1;
  cout << (n < 0 ? '1' : '0'); // 7
  n <<= 1;
  cout << (n < 0 ? '1' : '0'); // 6
  n <<= 1;
  cout << (n < 0 ? '1' : '0'); // 5
  n <<= 1;
  cout << (n < 0 ? '1' : '0'); // 4
  n <<= 1;
  cout << (n < 0 ? '1' : '0'); // 3
  n <<= 1;
  cout << (n < 0 ? '1' : '0'); // 2
  n <<= 1;
  cout << (n < 0 ? '1' : '0'); // 1
  n <<= 1;
  cout << (n < 0 ? '1' : '0')
     << endl;                  // 0

  return 0;
}
```

Двоичный дамп

```
// C++ Standard Operators
// Binary dump

#include <iostream>
#include <bitset>

int main()
{
  using namespace std;

  int n(13);

  cout << bitset<32>(n) << '\n' << endl;

  int mask(010000000000);

  cout << (n < 0 ? '1' : '0');            // 31
  cout << ((n & mask) != 0 ? '1' : '0'); // 30
  mask >>= 1;
  cout << ((n & mask) != 0 ? '1' : '0'); // 29
  mask >>= 1;
  cout << ((n & mask) != 0 ? '1' : '0'); // 28
  mask >>= 1;
  cout << ((n & mask) != 0 ? '1' : '0'); // 27
  mask >>= 1;
  cout << ((n & mask) != 0 ? '1' : '0'); // 26
  mask >>= 1;
  cout << ((n & mask) != 0 ? '1' : '0'); // 25
  mask >>= 1;
  cout << ((n & mask) != 0 ? '1' : '0'); // 24
  mask >>= 1;
  cout << ((n & mask) != 0 ? '1' : '0'); // 23
  mask >>= 1;
  cout << ((n & mask) != 0 ? '1' : '0'); // 22
  mask >>= 1;
  cout << ((n & mask) != 0 ? '1' : '0'); // 21
  mask >>= 1;
  cout << ((n & mask) != 0 ? '1' : '0'); // 20
  mask >>= 1;
  cout << ((n & mask) != 0 ? '1' : '0'); // 19
  mask >>= 1;
  cout << ((n & mask) != 0 ? '1' : '0'); // 18
  mask >>= 1;
  cout << ((n & mask) != 0 ? '1' : '0'); // 17
  mask >>= 1;
  cout << ((n & mask) != 0 ? '1' : '0'); // 16
  mask >>= 1;
  cout << ((n & mask) != 0 ? '1' : '0'); // 15
  mask >>= 1;
```

```
  cout << ((n & mask) != 0 ? '1' : '0'); // 14
  mask >>= 1;
  cout << ((n & mask) != 0 ? '1' : '0'); // 13
  mask >>= 1;
  cout << ((n & mask) != 0 ? '1' : '0'); // 12
  mask >>= 1;
  cout << ((n & mask) != 0 ? '1' : '0'); // 11
  mask >>= 1;
  cout << ((n & mask) != 0 ? '1' : '0'); // 10
  mask >>= 1;
  cout << ((n & mask) != 0 ? '1' : '0'); // 9
  mask >>= 1;
  cout << ((n & mask) != 0 ? '1' : '0'); // 8
  mask >>= 1;
  cout << ((n & mask) != 0 ? '1' : '0'); // 7
  mask >>= 1;
  cout << ((n & mask) != 0 ? '1' : '0'); // 6
  mask >>= 1;
  cout << ((n & mask) != 0 ? '1' : '0'); // 5
  mask >>= 1;
  cout << ((n & mask) != 0 ? '1' : '0'); // 4
  mask >>= 1;
  cout << ((n & mask) != 0 ? '1' : '0'); // 3
  mask >>= 1;
  cout << ((n & mask) != 0 ? '1' : '0'); // 2
  mask >>= 1;
  cout << ((n & mask) != 0 ? '1' : '0'); // 1
  mask >>= 1;
  cout << ((n & mask) != 0 ? '1' : '0')
     << endl;                                  // 0

