錬成問題

▪ 以下に示すのは、標準C++で保証されている、整数型の各型で表現可能な値の範囲とビット数である。

型	表現可能な値の範囲			ビット数
short int	(1)	～	(2)	(3)
int	(4)	～	(5)	(6)
long int	-2147483647	～	2147483647	32

▪ 整数リテラルには3種類の基数表記がある。基数の小さいほうから順に、(7)進リテラル、(8)進リテラル、(9)進リテラルである。

`0`は(10)進リテラル、`01`は(11)進リテラル、`10`は(12)進リテラル、`010`は(13)進リテラル、`0x1`は(14)進整数リテラルである。

なお、整数リテラルの型は、値や接尾語などによって異なる。**long**型とするためには(15)または(16)の整数接尾語を末尾に付けて、符号無し型とするためには(17)または(18)の整数接尾語を末尾に付ける。

▪ **char**型は少なくとも(19)ビットで構成される。**sizeof**(**char**)の値は(20)であり、その型は(21)である。(21)型は、<(22)>ヘッダ内で、(23)宣言によって定義されている。

単なる文字型は、符号付き文字型・符号無し文字型のいずれ(24)である。

▶ (24)の選択肢：(a)かと同一の型　(b)とも異なる独立した型

▪ 浮動小数点リテラルは基本的には**double**型である。**float**型とするためには(25)または(26)の浮動小数点接尾語を末尾に付け、**long double**型とするためには(27)または(28)の浮動小数点接尾語を末尾に付ける。

▪ 値を表現するための記憶域のことを(29)と呼ぶ。

▪ 以下に示すプログラムの実行結果を示せ。

```
#define NO  8
int NO1 = 1;
cout << "NO  = " << NO << '\n';
cout << "NO1 = " << NO1 << '\n';
```

(30)

▪ 以下に示すのは、文字`c`が数字文字であるかどうかを表示するプログラム部分である。

```
if (ch >= (31) && ch <= (32))
    cout << "chは数字文字\n";
else
    cout << "chは非数字文字\n";
```

```
#include <(33)>
if ((34)(ch))
    cout << "chは数字文字\n";
else
    cout << "chは非数字文字\n";
```

▪ 整数型と浮動小数点型の総称は、 (35) 型である。

▪ 以下に示す各プログラムの実行結果を示せ。

```
cout << 015 << '-' << 15 << '-' << 0x15 << '\n';
for (int i = 0; i < 10; i++)
    cout << char('0' + i);
```

(36)

```
cout << 15   / 2   << '\n';
cout << 15.0 / 2   << '\n';
cout << 15   / 2.0 << '\n';
cout << 15.0 / 2.0 << '\n';
```

(37)

```
cout << (double)15 / 2   << '\n';
cout << double(15) / 2   << '\n';
cout << static_cast<double>(15) / 2.0 << '\n';
```

(38)

```
cout << (true == true)  << '\n';
cout << (true != false) << '\n';
cout << (true && true)  << '\n';
cout << (true || false) << '\n';
```

(39)

```
cout << boolalpha;
cout << ('0' == 0)  << '\n';
cout << ('5' - '0') << '\n';
cout << (true == true)  << '\n';
cout << (true == false) << '\n';
cout << (sizeof(int) == sizeof(5)) << '\n';
cout << (sizeof(long) == sizeof(5L)) << '\n';
cout << (sizeof(double) == sizeof(5.0)) << '\n';
```

(40)

```
(41) animal {Dog, Cat = 2, Monkey};
cout << "犬 = " << Dog << '\n';
cout << "猫 = " << Cat << '\n';
cout << "猿 = " << Monkey << '\n';
```

犬 =
猫 =
猿 =
(42)

なお、*animal* のような型は (43) 型と呼ばれ、*Dog*, *Cat*, *Monkey* は (44) と呼ばれる。

▪ 以下に示すのは、0 ～ 255 までの整数を 1 行ずつ表示するプログラムである。なお、表示は、5 桁の幅の 8 進数、10 進数、16 進数表記を空白で区切って並べたものである。

```
for (int i = 0; i < 256; i++) {
    cout << (45) << (46) << i << ' '
         << (45) << (47) << i << ' '
         << (45) << (48) << i << '\n';
}
```

▪ 以下に示すのは、**double** 型変数 *x* の値を、全体を少なくとも 10 桁、小数点以下を 5 桁で表示するプログラムである（実行例に示すのは *x* が 3.14159265 の出力）。

```
cout << (49) << (50)(10) << (51)(5) << x << '\n';
```

□□□3.14159

▪ 8進リテラルの先頭は (52) で始まり、16進リテラルの先頭は (53) または (54) で始まる。

▪ 以下に示すリテラルの型を示せ。

`'A'` … (55)　　`5` … (56)　　`5L` … (57)　　`5U` … (58)

`5UL` … (59)　　`5F` … (60)　　`5.L` … (61)

▪ **char** 型で表現できる最小値を表す (62)、最大値を表す (63)、**int** 型で表現できる最小値を表す (64)、最大値を表す (65) といった、文字型や整数型の表現範囲を表すための (66) 形式マクロは<(67)>ヘッダ内で定義されている。

▪ **char** を除く整数型には、(68)、(69)、(70) の3種類があり、右側のものは左側と同じまたはより広い表現範囲をもつ。

▪ 浮動小数点型には、(71)、(72)、(73) の三つの型があり、右側のものは左側と同じまたはより広い大きさや高い精度をもつ。

▪ `5 / 2.5` や `5.7 / 3` などの、オペランドの型が異なる算術演算では、(74) の型変換が行われる。一方、プログラム上で (75) な型変換を指定して型変換を行うこともでき、そのような型変換を行うことを (76) という。整数値17を、**double** 型に (76) した2で割る演算は、“`17 /` (77) `2`” あるいは、“`17 /` (78) `(2)`” によって行える。

▪ 以下に示すのは、`0.0` から `1.0` まで `0.001` ずつ増やしていく様子を表示するプログラムであり、(79) のほうがより真の値に近い値が表示される。

▶ 選択肢：(a) ❶　(b) ❷

❶
```
for (float x = 0.0F; x <= 1.0F; x += 0.001F)
    cout << (80) << '\n';
```

❷
```
for (int i = 0; i <= 1000; i++)
    cout << (81)<float>(i) / 1000 << '\n';
```

▪ 以下に示すのは、**int** 型の変数 *n* と整数リテラル5の型情報を表示するプログラムである。
※実行結果は一例である（処理系に依存する）。

```
#include <iostream>
#include <(82)>
int n;
cout << "nの型は" << (83)(n)(84) << '\n';
cout << "5の型は" << (83)(5)(84) << '\n';
```

```
nの型はint
5の型はint
```

▪ 以下に示すのは、**int** 型の同義語である *INTEGER* を作る宣言である。

```
(85) int INTEGER;
```