プログラミング－言語・基礎・実践－ 18－14
（1994. 7. 21）

移植性のあるCの継続ライブラリ

多田 好克
電気通信大学 大学院 情報システム学研究科

Schemeなどで積極的に利用されている継続(continuation)を、ライブラリとしてC言語に付加した。本報告では、この継続ライブラリの実現法と問題点とを議論する。本ライブラリでは、すべての処理をCで記述した。また、スタック領域の同定やレジスタコンテクストの取得などもCの標準ライブラリのみを使って実現した。したがって、ライブラリ自体の移植性が高く、様々なC言語処理系のライブラリ関数群として継続の機能を活用することができる。現在、本ライブラリは80386, 68030, SPARC, R3000の各種CPUを対象にして、UnixおよびMS-DOSの各種オペレーティングシステム上のC, Gnu C, Turbo Cで、ソースコードを変更することなしに共用可能である。

A Portable Continuation Library

Yoshikatsu Tada

Graduate School of Information Systems
University of Electro-Communications

1-5-1 Choofugaoka Choofu-shi, Tokyo 182 JAPAN

This paper describes a full portable implementation of a C continuation library, which, without any modification, can be used by various compilers on manifold operating systems. Whole library, including the code of getting registers context or getting the address of current stack area, was written by C, so that the library is independent from hardwares, compilers and operating systems. In this paper, we summarize our implementation of the continuation library and discuss the techniques we used to make the library totally portable.

1. はじめに

継続(continuation)は、Scheme[Stee78]で積極的に取り入れられている言語機能の１つである。我々は、この継続に類似の機能をＣ言語にライブラリとして実現した。本報告では、その実現法と問題点とを議論する。

また、本継続ライブラリの実現においては、作成したライブラリの移植性を重視した。これは、ライブラリ関数も言語仕様の一部と考えられるほどＣ言語の仕様がライブラリに依存しているためであり、各種Ｃ処理系において同一のインタフェースとセマンティクスを持ったライブラリ関数を提供する必要があると考えたからである。つまり、ライブラリプログラム自体の移植性を向上させて各種Ｃ処理系のライブラリへの組込みを可能とし、同一のインタフェースとセマンティクスとを提供できることを目指したわけである。

本報告は、以下に示す順序でＣの継続ライブラリを議論する。まず、２章では継続ライブラリのインタフェースとセマンティクスとを定義する。３章では継続ライブラリの実現法を議論する。特に、3.2節ではライブラリの移植性を向上させるために、我々が考案し、採用した技法を示す。４章は本継続ライブラリを使ったプログラム例で、Ｃの継続の概念を明確にするために挿入したものである。５章では、今回作成した継続ライブラリの評価を行い、その問題点にも言及する。

2. 継続ライブラリのセマンティクス

本継続ライブラリは継続取得用関数c_get()と、取得した継続の実行を指示する関数c_cont(c)、それに継続を利用した処理の開始点となるメインルーチンc_main(argc, argv, envp)で構成される。また、取込みファイルcont.hが用意されており、継続ライブラリを使用する時に必要となるいくつかのデータ型が定義されている。

以下、この章では取込みファイルcont.hを簡単に説明し、それに続いて各関数の意味を説明する。

2.1. 取込みファイル：cont.h

取込みファイルcont.hでは、以下に列挙する３つの処理・宣言を実行する。

1) Ｃライブラリ関数setjmpとlongjmp用の取込みファイルsetjmp.hを取り込む。これらのライブラリ関数は、Ｃの継続実現において中心的な役割を果たす。

2) 継続実現のためのデータ構造を宣言し、その構造体へのポインタ型contを定義する。以下はこの定義の概要である。なお、この構造体の詳細は3.1節で説明する。

```
typedef struct {
    jmp_buf j;
    int     *s;
    int     *b;
} *cont;
```

3) 関数c_getとc_contの雛形宣言をする（以下に示すとおり）。

```
extern cont    c_get();
extern void    c_cont(cont);
```

2.2. 継続の取得：c_get

関数c_getは引数なしで呼び出され、その時点での継続を戻り値とする。将来、この継続を引数として関数c_contを呼び出すと、プログラムの実行はc_getの復帰直後に移動する（詳細は次節を参照）。

c_getの返す値は構造体へのポインタなので、値０には成り得ない。他方、c_contの呼出しによって継続の実行が開始された時には、c_getが戻り値０で復帰したように振舞う。

2.3. 継続実行の指示：c_cont

c_getによって取得した継続は、c_contを使って実行できる。継続の実行は、任意の時点で何度でも繰り返せるという点において、大域ジャンプ(non-local goto)やコルーチン呼出しとは

異なる。

c_contによって実行を指示された継続は、その継続の取得点直後から実行を開始する。この時、プログラムの実行環境は継続取得時の環境に等しい。より正確には、Cの大域変数には継続取得後の変更が反映されているが、その他の環境（関数の呼出し履歴や個々の関数の引数・局所変数の値など）には継続取得後の変更は反映されない。