  return 0;
}
```

Восьмеричный дамп

```
// C++ Standard Operators
// Octal dump

#include <iostream>

int main()
{
  using namespace std;

  int n(13);
  int mask(07000000000);

  cout << char(((n >> 30) & 03) + 48);   // 10
  cout << char(((n & mask) >> 27) + 48); // 9
  mask >>= 3;
  cout << char(((n & mask) >> 24) + 48); // 8
  mask >>= 3;
  cout << char(((n & mask) >> 21) + 48); // 7
  mask >>= 3;
  cout << char(((n & mask) >> 18) + 48); // 6
  mask >>= 3;
  cout << char(((n & mask) >> 15) + 48); // 5
  mask >>= 3;
  cout << char(((n & mask) >> 12) + 48); // 4
  mask >>= 3;
  cout << char(((n & mask) >> 9) + 48);  // 3
  mask >>= 3;
  cout << char(((n & mask) >> 6) + 48);  // 2
  mask >>= 3;
  cout << char(((n & mask) >> 3) + 48);  // 1
  mask >>= 3;
  cout << char((n & mask) + 48)
       << endl;                          // 0

  cout << oct << '\n' << n << endl;

  return 0;
}
```

Шестнадцатеричный дамп

```
// C++ Standard Operators
// Hexadecimal dump

#include <iostream>

int main()
{
  using namespace std;

  int n(13);
  int mask(0xf000000);
  int digit((n >> 28) & 0xf);

  cout << (digit > 9 ? char(digit + 87)
                     : char(digit + 48));  // 7
  digit = (n & mask) >> 24;
  cout << (digit > 9 ? char(digit + 87)
                     : char(digit + 48));  // 6
  mask >>= 4;
  digit = (n & mask) >> 20;
  cout << (digit > 9 ? char(digit + 87)
                     : char(digit + 48));  // 5
  mask >>= 4;
  digit = (n & mask) >> 16;
  cout << (digit > 9 ? char(digit + 87)
                     : char(digit + 48));  // 4
  mask >>= 4;
  digit = (n & mask) >> 12;
  cout << (digit > 9 ? char(digit + 87)
                     : char(digit + 48));  // 3
  mask >>= 4;
  digit = (n & mask) >> 8;
  cout << (digit > 9 ? char(digit + 87)
                     : char(digit + 48));  // 2
  mask >>= 4;
  digit = (n & mask) >> 4;
  cout << (digit > 9 ? char(digit + 87)
                     : char(digit + 48));  // 1
  mask >>= 4;
  digit = n & mask;
  cout << (digit > 9 ? char(digit + 87)
                     : char(digit + 48))
       << endl;                            // 0

  cout << hex << '\n' << n << endl;

  return 0;
}
```

Циклические алгоритмы

Двоичный дамп

```cpp
// C++ Instructions
// Binary dump

#include <iostream>
#include <bitset>

int main()
{
  using namespace std;

  int n(13);

  cout << bitset<32>(n) << '\n' << endl;

  for (int i = 0; i < 32; ++i, n <<= 1)
    cout << (n < 0 ? '1' : '0');        // 31 - 0
  cout << endl;

  return 0;
}
```

Двоичный дамп

```
// C++ Instructions
// Binary dump

#include <iostream>
#include <bitset>

int main()
{
  using namespace std;

  int n(13);

  cout << bitset<32>(n) << '\n' << endl;

  int mask(010000000000);

  cout << (n < 0 ? '1' : '0');              // 31
  for (int i = 0; i < 31; ++i, mask >>= 1)
    cout << (n & mask ? '1' : '0');         // 30 - 0
  cout << endl;

  return 0;
}
```

Восьмеричный дамп

```
// C++ Instructions
// Octal dump

#include <iostream>

int main()
{
  using namespace std;

  int n(13);
  int mask(07000000000);

  cout << char(((n >> 30) & 03) + 48);              // 10
  for (int i = 0; i < 10; ++i, mask >>= 3)
    cout << char(((n & mask) >> 27 - i * 3) + 48); // 9 - 0
  cout << endl;

