Hello World

ネタで書いたHello Worldのプログラム

最終的なソースコード

```
#include <iostream>
using namespace std;

int main()
{
    char* p = (char*)main;
    int current = 0; // 今何文字目書いてるか
    int wanted = 0;  // 欲しい文字
    int count = 0;   // 何回ループしたか

    char list[] = { 72, 101, 108, 108, 111, 32, 87, 114, 111, 108, 100 };

    do {
        wanted = list[current];

        char c = *p; // ポインタの指す場所にある値を文字として取得
        p++;         // ポインタを進める

        if (c == wanted && ++current) {
            cout << c;
        }

    } while (current < 11 && ++count);

    // 参考までに何回ループしたか表示
    cout << endl;
    cout << count;
    return 0;
}
```

概要

`cout << "Hello World" << endl`

これだけですむ伝統的な"Hello World"をあえて無駄な事をして表示してみるプログラムである。

このプログラムは適当にメモリの中を探索して文字の"Hello World"に該当する値があったらそれを表示するという、無駄な方法で"Hello World"を表示している。

なのでメモリの中に"Hello World"という文字扱いできる値が存在しなければ無限ループする事になるはずだ。

処理の説明

とりあえずmain関数

なにごともまずはここから始まる。

```
int main() {
    return 0;
}
```

メモリのどこから探索するか

とりあえずメモリの中を探していくが、**どこ**から探していくのかという事を決めねばならない。

今回は「main関数のある場所」から探していく事にした。

```
int main()
{
    char* p = (char*)main;
    return 0;
}
```

関数名はその関数のあるアドレスを表すので、それを`char*`にキャストして変数に入れておく。

とりあえず文字として表示してみる

```
#include <iostream>
using namespace std;

int main()
{
    char* p = (char*)main;

    char c = *p;        // pの位置にある値を文字として取得
    cout << c << endl; // 表示

    return 0;
}
```

ポインタの場所にある値を文字として変数にも保存して、それを表示する。

メモリの中に入っている値がそもそも文字として表示できる値じゃない場合は何も表示されないけど

ポインタを進める(次の文字へ)

ポイントを今いる場所から1個先へ進めて、またそこにある値を文字として表示してみる。

```
#include <iostream>
using namespace std;

int main()
{
    char* p = (char*)main;

    char c = *p;        // pの位置にある値を文字として取得
    cout << c << endl; // 表示

    // ポインタを進めてまたそこにある値を文字として表示
    p++;
    c = *p;
    cout << c << endl;

    return 0;
}
```

`p++`すると`p`はcharのポインタなので1バイト分先へ進む、これを繰り返せば、とりあえずmain関数がある場所から順番にメモリの中を探索していくことができるだろう。

とりあえず文字を表示し続ける

※このプログラムは動かすと無限ループしてしまうので実際には動かさない方がいい。

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
    char* p = (char*)main;

    do {
        // 文字を取得したら、ポインタを進める
        char c = *p;
        p++;

        cout << c;
    } while (true);

    return 0;
}
```

`do~while`で無限にポインタを進めながら、1文字ずつ文字を表示し続けるループである。

なぜ`do~while`を使ったのかと言えば特に理由はない、滅多に使わないからあえて無駄に使ってみた。

最初の一文字目"H"を表示する

最終的な目標は"Hello World"と表示することだが、とりあえず最初の1文字目の"H"を表示してみよう。

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
    char* p = (char*)main;

    do {
        // 文字を取得したら、ポインタを進める
        char c = *p;
        p++;

        // 目的の文字だったら表示
        if (c == 'H') {
            cout << c;
            break;
        }

    } while (true);

    return 0;
}
```

