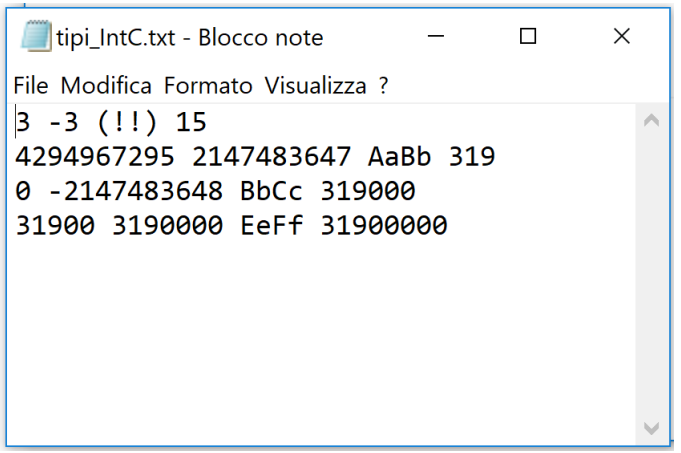
```
#include <iostream>
using namespace std;
#include <fstream>
#define SP ' '
struct Riga {                  //--- 4 campi da 4 byte = 16 byte ogni record
 unsigned int       unsint;
 int                intero;
 char               lett[4];
 int                altro;
};
  Riga rec;                    //--- dichiarazione di un record rec di tipo Riga

int main(){
  fstream ftxt, fdat;
      ftxt.open("tipi_IntC.txt", ios::out);
      fdat.open("tipi_IntC.dat", ios::out | ios::binary);

    rec.unsint  = 3;
    rec.intero  =-3;
    rec.lett[0] = 0x28; //--- codifica ASCII di '(' espressa in esadecimale
    rec.lett[1] = 33;   //--- codifica ASCII di '!' espresso in base 10
    rec.lett[2] = 0x21; //--- codifica ASCII di '!' espresso in esadecimale
    rec.lett[3] = 41;   //--- codifica ASCII di ')' espresso in base 10
    rec.altro   = 15;

    ftxt<< rec.unsint << SP << rec.intero << SP;
    ftxt<< rec.lett[0]<< rec.lett[1]<< rec.lett[2]<< rec.lett[3];
    ftxt<< SP << rec.altro << endl;
    fdat.write( (char *) &rec, sizeof(rec) );
//-----------------------------------------------

    rec.unsint  = 4294967295;    //--- oltre 4 miliardi
    rec.intero  = 2147483647;    //--- oltre 2 miliardi
    rec.lett[0] = 'A';           //--- carattere delimitato da apici
    rec.lett[1] = 'a';
    rec.lett[2] = 'B';
    rec.lett[3] = 'b';
    rec.altro   = 319;

    ftxt<< rec.unsint << SP << rec.intero << SP;
    ftxt<< rec.lett[0]<< rec.lett[1]<< rec.lett[2]<< rec.lett[3];
    ftxt<< SP << rec.altro << endl;
    fdat.write( (char *) &rec, sizeof(rec) );
```

Il programma ***tipi_IntC.cpp*** scrive sul **file di testo** **"tipi_IntC.txt"** (nome logico **ftxt**) 4 righe (separandole con **endl**) in ognuna delle quali scrive 7 dati: un intero senza segno, un intero, 4 caratteri e, infine, un altro intero. I 4 caratteri sono scritti consecutivamente (senza separarli con uno spazio) mentre gli altri dati sono seguiti da uno spazio (costante SP) e l'ultimo intero da un A CAPO (*endl*). La scrittura sul file di testo si realizza con **l'operatore <<**

Sulla prima riga scrive i valori 3 e -3 , due caratteri il cui codice ASCII è espresso in decimale e due in esadecimale.

Sulla seconda riga vengono scritti i **valori massimi per il tipo** `int` e `unsigned int` (poco più di 4 miliardi per l' `unsigned int`, la metà, circa 2 miliardi per l' `int`), mentre i char vengono valorizzati con il carattere delimitato da apici.