  cout << oct << '\n' << n << endl;

  return 0;
}
```

Шестнадцатеричный дамп

```
// C++ Instructions
// Hexadecimal dump

#include <iostream>

int main()
{
  using namespace std;

  int n(13);
  int mask(0xf000000);
  int digit((n >> 28) & 0xf);

  cout << (digit > 9 ? char(digit + 87)
                     : char(digit + 48));    // 7
  for (int i = 0; i < 7; ++i, mask >>= 4)
  {
    digit = (n & mask) >> 24 - i * 4;
    cout << (digit > 9 ? char(digit + 87)
                       : char(digit + 48));  // 6 - 0
  }
  cout << endl;

  cout << hex << '\n' << n << endl;

  return 0;
}
```

Функции

Двоичный дамп

```
// C++ Functions
// Binary dump

#include <iostream>
#include <bitset>

using namespace std;

void BinaryDump(int n)
{
  int mask(010000000000);

  cout << (n < 0 ? '1' : '0');              // 31
  for (int i = 0; i < 31; ++i, mask >>= 1)
    cout << (n & mask ? '1' : '0');         // 30 - 0
  cout << endl;
}

int main()
{
  int n;

  do
  {
    cout << "? ";
    cin >> n;
    if (cin.good())
      break;
    else
    {
      cout << "Invalid data!\n";
      cin.clear();
      while(cin.get() != '\n');
    }
  }
  while (1);

  BinaryDump(n);

  cout << '\n' << bitset<32>(n) << endl;

  return 0;
}
```

Восьмеричный дамп

```
// C++ Instructions
// Octal dump

#include <iostream>

using namespace std;

void OctalDump(int n)
{
  int mask(07000000000);

  cout << char(((n >> 30) & 03) + 48);              // 10
  for (int i = 0; i < 10; ++i, mask >>= 3)
    cout << char(((n & mask) >> 27 - i * 3) + 48); // 9 - 0
  cout << endl;
}

int main()
{
  int n;

  do
  {
    cout << "? ";
    cin >> n;
    if (cin.good())
      break;
    else
    {
      cout << "Invalid data!\n";
      cin.clear();
      while(cin.get() != '\n');
    }
  }
  while (1);

  OctalDump(n);

  cout << oct << '\n' << n << endl;

  return 0;
}
```

Шестнадцатеричный дамп

```
// C++ Instructions
// Hexadecimal dump

#include <iostream>

using namespace std;

void HexadecimalDump(int n)
{
  int mask(0xf000000);
  int digit((n >> 28) & 0xf);

  cout << (digit > 9 ? char(digit + 87)
                     : char(digit + 48));    // 7
  for (int i = 0; i < 7; ++i, mask >>= 4)
  {
    digit = (n & mask) >> 24 - i * 4;
    cout << (digit > 9 ? char(digit + 87)
                       : char(digit + 48));  // 6 - 0
  }
  cout << endl;
}

int main()
{
  int n;

  do
  {
    cout << "? ";
    cin >> n;
    if (cin.good())
      break;
    else
    {
      cout << "Invalid data!\n";
      cin.clear();
      while(cin.get() != '\n');
    }
  }
  while (1);

  HexadecimalDump(n);

  cout << hex << '\n' << n << endl;

  return 0;
}
```

Перегрузка функций

Двоичный, восьмеричный и шестнадцатеричный дампы

```
// C++ Overloaded functions
// Dump

#include <iostream>

using namespace std;

void BinaryDump(char n)
{
  char mask(0100);

  cout << (n < 0 ? '1' : '0');              // 7
  for (int i = 0; i < 7; ++i, mask >>= 1)
    cout << (n & mask ? '1' : '0');         // 6 - 0
  cout << endl;
}

void BinaryDump(short int n)
{
  short int mask(040000);

  cout << (n < 0 ? '1' : '0');              // 15
  for (int i = 0; i < 15; ++i, mask >>= 1)
    cout << (n & mask ? '1' : '0');         // 14 - 0
  cout << endl;
}

void BinaryDump(int n)
{
  int mask(010000000000);

  cout << (n < 0 ? '1' : '0');              // 31
  for (int i = 0; i < 31; ++i, mask >>= 1)
    cout << (n & mask ? '1' : '0');         // 30 - 0
  cout << endl;
}
```

