C++プログラミング講習

変数

1. 変数

変数は、データを保存しておくメモリ領域のことです。

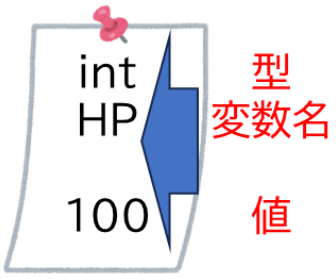
データというのは、ゲームで例えるならば、

整数の HP,攻撃力,スコア

文字列のプレイヤー名

とかですかね…

簡単に言うと、変数はメモです。



変数には名前をつけることができ、変数名といいます。

ここでは、HP が変数名になっています。

変数のデータを値と呼び、ここでは、100 が HP という変数の値です。

では int は何でしょうか？

int は変数の型の一種です。

型は保存できるデータの種類です。さっきの例だと整数や文字列といったものが型です。

C++の様々な型

C++での型	日本語
int	整数型(正確には 32bit 符号なし整数型)
float	実数型(正確には単精度浮動小数点数型)
string	文字列型
char	文字型
bool	真理値型

など…

ここで文字列と文字の違いを説明すると、

文字は a とか b とかの記号としての文字のことで、

文字列は、文字を複数集めたものです。

なぜ分ける必要があるのかはおまけに書くかも(書きました)

C++などの静的型付け言語では変数を宣言するときに型を指定する必要があります。

では、C++で変数を使ってみましょう

```
1   #include <bits/stdc++.h>
2   using namespace std;
3
4   int main() {
5
6       int A;
7       A = 10;
8       cout << A + 10 << endl;
9
10      int B = 100;
11      B = B + 3;
12      cout << B << endl;
13  }
```

上から見ていきます。

変数を使うとき、はじめに宣言を行う必要があります。

宣言は、型を書いた後に変数名を書くことでできます。

例:

```
1  int name;
```

6 行目では、int 型で変数名が A の変数を宣言しています。

変数に値を上書きするとき、代入を使います。つまり、前の値を消してから、新しい値を書き込みます。

例:

```
1  name = 100;
```

注意⚠この代入の=は数学の等号(例:3+4=7)とは全く別物なので気をつけましょう

7 行目では、変数 A に 10 を代入しています。

変数の値を見たいときは、参照をします。変数名を書くだけです。

例:

```
1  cout << name << endl;
```

10 行目 変数を宣言と同時に代入を行うこともできます。初期化とも言います。

ここでクイズ

12 行目の出力はどうなるでしょうか。正解は[　　]

なぜそうなるかを見てみましょう。

```
1  int B = 100;
2  B = B + 3;
```

実はこれはさっき1章でててきた演算子の優先順位が関わっています。

プログラムは基本上から下、左から右に実行されますが、演算子には、優先順位があることを、1章でやりました。そして代入＝の優先順位は今まで見てきた演算子の中で、最も優先順位が低い演算子です。なので、

```
1  int B = 100;
2  B = (B + 3);//←この順番
```

B ＋ 3 が実行されてから、B ＋ 3 の値が B に代入されます。

[参考] C と C++の演算子-演算子の優先順位

このコードはこのように書くこともできます

```
1  int B = 100;
2  B += 3;
```

優先順位は代入＝と同じで、他の四則演算もできます。+= -= *= /= %=

```
1   #include <bits/stdc++.h>
2   using namespace std;
3
4   int main() {
5       int A = 10, B;
6       B = A;
7       A = 100;
8       cout << A << endl;
9       cout << B << endl;
10  }
```

5 行目 カンマ , で区切ることで、一度に同じ型の変数を複数宣言できます。

6 行目では、B に A の値をコピーしています。

コピーした後、A の値が変わっても B は影響を受けません。

変数名のルール

・数字で始まる名前

・半角のアンダーバー _ 以外の記号が入っている名前

・キーワード(すでに C++で使われている単語 例:型名)[参考]:キーワード (C++)

・同じ変数名(スコープが違う場合を除く)

型変換

異なる型同士の計算では、型変換が行われます。

できない方同士は、コンパイルエラーになります。(例: 文字列と数字)

```
1   #include <bits/stdc++.h>
2   using namespace std;
3
4   int main() {
5       int A = 10;
6       float pie = 3.14;
7       string S = "FFT";
8       cout << (int)pie << endl;
9       cout << pie * A * A << endl;
10      cout << 10 + pie << endl;
11      cout << A / pie << endl;
12      pie = A;
13      cout << pie << endl;
14      /*
15      ↓コンパイルエラーになる処理
16      cout << S + A << endl;
17      cout << pie * A << endl;
18      S = A;
19      */
20  }
```

浮動小数点数の小数部の桁数を指定したいときは出力する前にこれをしてください。

```
1  cout << fixed << setprecision(15);  //15桁に設定
```

8 行目では、計算せずに、ただ型の変換だけを行っていて、そのような型変換を

キャストと呼びます。浮動小数点数型の数を整数型の数にキャストすると、小数点以下

の値が切り捨てられます。

入力を受け取る

```cpp
#include <bits/stdc++.h>
using namespace std;
int main() {
    int a; string s;
    cin >> a >> s;
    cout << a << endl << s << endl;
}
```

cin と>>を使います。変数に入力の値をいれることができます。

変数のスコープ

main 関数は、{ }で囲われています。この{ }で囲われている部分をブロックと言います。

ブロック内で定義された変数は、それより内側のブロックでしか使えないというルールがあります。また、その変数が使える範囲をスコープといいます

同じ名前の変数が複数ある場合、最も内側で定義された変数を使います。(同じブロック内で同じ変数を定義することはできない)

