Teoria Informacji i Kodowania

Implementacja funkcji CRC

Arkadiusz Ostrzyżek WCY22KY2S1

Treść zadania

Napisz program dopisujący do pliku CRC32 oraz sprawdzający integralność danych w pliku z dopisanym CRC.

Analiza zadania

Kod CRC32 pozwala na sprawdzenie, czy w pliku zaszły zmiany od czasu ostatniego zapisu. Kod ten, jest wyliczany na podstawie zawartości pliku, a następnie umieszczany na jego końcu w postaci binarnej. Takie sprawdzenie jest przydatne na przykład w trakcie przesyłu danych przez internet, gdzie przerwania i zakłócenia mogłyby spowodować modyfikacje zawartości.

Założenia

Zakładam, że program wykona operację tylko raz w celach demonstracyjnych, a więc nie konieczne jest implementowanie obliczania całej tablicy CRC, jednak program wykorzystuje tą samą technikę co do obliczania takowej.

Algorytm

Opis działania

Funkcja crc32 przyjmuje dwa argumenty: wskaźnik na ciąg znaków s i długość tego ciągu length.
Do obliczeń używamy wielomianu 04C11DB7, jednak w kodzie zapisana jest jego odwrotność, ze względu na używaną negację.

Zmienna crc jest inicjalizowana wartością 0xFFFFFFFF. Następnie, dla każdego bajtu w ciągu znaków, wykonuje następujące operacje:
- Wykonuje operację XOR na bieżącej wartości crc i bajcie. - Wykonuje 8 iteracji, w

których przesuwa bity crc o jeden bit w prawo, a następnie wykonuje operację XOR z wartością 0xEDB88320 i wynikiem operacji AND na - (crc & 1).

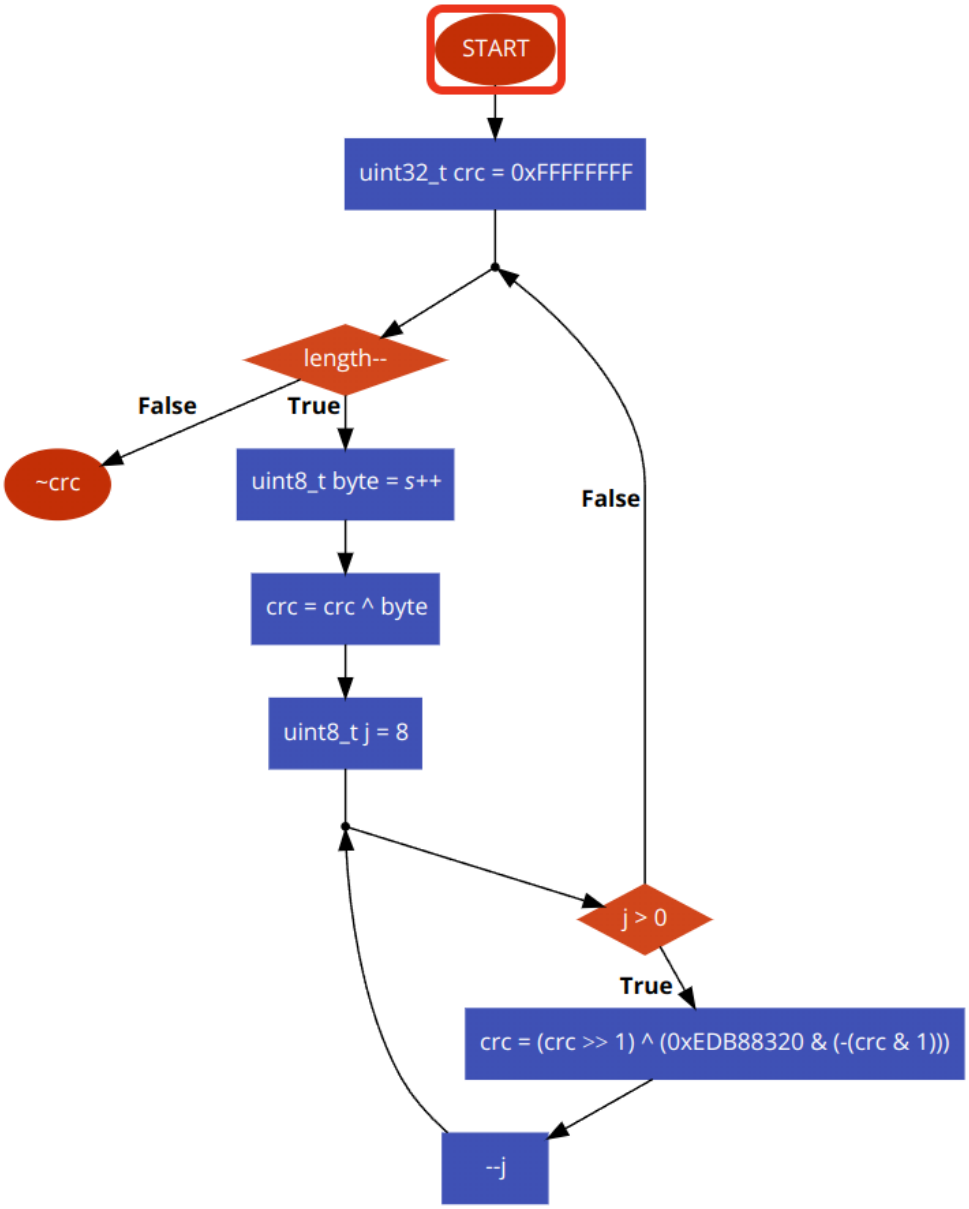
-(crc & 1) zwraca zawsze wynik -1 albo 0. Dla wartości 0, program przesunie całość o jedno miejsce, dla wartości -1, zostanie wykonany AND z wybranym przez nas wielomianem, co oznacza iż wielomian pozostanie w niezmienionej przez nas formie. Następnie zostanie wykonana operacja XOR pomiędzy przesuniętym o 1 w prawo crc.