```
void OctalDump(char n)
{
  int mask(070);

  cout << char(((n >> 6) & 03) + 48);                // 2
  for (int i = 0; i < 2; ++i, mask >>= 3)
    cout << char(((n & mask) >> 3 - i * 3) + 48);    // 1 - 0
  cout << endl;
}

void OctalDump(short int n)
{
  int mask(070000);

  cout << (n < 0 ? '1' : '0');           // 5
  for (int i = 0; i < 5; ++i, mask >>= 3)
    cout << char(((n & mask) >> 12 - i * 3) + 48); // 4 - 0
  cout << endl;
}

void OctalDump(int n)
{
  int mask(07000000000);

  cout << char(((n >> 30) & 03) + 48);               // 10
  for (int i = 0; i < 10; ++i, mask >>= 3)
    cout << char(((n & mask) >> 27 - i * 3) + 48); // 9 - 0
  cout << endl;
}
```

```
void HexadecimalDump(char n)
{
  int digit((n >> 4) & 0xf);

  cout << (digit > 9 ? char(digit + 87)
                     : char(digit + 48));    // 1
  digit = n & 0xf;
  cout << (digit > 9 ? char(digit + 87)
                     : char(digit + 48));    // 0
  cout << endl;
}

void HexadecimalDump(short int n)
{
  int mask(0xf00);
  int digit((n >> 12) & 0xf);

  cout << (digit > 9 ? char(digit + 87)
                     : char(digit + 48));    // 3
  for (int i = 0; i < 3; ++i, mask >>= 4)
  {
    digit = (n & mask) >> 8 - i * 4;
    cout << (digit > 9 ? char(digit + 87)
                       : char(digit + 48)); // 2 - 0
  }
  cout << endl;
}

void HexadecimalDump(int n)
{
  int mask(0xf000000);
  int digit((n >> 28) & 0xf);

  cout << (digit > 9 ? char(digit + 87)
                     : char(digit + 48));    // 7
  for (int i = 0; i < 7; ++i, mask >>= 4)
  {
    digit = (n & mask) >> 24 - i * 4;
    cout << (digit > 9 ? char(digit + 87)
                       : char(digit + 48)); // 6 - 0
  }
  cout << endl;
}
```

```
int main()
{
  int n;

  do
  {
    cout << "? ";
    cin >> n;
    if (cin.good())
      break;
    else
    {
      cout << "Invalid data!\n";
      cin.clear();
      while(cin.get() != '\n');
    }
  }
  while (1);

  BinaryDump(n);
  cout << endl;
  BinaryDump((short int)n);
  cout << endl;
  BinaryDump((char)n);
  cout << endl;

  OctalDump(n);
  cout << endl;
  OctalDump((short int)n);
  cout << endl;
  OctalDump((char)n);
  cout << endl;

  HexadecimalDump(n);
  cout << endl;
  HexadecimalDump((short int)n);
  cout << endl;
  HexadecimalDump((char)n);
  cout << endl;

  return 0;
}
```

Функции. Обобщённый указатель на тип данных

Двоичный, восьмеричный и шестнадцатеричный дампы

```
// C++ Functions
// Dump

#include <iostream>

using namespace std;

void BinaryDump(void* p, char type)
{
  int mask;

  switch (type)
  {
    case 'c' :  // char
    {
      char* p_cast = (char*)p;
      mask = 0100;
      cout << (*p_cast < 0 ? '1' : '0');       // 7
      for (int i = 0; i < 7; ++i, mask >>= 1)
        cout << (*p_cast & mask ? '1' : '0');  // 6 - 0
      cout << endl;
      break;
    }