メモリの中を次々と探索して、"H"に該当する値だった場合だけ表示してループを抜ける。

main関数のある場所からメモリを探索していくので、そのどこかで"H"に該当する値があれば"H"表示されるはずだ。

運悪くメモリのどこにも"H"に該当する値がない場合はきっと無限ループする。

"H"の次の文字を表示するにはどうするか

とりあえず"H"は表示できるが、"H"を表示したら次は"e"を表示したい。

実現する方法は沢山あると思うが、このプログラムではこんな考え方をした。

1. 表示する文字は**今何文字目を表示したいか**がわかれば決まる
2. 表示したい文字を覚えておいて、メモリの中身とその表示したい文字が一致したら表示

```
#include <iostream>
```

```
#include <string>
using namespace std;

int main()
{
    char* p = (char*)main;

    // 1. 変数を用意
    int current = 0;              // 今何文字目？
    char wanted = 0;              // 表示したい文字
    char list[] = "Hello World"; // 表示する文字リスト

    do {
        // 2. 表示したい文字を決める
        wanted = list[current];

        // 文字を取得したら、ポインタを進める
        char c = *p;
        p++;

        if (c == wanted) { // 3. 欲しい文字とメモリの値が一致するか
            cout << c;
            ++current; // 4. 今何文字目？を増やす
        }

    } while (current <= 11); // 5. Hello Worldは11文字なので11文字になるまでループとする

    return 0;
}
```

変数が増えたり処理もあれこれ追加されたので追加したり変更した場所には番号を書いておいた。

そもそも表示する文字リストで"Hello World"という文字を用意してしまった時点でそれを表示すればええやんという気持ちにもなってしまうが、まぁそれを言うのは野暮だろう、あえて無駄なコードを書いているのだから。

これでほぼ最終的なソースコードの形になってきた。

何回ループしたのか？

このプログラムは目的の文字が見つかるまでループするが、実際どれくらいループする事になるのか

参考までにループ回数を表示する処理も入れておこう。

```
#include <iostream>
#include <string>
using namespace std;
```

```
int main()
{
    // 中略
    int count = 0;

    do {
        // 中略
    } while (current <= 11 && ++count);

    cout << endl;
    cout << count;

    return 0;
}
```

長いので途中は省略した。

ループ回数用の変数を用意して、whileの条件式の後ろに`++count`している。

`++count`をあえてwhileの条件式に入れる必要もないが

このプログラムを書いているとき、スクショの範囲に収めるために行数を節約したかったのでこうしただけである。

複数の条件を繋げて書いた場合、一般的には左の条件から評価される。

`if(A && B)`と合った場合、Aが`false`の時点で全体が`false`と決まるのでその場合、右の式は評価されない。

プログラムにおいて**評価されない**というのは、その処理は実行されないという意味である。

`if(current <= 11 && ++count)`であれば、左の条件はループが続く限り`true`なので、右の条件も評価しないと全体の結果がわからない。

よって、ループ中は常に左の条件、右の条件が**評価**(実行)されるので、その動作を利用してカウントを増やしている。

こういう書き方は言語によっては**ショートサーキット**と呼ばれたりもする。

可読性が落ちるので多用するのは良くないが、ときには便利なので覚えておいても損はしないだろう。

また今回のプログラムで文字を表示する部分の処理もショートサーキットが使える。

```
        if (c == wanted) {
            cout << c;
            ++current;
        }
```

この処理は、このように書き換えても問題はない。

```
    if (c == wanted && ++current) {
        cout << c;
    }
```

`c == wanted`が`true`にならない限り右の条件は評価されないので、元の処理とやっていることは同じになる。

仕上げ

プログラム内で表示する文字リストとして"Hello World"という文字列を書いている場所がある。

```
    // 表示する文字リスト
    char list[] = "Hello World";
```

別にこれでもいいのだが、なんとなくこれから"Hello World"を表示しようというときに、すでにその文字が見えているというのもちょっとカッコ悪い気がする。

もっとエレガントに無駄で頭のいい方法があればいいのだが、そこまで思いつかないので陳腐なごまかしとして、とりあえず文字に見えないような配慮だけしておきたいと思う。

結局文字というのは内部的には数値でしかないので、数値の配列にする。ただそれだけである。

```
char list[] = "Hello World";
```

👆これを、👇こうする

```
char list[] = { 72, 101, 108, 108, 111, 32, 87, 114, 111, 108, 100 };
```

見た目は違うがやっていることは同じだ、文字で書くか、数値で書くかの違いしかない。

これで最初に載せた最終的なソースコードと同じになる。

以上、めでたしめでたし。