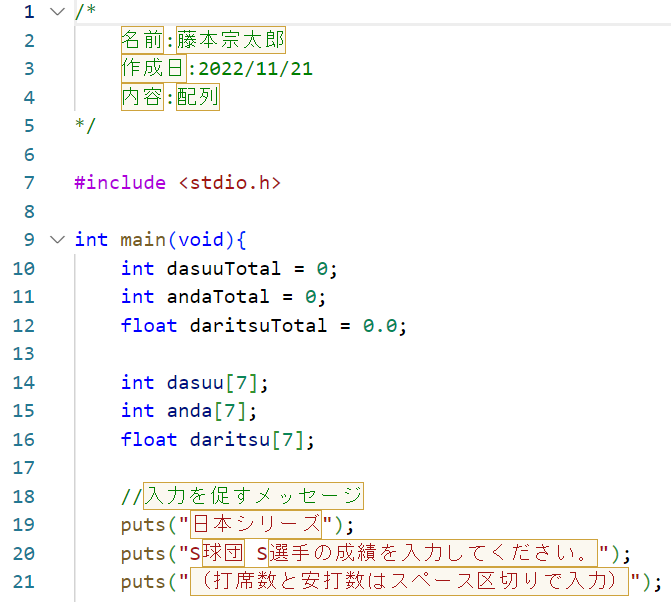
**◆目的**

　コンピュータでは多くのデータを扱うことが頻繫に起こる。プログラムで多くのデータを扱う際、すべてのデータに異なる名前を付けて変数を宣言しようとすると、宣言するだけで大変な作業になる。また、プログラム中で使用する際も何かと不便である。このような問題を解決するため「配列」と呼ばれるものがある。「配列」は、同じ型のデータをひとまとめにして1つの名前を付けたものである。本実験では、配列の宣言方法、使用方法について学び、理解することを目的とする。