    case 's' :  // short int
    {
      short int* p_cast = (short int*)p;
      mask = 040000;
      cout << (*p_cast < 0 ? '1' : '0');       // 15
      for (int i = 0; i < 15; ++i, mask >>= 1)
        cout << (*p_cast & mask ? '1' : '0');  // 14 - 0
      cout << endl;
      break;
    }

    case 'i' :  // int
    {
      int* p_cast = (int*)p;
      mask = 010000000000;
      cout << (*p_cast < 0 ? '1' : '0');       // 31
      for (int i = 0; i < 31; ++i, mask >>= 1)
        cout << (*p_cast & mask ? '1' : '0');  // 30 - 0
      cout << endl;
      break;
    }
  }
}
```

```
void OctalDump(void* p, char type)
{
  int mask;

  switch (type)
  {
    case 'c' :  // char
    {
      char* p_cast = (char*)p;
      mask = 070;
      cout << char(((*p_cast >> 6) & 03) + 48);              // 2
      for (int i = 0; i < 2; ++i, mask >>= 3)
        cout << char(((*p_cast & mask) >> 3 - i * 3) + 48); // 1 - 0
      cout << endl;
      break;
    }

    case 's' :  // short int
    {
      short int* p_cast = (short int*)p;
      mask = 070000;
      cout << (*p_cast < 0 ? '1' : '0');    // 5
      for (int i = 0; i < 5; ++i, mask >>= 3)
        cout << char(((*p_cast & mask) >> 12 - i * 3) + 48);// 4 - 0
      cout << endl;
      break;
    }

    case 'i' :  // int
    {
      int* p_cast = (int*)p;
      mask = 07000000000;
      cout << char(((*p_cast >> 30) & 03) + 48);              // 10
      for (int i = 0; i < 10; ++i, mask >>= 3)
        cout << char(((*p_cast & mask) >> 27 - i * 3) + 48);// 9 - 0
      cout << endl;
      break;
    }
  }
}
```

```
void HexadecimalDump(void* p, char type)
{
  int mask;

  switch (type)
  {
    case 'c' :  // char
    {
      char* p_cast = (char*)p;
      int digit((*p_cast >> 4) & 0xf);
      cout << (digit > 9 ? char(digit + 87)
                         : char(digit + 48));  // 1
      digit = *p_cast & 0xf;
      cout << (digit > 9 ? char(digit + 87)
                         : char(digit + 48));  // 0
      cout << endl;
      break;
    }

    case 's' :  // short int
    {
      short int* p_cast = (short int*)p;
      mask = 0xf00;
      int digit((*p_cast >> 12) & 0xf);
      cout << (digit > 9 ? char(digit + 87)
                         : char(digit + 48));    // 3
      for (int i = 0; i < 3; ++i, mask >>= 4)
      {
        digit = (*p_cast & mask) >> 8 - i * 4;
        cout << (digit > 9 ? char(digit + 87)
                           : char(digit + 48));  // 2 - 0
      }
      cout << endl;
      break;
    }

    case 'i' :  // int
    {
      int* p_cast = (int*)p;
      mask = 0xf000000;
      int digit((*p_cast >> 28) & 0xf);
      cout << (digit > 9 ? char(digit + 87)
                         : char(digit + 48));    // 7
      for (int i = 0; i < 7; ++i, mask >>= 4)
      {
        digit = (*p_cast & mask) >> 24 - i * 4;
        cout << (digit > 9 ? char(digit + 87)
                           : char(digit + 48));  // 6 - 0
      }
      cout << endl;
      break;
```

```
    }
  }
}
```

```
int main()
{
  int n;
  int* p = &n;

  do
  {
    cout << "? ";
    cin >> n;
    if (cin.good())
      break;
    else
    {
      cout << "Invalid data!\n";
      cin.clear();
      while(cin.get() != '\n');
    }
  }
  while (1);

  BinaryDump((char*)p, 'c');
  BinaryDump((short int*)p, 's');
  BinaryDump((int*)p, 'i');
  cout << endl;

  OctalDump((char*)p, 'c');
  OctalDump((short int*)p, 's');
  OctalDump((int*)p, 'i');
  cout << endl;