継続ライブラリの実現法を変えれば、大域変数の値をも含めた継続を実現することもできる。しかし、大域変数の値を含めない環境の方が応用プログラムの記述に柔軟性が得られる。本継続ライブラリは、実用的な見地に立って、大域変数の値を継続環境に含めなかった。

2.4. 継続ライブラリの実行開始点：c_main

継続ライブラリの提供する機能を利用するプログラムは、通常の関数mainからではなく、関数c_mainから処理を開始する。このc_mainには関数main同様３つの引数argc, argv, envpが用意されており、外部（コマンド行）からの情報を受け取ることができる。

c_mainの使用例を初めとする継続ライブラリの使い方は、４章のプログラム例を参照のこと。

3. 継続ライブラリの実現法

継続ライブラリの移植性を高めるために、ライブラリの記述にはアセンブリ言語を一切使わずC言語のみを使用した。また、移植性の高いthreadsライブラリ[Tada92]を実現した際に議論し、実際に利用した様々な技法を今回も活用した。この章では、まず、継続を実現するための方法を議論し、続いて移植性を高めるために使用した技法について説明する。

3.1. 継続実現に必要な情報

継続を実現するためには、継続取得時の各種記憶域の内容と、その時点までの制御の履歴とを取得し、保存する必要がある。また、これらの情報を使って継続の実行ができなければならない。

2.3節で述べたように、本継続ライブラリは大域変数の内容を継続に保存されるべき情報とは考えていない。これは、応用プログラムの書き易さという、実用的な見地からの帰結である。したがって、本継続ライブラリの実現には、

1）レジスタの値（PCやSPの値を含む）
2）関数の引数や局所変数の値
3）制御の履歴

を取得・保存し、継続実行時にはこれらの情報を復元してやれば良い。なお、レジスタの値を復元する際にプログラムカウンタ(PC)の値も変化し、継続取得点から実行が開始されることになる。

C言語では、上述の各種情報の内の2)と3)の情報は共にスタックに格納されている。したがって、本継続ライブラリの実現では、継続取得時にスタックの全内容およびレジスタの値を保存し、継続実行時にはそれらをスタックとレジスタにそれぞれコピーする。

継続情報の保存場所にはCのデータ領域を使用した。本継続ライブラリでは大域変数の内容は継続に保存しないため、データ領域を継続情報の保存場所に利用し易い。ただし、継続情報に大域変数の値を含むよう変更した場合でも、ライブラリ関数malloc等に皮をかぶせれば同様の実現は可能である。

我々の作成した継続ライブラリは、継続取得時にmallocを使ってスタックおよびレジスタ保存領域を確保する。スタック保存領域の大きさは継続取得時のスタックの大きさに等しい。他方、レジスタ保存領域は2.1節で言及した構造体のメンバ名jの領域になる。ちなみに、mallocを使って確保したこの構造体には、レジスタの値以外に継続取得時のスタックへのポインタsとスタック保存領域へのポインタbとが含まれる。

3.2. 移植性の高い継続ライブラリ実現法

前節で示した各情報を取得、保存、復元するためには、CPUアーキテクチャ、ハードウェアアーキテクチャ、オペレーティングシステム、Cコンパイラ、Cライブラリの実現法を考慮した

プログラムを書かなければならない。たとえば、レジスタの値の保存はCPUアーキテクチャに依存するし、コピーすべきスタック領域の決定法はオペレーティングシステムやCコンパイラの実現法に依存すると考えられる。

本ライブラリの実装にあたっては、これらの依存部分の大半を、各種のC言語処理系に不偏な性質とライブラリとを利用して記述した。たとえば、レジスタの値の取得にはライブラリ関数のsetjmpを利用したし、コピーすべきスタック領域は関数mainの引数のアドレスと継続取得時に実行している関数c_getの局所変数のアドレスとより決定した。

以下、この章の残りの部分では移植性を高めるために使用した様々な技法について議論する。

3.2.1. レジスタの値の取得・復元

レジスタの値の取得にはCライブラリ関数の_setjmpを、復元には_longjmpをそれぞれ利用した。ただし、Cライブラリによっては_setjmp, _longjmpのないものもある[Tada92]。この場合は、setjmp, longjmpを利用した。実際にはマクロ名SETJMP, LONGJMPを使ってプログラミングし、システムごとに利用するライブラリ関数を定義している。

ライブラリ関数setjmpを実行した時の戻り値は0である。関数c_getではsetjmpによってレジスタの値を取得すると同時にsetjmpの戻り値を調べ、0であれば引続きスタックの内容を取得・保存する。その後、これらを保存した構造体へのポインタを戻り値として復帰する。

