C++講習会 #3 配列と関数

高校 2 年 赤澤侑

2020/09/27

1 配列

1.1 (生) 配列

“与えられた数字列を逆順に出力する”プログラムを考えてみます．このときに必要になるのが配列というものです．ソースコード 1 を見てみてください．main 関数の最初に int a[100000] = {}; という文があるのが判りますね．これが配列です．配列というのはある型の変数をたくさん連ねたものです．この場合は 100000 個分のメモリ領域を確保して int 型の変数を連ねています．そして実際にこのプログラムを実行してみると入力した数字列が逆順に表示されることがわかるでしょう．配列の i 番目の値にアクセスするときには a[i] というふうに書きます．これは文字列の操作をやったときと同じですね．よく考えてみれば文字列も char 型の変数を連ねたものと解釈できる[1]ので，このことにも納得がいきます．

ソースコード 1: 配列

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

int main(){
        int a[100000];
        int N;
        cin >> N;
        for(int i = 0; i < N; ++i)cin >> a[i];
        for(int i = N-1; i >= 0; --i)cout << a[i] << ' ';
        cout << "\n";
        return 0;
}
```

配列は最初に要素を指定することができます．ソースコード 2 を見てください．これを実行すると 2 4 8 256 1024 が一行ずつ出力されることでしょう．このように型 変数名 [大きさ] = { 要素 } とすることで，配列の中身を初期化することができます．

では初期化しなかったらどうでしょうか？多くの場合 0 で初期化されますが必ずしもそうとは限らないので注意しましょう．不定であることもあります．全て 0 にしたければ int a[100000] = ; などと記述するか，ループを回して全て 0 を代入してしまえば良いです．

また，配列の要素数に (特殊なものを除いて) 変数を用いることはできません[2]ので注意してください．

そして最大の注意として配列の添字 ([] の中に入れる数字) は先頭の要素で 0 です．即ち 0 始まりで数えているということも注意が必要です．ソースコード 1 では 0〜99999 にアクセスできます．

[1] C++に於いては std::string は char 型の配列と全く同じものではありませんが，実際 C 言語では文字列を char 型の配列として扱います．

[2] ローカル変数の場合は要素数に const 定数というものを指定することができます．然し，グローバル変数に対しては const 定数も使うことができません．

ソースコード 2: 配列の初期化

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

int main(){
        int a[5] = {2,4,8,256,1024};
        for(int i = 0; i < 5; ++i)cout << a[i] << endl;
        return 0;
}
```

課題 1 ソースコード 1 を動かしてみよ

課題 2 ソースコード 2 を動かしてみよ

1.2 std::vector

vector は可変長配列と呼ばれるもので要素数を変更することができます．次のソースコード 3 を見てください．まず，vector を使うためには#include<vector>と記述する必要があります．vector 型の変数の宣言は書いてあるとおり “vector<型> 変数名 (大きさ)”です．同じ値で初期化する場合には “vector<型> 変数名 (大きさ, 初期値)”とすることができます．便利ですね．ではコードを詳しく見てみましょう．生配列と同じように要素にアクセスするときには a[i] でアクセスできます．そしてここが vector が可変長配列と呼ばれる所以ですが，a.push_back(x) という関数が書かれています．これは a の最後に x という要素を追加してねーーという意味です．またここでは書いていませんが a.resize(x) とすれば配列全体の大きさを x にすることができます．

ソースコード 3: std::vector

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

int main(){
        int N;
        cin >> N;
        vector<int> a(N);
        for(int i = 0; i < N; ++i){
                cin >> a[i];
        }
        cout << a.size() << endl;
        for(int i = 0; i < a.size(); ++i)cout << a[i] << '␣'; cout << endl;

        a.push_back(10);
        cout << a.size() << endl;
        for(int i = 0; i < a.size(); ++i)cout << a[i] << '␣'; cout << endl;

        a.push_back(256);
        cout << a.size() << endl;
        for(int i = 0; i < a.size(); ++i)cout << a[i] << '␣'; cout << endl;
        return 0;
}
```

課題 3 ソースコード 3 を動かしてみよ

1.3 多次元配列

実は配列の要素に配列を指定することもできます．配列の配列とも呼べるものですが，例えば配列に配列を入れたもの (これを二次元配列という) はちょうど表のような形になるのがわかるでしょう．生配列と vector では次のようにして二次元配列を作ることができます．

ソースコード 4: 二次元配列の書き方