  HexadecimalDump((char*)p, 'c');
  HexadecimalDump((short int*)p, 's');
  HexadecimalDump((int*)p, 'i');

  return 0;
}
```

Шаблоны функций

Двоичный, восьмеричный и шестнадцатеричный дампы

```
// C++ Function templates
// Dump

#include <iostream>

using namespace std;

template <typename T>
void BinaryDump(T n)
{
  T mask;
  int size(8 * sizeof(T) - 1);

  switch (sizeof(T))
  {
    case 1 :  // char
      mask = 0100;
      break;

    case 2 :  // short int
      mask = 040000;
      break;

    case 4 :  // int or long int
      mask = 010000000000;
      break;
  }
  cout << (n < 0 ? '1' : '0');                  // 7 or 15 or 31
  for (int i = 0; i < size; ++i, mask >>= 1)
    // 6 - 0 or 14 - 0 or 30 - 0
    cout << (n & mask ? '1' : '0');
  cout << endl;
}
```

```
template <typename T>
void OctalDump(T n)
{
  T mask;
  int size((8 * sizeof(T)) / 3);
  int base(3 * ((8 * sizeof(T) / 3) - 1));

  switch (sizeof(T))
  {
    case 1 :  // char
      mask = 070;
      cout << char(((n >> 6) & 03) + 48);  // 2
      break;

    case 2 :  // short int
      mask = 070000;
      cout << (n < 0 ? '1' : '0');         // 5
      break;

    case 4 :  // int or long int
      mask = 07000000000;
      cout << char(((n >> 30) & 03) + 48); // 10
      break;
  }
  for (int i = 0; i < size; ++i, mask >>= 3)
    // 1 - 0 or 4 - 0 or 9 - 0
    cout << char(((n & mask) >> base - i * 3) + 48);
  cout << endl;
}
```

```
template <typename T>
void HexadecimalDump(T n)
{
  T mask;
  int size(2 * sizeof(T) - 1);
  int base(8 * (sizeof(T) - 1));
  int digit((n >> (8 * sizeof(T) - 4)) & 0xf);

  switch (sizeof(T))
  {
    case 1 :  // char
      mask = 0xf;
      break;

    case 2 :  // short int
      mask = 0xf00;
      break;

    case 4 :  // int or long int
      mask = 0xf000000;
      break;
  }
  // 1 or 3 or 7
  cout << (digit > 9 ? char(digit + 87)
                     : char(digit + 48));
  for (int i = 0; i < size; ++i, mask >>= 4)
  {
    // 0 or 2 - 0 or 6 - 0
    digit = (n & mask) >> base - i * 4;
    cout << (digit > 9 ? char(digit + 87)
                       : char(digit + 48));
  }
  cout << endl;
}
```

```
int main()
{
  int n;

  do
  {
    cout << "? ";
    cin >> n;
    if (cin.good())
      break;
    else
    {
      cout << "Invalid data!\n";
      cin.clear();
      while(cin.get() != '\n');
    }
  }
  while (1);

  BinaryDump(n);
  cout << endl;
  BinaryDump((short int)n);
  cout << endl;
  BinaryDump((char)n);
  cout << endl;

  OctalDump(n);
  cout << endl;
  OctalDump((short int)n);
  cout << endl;
  OctalDump((char)n);
  cout << endl;

  HexadecimalDump(n);
  cout << endl;
  HexadecimalDump((short int)n);
  cout << endl;
  HexadecimalDump((char)n);
  cout << endl;

  BinaryDump<int>(n);
  cout << endl;
  BinaryDump<short int>(n);
  cout << endl;
  BinaryDump<char>(n);
  cout << endl;

  OctalDump<int>(n);
  cout << endl;
  OctalDump<short int>(n);
  cout << endl;
```

```
  OctalDump<char>(n);
  cout << endl;

  HexadecimalDump<int>(n);
  cout << endl;
  HexadecimalDump<short int>(n);
  cout << endl;
  HexadecimalDump<char>(n);
  cout << endl;

  return 0;
}
```