Po przetworzeniu wszystkich bajtów, funkcja zwraca zanegowaną wartość crc.

Kod

```
1  uint32_t crc32(const char* s, size_t length)
2  {
3      uint32_t crc = 0xFFFFFFFF;
4      while (length--)
5      {
6          uint8_t byte = *s++;
7          crc = crc ^ byte;
8          for (uint8_t j = 8; j > 0; --j)
9          {
10             crc = (crc >> 1) ^ (0xEDB88320 & (-(crc & 1)));
11         }
12     }
13     return ~crc;
14 }
```

Flowchart


START
uint32_t crc = 0xFFFFFFFF
length--
False
True
~crc
uint8_t byte = s++
False
crc = crc ^ byte
uint8_t j = 8
j > 0
True
crc = (crc >> 1) ^ (0xEDB88320 & (-(crc & 1)))
--j

Testowanie

Dodawanie CRC

Początkowy plik:

```
1       Plik testowy
2       Implementacja crc Na TIK
3       CRC:
```

```
   00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00 50 6C 69 6B 20 74 65 73 74 6F 77 79 0A 49 6D 70 Plik testowy Imp
10 6C 65 6D 65 6E 74 61 63 6A 61 20 63 72 63 20 4E lementacja crc N
20 61 20 54 49 4B 0A 43 52 43 3A                   a TIK CRC:
```

Plik po dodaniu CRC:

```
1       Plik testowy
2       Implementacja crc Na TIK
3       CRC:????
```

```
   00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00 50 6C 69 6B 20 74 65 73 74 6F 77 79 0A 49 6D 70 Plik testowy Imp
10 6C 65 6D 65 6E 74 61 63 6A 61 20 63 72 63 20 4E lementacja crc N
20 61 20 54 49 4B 0A 43 52 43 3A AC A0 F3 F0       a TIK CRC:????
```

```
CRC32: f0f3a0ac
```

