【出力結果】
test-data0123456789
789
1953719668

【説明】
メモリの0番地からchar配列t2が宣言されるとする。
各番地には1バイトが割り当てられていること、char型は1バイトであること、プロセッサはlittle endianであることから、

0番地 -> t2[15]
1番地 -> t2[14]
…
15番地 -> t2[0]
と割り当てられる。

次に宣言される配列t1も同様に、
16番地 -> t1[15]
17番地 -> t1[14]
…
31番地 -> t1[0]
と割り当てられる。

次に宣言される配列sは19文字と終端記号で20バイト分であり、
32番地 -> s[19]='\0'
33番地 -> s[18]='9'
…
51番地 -> s[0]='t'
と割り当てられる。

ここで構文doの箇所について考える。
配列sの終端記号までの20回繰り返される。
最初の15回は,t1[0]からt1[14]まで配列sの文字が正しく代入される。
16回目はt1[15]の終端記号に、s[15]が代入されてしまう。
17~20回目については特に、
t1[16]は、t2[0]を指すことになり、s[16]=7の値で上書き。
t1[17]は、t2[1]を指すことになり、s[17]=8の値で上書き。
t1[18]は、t2[2]を指すことになり、s[18]=9の値で上書き。
t1[19]は、t2[3]を指すことになり、s[19]='\0'の値で上書き。

従って、
puts(t1)はt1領域を超えてt2領域にある終端記号までのsの値全てを表示した。
puts(t2)はdo構文で上書きされたs[16]から終端記号までの”789”を表示した。

最後のprintf出力について。
char型配列t1、つまりchar型ポインタt1を無理やりint型ポインタにキャストしている。
int型は4バイトなので、t1[0]~t1[3]の”test”が読まれている。
それぞれの文字を16進数で出力してみると、
’t’ -> 74
‘e’ -> 65
’s’ -> 73
となった。
従って、
１６進数で74736574、これを10進数に変換すると1953719668となり
これがint型の値として表示されたと思われるs。