Tutorial 1: C-Language

Problem 1: Data Type

What are the ranges of the following data types?

- `int` 32 bits –2^{31}..2^{31}-1 OR -2147483648..2147483647 (0..4294967295 if unsigned)
- `long` same as `int`
- `long long` 64 bits –2^{63}..2^{63}-1 OR -9223372036854775808..9223372036854775807 (0..18446744073709551615 if unsigned)
- `short` 16 bits –2^{15}..2^{15}-1 OR -32768..32767 (0..65535 if unsigned)
- `char` 8 bits –2^{7}..2^{7}-1 OR -128..127 (0..255 if unsigned)

Problem 2: Data Type Conversion

Consider the C code in Figure 1. Answer the following questions.

What are the meaning of statements `(b = (short) a;)` and `(c = (char) a;)`?

What are the outputs of the `printf` statements?

What are the outputs of the `printf` statements if `(a = -232590606)`?

```c
#include <stdio.h>
int main (void)
{
    int a=-232644062;
    short b;
    char c;

    b = (short) a;
    c = (char) a;
    printf("integer = \"%d\"\n\n", a);
    printf("short = \"%d\"\n\n", b);
    printf("char = \"%d\"\n\n", c);
    return 0;
}
```

Figure 1: Program on Data Type Conversion
Problem 3: More Data Type Conversion

Consider the C code in Figure 4. Answer the following questions.
What are the meaning of statements (a = (int) b;) and (c = (char) b;)?
What are the outputs of the printf statements?
What are the outputs of the printf statements if (b = -30848)?

```c
#include <stdio.h>

int main (void)
{
    short b = -32768;
    int a;
    char c;

    a = (int) b;
    c = (char) b;
    printf("short = \"%d\"\n\n", b);
    printf("integer = \"%d\"\n\n", a);
    printf("char = \"%d\"\n\n", c);
    return 0;
}
```

Figure 4: Program on Data Type Conversion
So, the output of the `printf` statements are.

`short = -32768`
`integer = -32768`
`char = 0`

Figure 5: The Outputs of `printf` Statements

For `(b = -30848)` the hex representation is `(a = 0x8780)`. Two statements `(a = (int) b;)` and `(c = (char) b;)` have the following effects:

- `(a = (int) b;)` : `(b = 0x8780)` in 16 bits Æ `(a = 0xffff8780)` in 32 bits
- `(c = (char) b;)` : `(b = 0x8780)` in 16 bits Æ `(c = 0x80)` in 8 bits

Converting back from the hex format to decimal give `(a = -30848)` and `(c = -128)`.

So, the output of the `printf` statements are.

`short = -30848`
`integer = -30848`
`char = -128`

Figure 6: The Outputs of `printf` Statements

Problem 4: Some More Data Type Conversion
Consider the C code in Figure 7. Answer the following questions.

What are the meaning of statements `(b = (int)(short) a;)` and `(c = (int)(char) a;)`?
What are the outputs of the `printf` statements?
What are the outputs of the `printf` statements if `(a = -232590606)`?

```c
#include <stdio.h>

int main (void)
{
    int a=-232644062;
    int b;
    int c;
    b = (int)(short) a;
    c = (int)(char) a;
    printf("integer = \"%d\n\n\n\",  a);
    printf("int_short = \"%d\n\n\n\", b);
    printf("int_char = \"%d\n\n\n\", c);
    return 0;
}
```

Figure 7: Program on Data Type Conversion

Two type casts statements `(b = (int)(short) a;)` and `(c = (int)(char) a;)` convert `(a)` to types `(short)` and `(char)` and back to `(int)`.

Again representing `(a = -232644062)` as `(a = 0xf2222222`). Two statements `(b = (int)(short) a;)` and `(c = (int)(char) a;)` have the following effects:

- `(b = (int)(short) a;)` : `(a = 0xf2222222)` in 32 bits Æ `(0x2222)` in 16 bits Æ `(0x2222)` in 16 bits Æ `(b = 0x00002222)` in 32 bits
Problem 5: Still Some More Data Type Conversion

Consider the C code in Figure 10. Answer the following questions.

What are the outputs of the printf statements?

What is the output of the last printf statement, if (+ 1) is removed from the statement (d = (short *) &a + 1) ?

```c
#include <stdio.h>

int main (void)
{
    int a=-232644062;
    char *b, *c;
    short *d;

    b =(char *) &a;
    c =(char *) &a + 1;
    d =(short *) &a + 1;

    printf("char1 = \"\%d\"\n\n", *b);
    printf("char2 = \"\%d\"\n\n", *c);
    printf("short = \"\%d\"\n\n", *d);

    return 0;
}
```

Figure 10: Program on Data Type Conversion with Pointers
For \( a = -232644062 \) the hex representation is \( a = 0xf2222222 \). We can represent ‘\( a \)’ as concatenation of 4 bytes as: \( a = 0xf2222222 \). The statement \( b = (\text{char} *) \&a; \) obtains the address of the \text{int} \ type variable ‘\( a \)’ and converts it to the address of the \text{char} \ type through type cast \( (\text{char} *) \) and assigns it to pointer \( c \) of type \text{char}. So what gets printed out by the second \text{printf} \ statement the first (right most) byte of ‘\( a \)’ which is \( 0x22 = 34 \). The \( c = (\text{char} *) \&a + 1; \) will access the second (from right) byte of ‘\( a \)’ which is again \( 0x22 = 34 \). The \( d = (\text{short} *) \&a + 1; \) obtains the address of the integer type variable ‘\( a \)’ and converts it to the address of the \text{short} \( (16\text{-bit}) \) type through type cast \( (\text{short} *) \) and adds 1 to it, and assigns it to pointer \( d \) of type \text{short}. So what gets printed out by the last \text{printf} \ statement the second (left) half of ‘\( a \)’ which is \( 0xf222 = -3550 \). So, the output of the \text{printf} \ statements are.

\[
\begin{align*}
\text{char1} &= \text{"34"} \\
\text{char2} &= \text{"34"} \\
\text{short} &= \text{"-3550"}
\end{align*}
\]