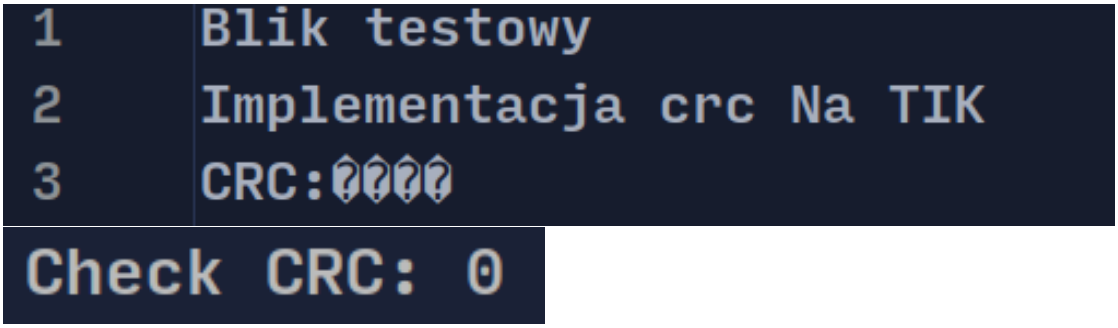
Wartość zapisana w pliku wydaje się być początkowo błędna, jednak wartość F0F3A0AC jest zapisana używając little-endian, a wartość ACA0F3F0 jest zapisana używając big endian.

Sprawdzanie CRC

Dla niezmienionego pliku wynik sprawdzenia wynosi 1.

```
Check CRC: 1
```

Następnie zmodyfikujemy plik, zmieniając dowolną z liter.



Plik został zmodyfikowany, a więc wartość CRC jest niezgodna z resztą zawartości pliku i wynik wynosi 0.

Wnioski

Kodowanie CRC można zaimplementować w języku C++. Implementacja wymaga opanowania operacji na bitach, w celu przyśpieszenia obliczeń. CRC może być używane jako mechanizm potwierdzający integralność pliku.

Kod źródłowy programu

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <cstring>

using namespace std;

string readFileIntoString(const string& filePath) {
    ifstream fileStream(filePath, ios::binary);
    stringstream stringStream;
    stringStream << fileStream.rdbuf();
    return stringStream.str();
}

void appendToFile(const string& filePath, uint32_t data) {
    ofstream fileStream(filePath, ios::binary | ios::app);
    fileStream.write(reinterpret_cast<const char *>(&data),
        sizeof(data));
    fileStream.close();
}
```

```
20
21 uint32_t crc32(const char* s, size_t length) {
22     uint32_t crc = 0xFFFFFFFF;
23     while (length--)
24     {
25         uint8_t byte = *s++;
26         crc = crc ^ byte;
27         for (uint8_t j = 8; j > 0; --j)
28         {
29             crc = (crc >> 1) ^ (0xEDB88320 & (-(crc & 1)));
30         }
31     }
32     return ~crc;
33 }
34
35 bool checkCRC(const string& filePath) {
36
37     string fileContent = readFileIntoString(filePath);
38     size_t fileSize = fileContent.size();
39
40     if (fileSize < sizeof(uint32_t)) {
41         cerr << "File size is too small: " << filePath << endl;
42         return false;
43     }
44
45     string contentExceptCRC = fileContent.substr(0, fileSize -
        sizeof(uint32_t));
46     uint32_t calculatedCRC = crc32(contentExceptCRC.c_str(),
        contentExceptCRC.size());
47
48     uint32_t fileCRC;
49     memcpy(&fileCRC, &fileContent[fileSize - sizeof(uint32_t)],
         sizeof(fileCRC));
50
51     return calculatedCRC == fileCRC;
52 }
53
54 int main() {
55     string input = readFileIntoString("input.txt");
56     uint32_t crc = crc32(input.c_str(), input.length());
57     cout << "CRC32: " << hex << crc << endl;
58     appendToFile("input.txt", crc);
59     bool check = checkCRC("input.txt");
60     cout << "Check CRC: " << check << endl;
```

```
61         return 0;
62 }
```