**◆課題**

　変数を使用せずに配列を使用して打席数・安打数・打率を記録するようにプログラムを改良しなさい。

**<ソースコード>**



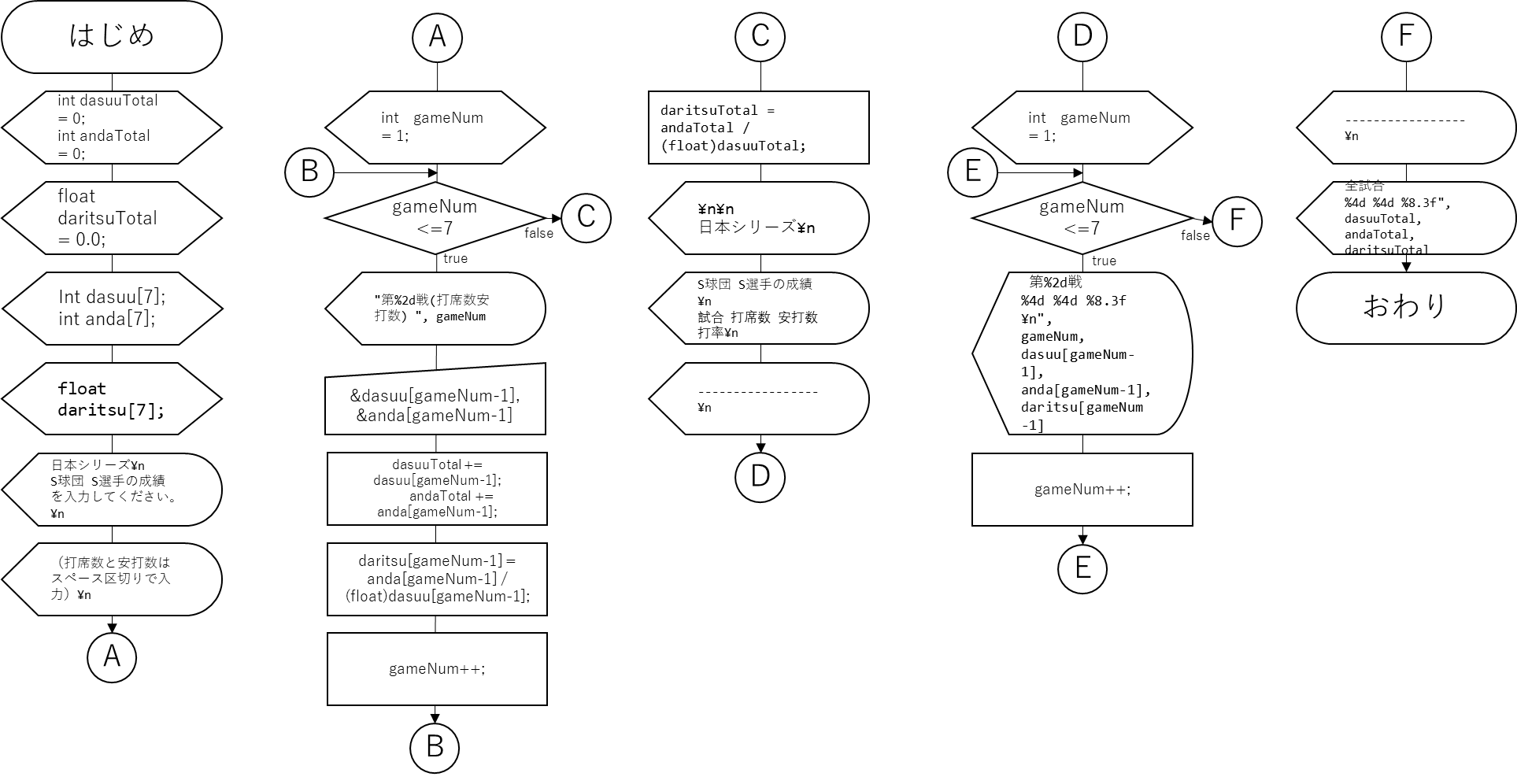
**◆調査(配列の使い方・使用時の注意点)**

・【型 配列名[] = {要素}】

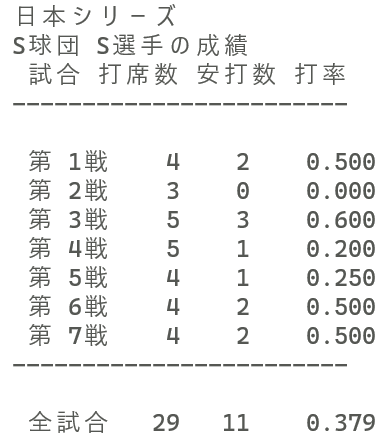
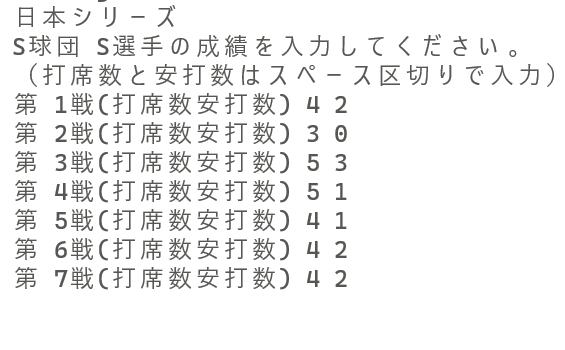
・【型 配列名[要素数] = {要素}】

・添字は0からある

**<流れ図>**



**<実行結果>**



**◆調査(配列の使い方・使用時の注意点)**

・【型 配列名[] = {要素};】

・【型 配列名[要素数] = {要素};】

・添字は0からある

**◆考察**

　今回の実験で使用しているコンパイラは、**存在しない配列の要素**を指定したプログラムをコンパイルできるということに気が付いた。そして、下記のプログラム(*<疑問>*)の『**int int1 = array2[1];**』はどのように機能するのか、疑問が生じた。

**<疑問>**

テーブル

自動的に生成された説明

下記のプログラムの実行結果(*<変数の配置確認プログラムの実行結果>*)からわかるように、変数は宣言するのが**早かった順**に、アドレスが大きい方から**小さい方**に向かって**連続して**(intは4byte)スタックに置かれている。よって、私は上記のプログラム(*<疑問>*)の『**int int1 = array2[1];**』ではarray2[**1**] *(&array2[1] = &array2[0]+0x4 =0x93FE98)* が**array1[0]**になり、int1に**1**が代入されると予想した。