I valori scritti sulla terza riga sono ottenuti dai precedenti mediante incremento unitario (operatore ++); quindi, si ottengono i **valori minimi per il tipo** `unsigned int` e `int` (rispettivamente 0 per l' `unsigned int`, circa **-2** miliardi per l' `int`); sul tipo char, l'incremento

```
    rec.unsint++;
    rec.intero++;
    for (int k=0; k<4; k++)
        rec.lett[k]++;
    rec.altro *= 1000;

    ftxt<< rec.unsint << SP << rec.intero << SP;
    ftxt<< rec.lett[0]<< rec.lett[1]<< rec.lett[2]<< rec.lett[3];
    ftxt<< SP << rec.altro << endl;
    fdat.write( (char *) &rec, sizeof(rec) );

//-----------------------------------------------

    rec.unsint = 31900;
    rec.intero = 3190000;
    for (int k=0; k<4; k++)
        rec.lett[k] += 3;
    rec.altro  *= 100;

    ftxt<< rec.unsint << SP << rec.intero << SP;
    ftxt<< rec.lett[0]<< rec.lett[1]<< rec.lett[2]<< rec.lett[3];
    ftxt<< SP << rec.altro << endl;
    fdat.write( (char *) &rec, sizeof(rec) );


      ftxt.close();
      fdat.close();
      return 0;
}
```

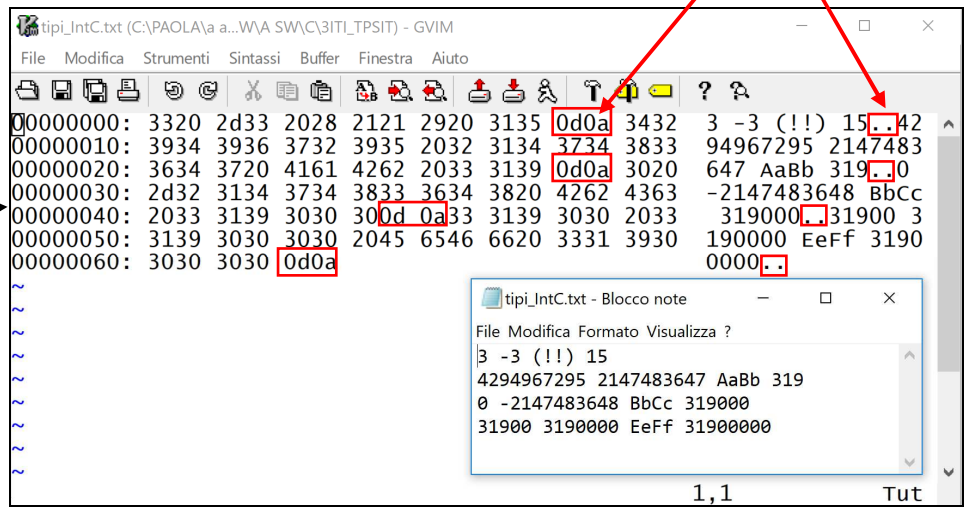
Nella prima colonna i valori esadecimali del progressivo del primo byte di riga a partire da 0; poichè su ogni riga ***Vim*** espone 16 caratteri contenuti nel file **"tipi_IntC.txt"**, sulla prima riga è indicato il byte 0, sulla seconda riga c'è 00000010 che rappresenta il byte 16, etc. etc.

unitario si applica al codice ASCII: il carattere dopo la A è B, etc; l'ultimo intero è ottenuto dal valore precedente moltiplicato 1000.

Sulla quarta riga, multipli di 319, i codici ASCII dei char vengono incrementati di 3, l'ultimo intero è ottenuto dal valore precedente moltiplicato 100.

Aprendo il file con **Blocco Note** tutti i valori sono in chiaro e leggibili perchè **espressi in caratteri** (anche **i numeri sono esposti** in chiaro **mediante elenco delle cifre che li compongono**).

Aprendo il file di testo con l'editor ***Vim*** e selezionando Strumenti > Converti a esadecimale, sulla destra si vedono i singoli caratteri; al centro per ogni carattere si vede il codice ASCII espresso in esadecimale: il carattere 3 corrisponde al codice ASCII 33_{hex}, lo spazio è codificato con 20_{hex}, il meno (-) con $2D_{hex}$, il punto esclamativo (!) con 21_{hex}, *endl* è rappresentato da 2 caratteri $0D_{hex}$ e $0A_{hex}$ (.. a destra)



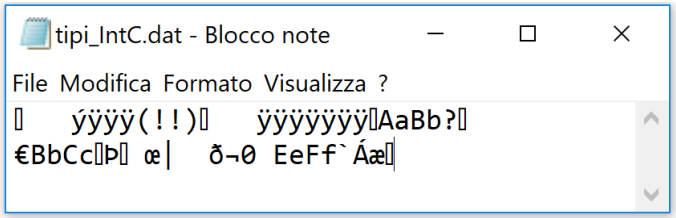
A)

Il **file binary** viene invece scritto utilizzando la funzione (metodo) **write** che prevede in input una struttura record di lunghezza fissa (`rec` di tipo `Riga` , struttura record definita nel programma); in questo caso il record occupa 16 byte = 4(`unsint`)+ 4(`intero`)+ 4(array `lett[4]`)+ 4(`altro`).