```
int a[100][100];
vector<vector<int>> a(100, vector<int>(100));
```

二次元配列の各要素へのアクセスは a[i][j] とすれば良いです．最初のうちは使う機会が少ないかもしれませんが，たとえば二次元平面を表現したり，グラフの表現に二次元配列は必須です．また要素 2 つだけではなく 3 つ，4 つとすることもできます．これらをすべて合わせて多次元配列といいます．

課題　Verify-配列

A. N 個の非負整数からなる数字列 a が与えられる．a の連続する 2 要素であってその差の絶対値が K 以下であるものの総数を求めよ．($2 \leq N \leq 100$, $|a_i| < 1000$, $0 \leq K \leq 1000$)

B. 3×3 マスの盤面が与えられる．盤面には'o' もしくは'x' のいずれかが書いてあり，o が縦もしくは横に揃っていれば@$\sum$ くんの勝ち，x が揃っていれば Nyapo くんの勝ちである．どちらが勝利したか？前者なら “Atsum”, 後者なら “Nyapo” と出力せよ．(どちらも買っているような状況は存在しないとする)

C. N 個の要素からなる数字列 a に対して Q 個の query に答えよ．query は"1 x"もしくは"2"という形をしていて前者の場合は数字列の末尾に x を追加し，後者の場合は数字列を全て出力せよ．($0 \leq N \leq 100$, $0 \leq Q \leq 1000$, $|a_i| < 1000$, $|x| < 1000$)

2　関数

ここでソースコード 3 をもう一度確認してください．ソースコードの 12 行目から同じような処理が何度も書かれていることがわかるでしょう．今回は 3 回しか繰り返していませんが，これが 10 回も 20 回も繰り返す必要があるならそれはとても冗長になってしまいます．このように同じ動作は関数に指定舞うことができます．数学に於いては関数というのはある集合からある集合への対応関係を言ったりするわけですが，プログラムの世界ではある決まった動作をするだけのこともあります．例えば先ほど紹介した push_back や resize，最初のプログラムとして扱った pow なども関数の一つです．関数は次のように記述します．

ソースコード 5: 関数

```
戻り値の型 hoge(引数){
  ここに処理が入る
  return 戻り値;
}

//具体的には
int plus(int a, int b){
  return a + b;
}
```

一般に関数というのは値を返します．それは真偽値 (bool) であったり整数 (int, long long) であったり，文字列 (string) であったりあるいは配列 (vector) であったりします．ただし，何も返さない戻り値なしという関数も作れます．そのようなときには “戻り値の型” のところに void と書き，そうでないときには戻り値となる型を書きます．変数名は好きに付けて大丈夫ですが，すでに宣言されている関数[3]や予約語とダブることがないようにしましょう．

ここで先程のプログラムの重複部分を関数にすると次のようになります．

[3]実はオーバーロードというものがあって同一関数名の関数を作ることができますが，今回はその説明は省きます．

ソースコード 6: 一部の処理を関数化

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

void show_all(vector<int> a){
        cout << a.size() << endl;
        for(int i = 0; i < a.size(); ++i)cout << a[i] << '␣'; cout << endl;
        return;
}

int main(){
        int N;
        cin >> N;
        vector<int> a(N);
        for(int i = 0; i < N; ++i){
                cin >> a[i];
        }
        show_all(a);

        a.push_back(10);
        show_all(a);

        a.push_back(256);
        show_all(a);
        return 0;
}
```

プログラムが見やすくなったばかりか，何をやっているのかわかりやすくなりましたね．本来ならばもう少し詳しく説明すべきですが今回はここまでとします．

課題 4 ソースコード 6 を動かしてみよ

課題　Verify-関数

A. int 型の変数 a, b が与えられるので a と b の差の絶対値即ち $|a-b|$ を返す関数を作れ (そのような関数として abs という関数があるがそれは用いずに実装せよ) .

B. int 型の変数 x, a, b が与えられるので x^2+ax+b を返す関数を作れ.

C. string 型の変数 s で表現された数字が与えられるので，その桁和を返す関数を作れ.

D. 英小文字からなる文字列が与えられるのでその中に'a' が含まれるか判定する関数を作れ.