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    int A = 10, B = 3;
    {
        string A = "HELLO", C = "CPP";
        cout << A << endl; //より内側で定義された方を使う
        cout << B << endl;
    }
    //cout << C << endl; ←Cをスコープ外で参照しているのでコンパイルエラー
    cout << A << endl;
    cout << B << endl;
}
```

また、関数の外部で変数を定義することもでき、そのような変数をグローバル変数と呼びます。

おまけ

コンピューターで半角文字を扱うとき、ASCII コードがよく使われています。

C++もそうで、文字には、ASCII コードに対応した数字が振り分けられています。

文字	10進	16進	文字	10進	16進	文字	10進	16進	文字	10進	16進	文字	10進	16進	文字	10進	16進	文字	10進	16進
NUL	0	00	DLE	16	10	SP	32	20	0	48	30	@	64	40	P	80	50	`	96	60	p	112	70
SOH	1	01	DC1	17	11	!	33	21	1	49	31	A	65	41	Q	81	51	a	97	61	q	113	71
STX	2	02	DC2	18	12	"	34	22	2	50	32	B	66	42	R	82	52	b	98	62	r	114	72
ETX	3	03	DC3	19	13	#	35	23	3	51	33	C	67	43	S	83	53	c	99	63	s	115	73
EOT	4	04	DC4	20	14	$	36	24	4	52	34	D	68	44	T	84	54	d	100	64	t	116	74
ENQ	5	05	NAK	21	15	%	37	25	5	53	35	E	69	45	U	85	55	e	101	65	u	117	75
ACK	6	06	SYN	22	16	&	38	26	6	54	36	F	70	46	V	86	56	f	102	66	v	118	76
BEL	7	07	ETB	23	17	'	39	27	7	55	37	G	71	47	W	87	57	g	103	67	w	119	77
BS	8	08	CAN	24	18	(	40	28	8	56	38	H	72	48	X	88	58	h	104	68	x	120	78
HT	9	09	EM	25	19	)	41	29	9	57	39	I	73	49	Y	89	59	i	105	69	y	121	79
LF*	10	0a	SUB	26	1a	*	42	2a	:	58	3a	J	74	4a	Z	90	5a	j	106	6a	z	122	7a
VT	11	0b	ESC	27	1b	+	43	2b	;	59	3b	K	75	4b	[	91	5b	k	107	6b	{	123	7b
FF*	12	0c	FS	28	1c	,	44	2c	<	60	3c	L	76	4c	¥	92	5c	l	108	6c	|	124	7c
CR	13	0d	GS	29	1d	-	45	2d	=	61	3d	M	77	4d	]	93	5d	m	109	6d	}	125	7d
SO	14	0e	RS	30	1e	.	46	2e	>	62	3e	N	78	4e	^	94	5e	n	110	6e	~	126	7e
SI	15	0f	US	31	1f	/	47	2f	?	63	3f	O	79	4f	_	95	5f	o	111	6f	DEL	127	7f

(e-words.jp/w/ASCII.html より)

```cpp
#include <bits/stdc++.h>
using namespace std;
int main() {
    char C = 'a';
    cout << (int)C << endl;
}
```

このように char 型の変数を整数型に型変換させると、その文字に対応する 10 進数の ASCII コードの値が見られます。

またアルファベットは、連番になっているため、このような足し算引き算ができます。

```cpp
#include <bits/stdc++.h>
using namespace std;
int main() {
    char C = 'A';
    cout << 'D' - C << endl;
    C += 1;
    cout << C << endl;
    cout << C + 1 << endl;
    cout << char(C + 1) << endl;
    cout << char(65) << endl;
}
```

初期化する前の変数

```cpp
#include <bits/stdc++.h>
using namespace std;
int main() {
    int a;
    cout << a << endl;
}
```

実行してみると、環境によっては、変な値が出てきたり、0 になったり、コンパイルエラーになったりします。初期化していないローカル変数を読み取ると、未定義動作を引き起こします。未定義動作については未定義動作-wikipedia 参考

未定義動作は、なにが起こるのかがわからないのでなるべくやらないように気をつけましょう。

オーバーフロー 参考:【2038 年問題】世界中のシステムがぶっ壊れる？

32bit 符号付き整数型である int 型は名前の通り、メモリ内部では32桁の2進数で表されています

正の整数のときは、符号部は0で、負の整数のときは、符号部を1にしています。

つまり、int 型は-(2 を 32 回かけた数)～(2 を 32 回かけた数)－1までの範囲の数(-2147483648～2147483647)しか表現できません。この範囲を超えてしまうことをオーバーフローと言い、オーバーフローすると正しい計算結果が求められません。

```cpp
#include <bits/stdc++.h>
using namespace std;
int main() {
    int i = 2000000000;
    cout << i + 2000000000 << endl;
    cout << (i + 2000000000) / 100 << endl;//計算途中に超えるのもだめ
}
```

これを回避するためには、int64_t 型を使います。
int64_t 型は、64bit 符号付き整数型で
-9223372036854775808～9223372036854775807 を表せます。
int を int64_t に変えて実行すると、正しい結果になりました。
ちなみに C++では符号つき整数型のオーバーフローは未定義動作です。