Aprendo il file binary con **Blocco Note** vengono esposti caratteri non stampabili, ad eccezione dei caratteri corrispondenti all'array `lett[4]` :

(!!) per il primo record **AaBb** per il secondo record

BbCc per il terzo record **EeFf** per il quarto



C) Nel secondo record, il campo `intero` contiene **FFFF FF7F** cioè il numero **7FFFFFFF**$_{hex}$ = **01111111111111111111111111111111**$_2$ cioè il numero **positivo** più grande rappresentabile in 32 bit = **2.147.483.647**$_{10}$

Il campo `altro` contiene 319_{10} = **13F**$_{hex}$ che viene quindi memorizzato così: **3F01 0000**

Nel terzo record il campo `altro` contiene 319000_{10} = **4DE18**$_{hex}$ che viene memorizzato così: **18DE 0400**

Il campo `intero` contiene **0000 0080** cioè il numero **80000000**$_{hex}$ cioè il numero **10000000000000000000000000000000**$_2$ cioè il numero negativo più piccolo rappresentabile in 32 bit = **-2.147.483.648**$_{10}$

B) NOTA: i **processori Intel** memorizzano i dati numerici multi-byte con il sistema ***little endian***, cioè memorizzano prima il byte meno significativo (a sinistra) per finire con quello più significativo (a destra).

Aprendo il file **binary** con l'editor ***Vim*** su ogni riga trova spazio un intero record (16 byte); sulla destra si vedono i singoli caratteri, non stampabili, quindi espressi generalmente come una serie di punti (.); al centro ogni coppia di caratteri rappresenta 1 byte, quindi 0300 0000 è il contenuto dei primi 4 byte (occupati da `unsint`) espresso in esadecimale.

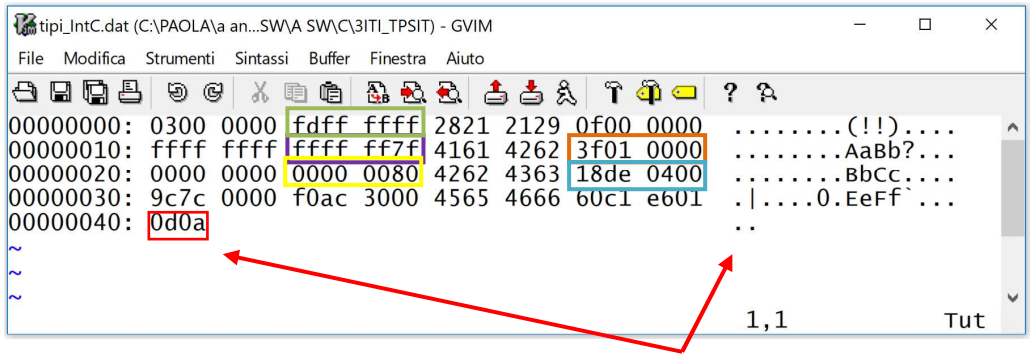
Quindi **0300 0000** è il numero **00000003**$_{hex}$ cioè il numero 3_{10}.

I successivi 4 byte **FDFF FFFF** contengono il numero **FFFFFFFD**$_{hex}$ cioè il numero -3, infatti il numero è **negativo** (primo byte = **FF** cioè inizia con 1, quindi è la codifca di un numero negativo).

Considerando solo il byte meno significativo **FD**$_{hex}$ = 11111101_2 questa è la rappresentazione (su 8 bit) dell'opposto di 3 = 00000011_2 (-3 : **FD**$_{hex}$).

Il campo `altro` contiene **0F00 0000** cioè il numero **0000000F**$_{hex}$

considerando solo il byte meno significativo **0F**$_{hex}$ = 00001111_2 = 15_{10}



NOTA: nel **file binary** solo a fine file è presente 0D0A$_{hex}$ = CRLF = newline ovvero i record sono consecutivi, non separati da CRLF