**<変数の配置確認プログラム> 　　　　<変数の配置確認プログラムの実行結果>**

テキスト

自動的に生成された説明テキスト

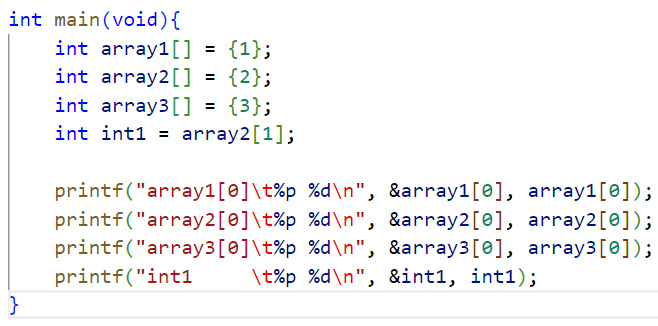
自動的に生成された説明

　その予想が正しいか確認すべく、下記のプログラム(*<存在しない要素プログラム>*)を実行した。予想は間違っていた。先ほどのプログラム(*<変数の配置確認プログラム>*)では、変数が宣言するのが**早かった順**にスタックに置かれていたが、このプログラムではそのようになっていない(*<存在しない要素プログラムの実行結果>*)ためだ。

**&array1[0] > &array2[0] > &array3[0]**ではなく、**&array2[0] > &array1[0] > &array3[0]**に何故かなってしまっているのだ。

**<存在しない要素プログラム>　　　　　　　　　　　　<存在しない要素プログラムの実行結果>**

テキスト

自動的に生成された説明

※ちなみにこの『13631156』は、

main関数が呼ばれた直後に実行される『**push ebp**』という命令により、スタックに格納された

**ebpレジスタ**の値のようだ。(逆アセンブルより判明)

**&array2[0] > &array1[0] > &array3[0]になる理由**をしばらく考えたが分からなかった。また、自分の中で、**『ある配列変数で存在しない要素を指定することによって、他の変数を参照する』**という目標ができた。よって、何故、変数が宣言するのが早かった順にスタックに置かれていないのか、また、ある配列変数で存在しない要素を指定した時、他の変数を指定したことになるという結果はどうにかして得られないものかと、Yahoo!知恵袋で質問した。するとこのような回答が得られた。

*・理由なんてありません。強いて言えば、「あなたが使ったコンパイラが宣言順に関係なく、宣言順位外の独自に定めた優先順位に従って処理をした」というだけの話です。*

*・宣言した変数をどうメモリに配置するかは言語としては特に決まりはなく、なんの期待もできません。もちろん、コンパイラは何らかの法則にしたがって配置をするので、コンパイラを決めるとその順番も予想できるでしょうが、少なくともC/C++言語のレベルではなんの決まりはありません。****ただし、構造体の中身のメモリ上の配置は記述順になります。***

*(Yahoo!知恵袋「スタックに置かれる変数」*[*https://detail.chiebukuro.yahoo.co.jp/qa/question\_detail/q10271215985*](https://detail.chiebukuro.yahoo.co.jp/qa/question_detail/q10271215985)*）*

結局、どのような法則によって変数が宣言するのが早かった順にスタックに配置されなかったのかは分からなかったが、構造体のメンバは特別に必ず、宣言した順にスタックに保持されるということが分かった。それなら構造体を使用することで、先程述べた『ある配列変数で存在しない要素を指定することによって、他の変数を参照する』という目標は達成できそうだ。

構造体のプログラム(*<存在しない要素プログラム②>*)を組んで実行した。そのプログラムの実行結果(*<存在しない要素プログラム②の実行結果>*)からわかるように、確かに構造体のメンバは宣言した順にスタックに配置されている。また、構造体が1つの変数として扱われているためか、構造体のメンバはアドレスが大きい方に向かってスタックに保持されている。『**int int1 = struct1.array2[1];**』(*<存在しない要素プログラム②>*)でstruct1.array3[0]の値を取得しているのが確認できる。遂に目標を達成することができたのである。

テキスト

自動的に生成された説明**<存在しない要素プログラム②>** 　**<存在しない要素プログラム②の実行結果>**