なお、setjmpを使って取得した継続を実行した場合、その処理の最後でlongjmpを実行する。その時、実行は今議論しているsetjmpに移動し、戻り値は1となる。c_getは、setjmpの戻り値が1であった場合、戻り値0で直ちに復帰する。

3.2.2. スタック領域の取得

取得・保存すべきスタック領域の範囲は、基本的には関数mainの引数のアドレスから継続取得時に実行している関数（つまりc_get）の局所変数のアドレスまでである。実際には、この範囲よりも少し広い範囲が取得・保存の対象となるが、本ライブラリでは以下に述べる実践的な方法によって範囲の上限、下限を決定している。

［上限の決定］：取得・保存すべきスタックのスタックトップはc_getの局所変数のアドレスよりも少し上にある*)。何バイト上にあるかはCPUアーキテクチャとコンパイラとに依存するが、静的に定まる値で押さえることは可能である。つまり、この値は関数c_getの呼び出され方などの動的な理由によっては変化しない。また、この値を大きく見積ると保存領域を浪費するが、継続のセマンティクスに影響を与えることはない。

以上のような考察を基に、本ライブラリではc_getの局所変数のアドレスから、ある定数値だけ範囲を広げたアドレスを保存すべきスタック領域の上限としている。また、この値はCコンパイラの出力するアセンブリプログラムを見て定めている。ただし、移植性を重視する場合にはc_getを再帰的に呼び出すことによって、この値を得ることもできる。

［下限の決定］：結論から先に述べると、スタック領域の下限は関数mainの引数argcのアドレスそのものである。継続に含むべき情報の範囲は、これよりも広いが実用的にはこれで十分である。つまり、argcのアドレスより下の部分は、本継続ライブラリで扱うすべての継続に共通であり、継続の操作において取得・保存・復元する必要はない。

3.2.3. スタック領域の復元

継続を実行するには、取得・保存したスタック領域を復元しなければならない。通常、スタックにはフレームポインタの値などのスタックのアドレスに依存した情報も含まれるので、スタック領域の情報は取得時と同一のアドレスに復元する必要がある。

また、継続のセマンティクスでは、継続取得後の処理で関数の復帰が繰り返され、取得時の

---

* 上というのはスタックの伸びる方向と考える。

フレームが既に存在しなくなった場合でも、当該継続を実行することができる（蛇足ながら、setjmpでは、このようなことはできない）。そのため、スタック領域の復元においては、復元処理を実行している関数のスタックフレームを壊さないような配慮が必要である。

ただし、スタック領域の復元においては、復元処理をする関数c_contのフレームだけを、復元スタックの領域外に移動させれば良い。なぜならc_contはlongjmpを使って制御を放棄するので、通常の関数復帰のように呼び出した関数のフレームを必要としないからである。

一般にスタックフレームの移動は、アセンブリ言語を使って記述する。これはスタックポインタの再設定などの処理が含まれるからである。しかし、本ライブラリの実現に際しては移植性を高めるために、スタックフレームの移動をC言語のみで記述した。以下にその方法を説明する。

上述の目的を達成するためには、c_contのフレームの移動先を特定する必要はない。つまり、継続ライブラリの実現においては、復元されるスタックの領域外にc_contのフレームを移動しさえすればそれで良い。そこでc_cont内で繰り返し自分自身を再帰的に呼び出し、使用フレームをスタックの上方へ徐々に追い出していく方法を採用した。

c_contは、まず、局所変数のアドレスと復元すべきスタック領域の上限とを比較し、前者が後者を越えて定数値以上離れるまで再帰呼出しを繰り返す。また、この時の定数値の決定法には3.2.2節の上限の決定時の議論が適用できる。

3.2.4. c_getとc_contの処理の概要

以上の議論を踏まえて、関数c_getは以下のような処理を実行する。

```
レジスタ保存用の構造体cを確保する
スタック保存用の領域bufを確保する
cのメンバsとbを設定する
cのメンバjを引数にしてSETJMPを実行
if (SETJMPの戻り値が0) {
    スタック領域のコピー
    return(cへのポインタ)
} else {
    return(0)
}
```

また、関数c_contは以下のような処理を実行する。

```
関数のフレームが復元するスタックの
    領域外に出るまで再帰呼出しを実行
引数で指示された構造体を使って
    スタック領域を復元する
LONGJMPを実行
```

4. 継続ライブラリを使ったプログラム例

継続の概念は、C言語で一般的に使われるまでには到っていない。この章では、2章および3章で定義した継続ライブラリを使う簡単なプログラムを示し、C言語における継続ライブラリの概念を補足する。

ここで示すのは、継続を使って2から順に素数を求めるプログラムである。4行目の記号定数N_PRIMEで指定した個数の素数を求め、その結果を画面に出力する。N_PRIMEが10の場合の出力は、

```
2,3,5,7,11,13,17,19,23,29, .....
```

のようになる。

プログラムはc_main（26行目）から実行を開始する。大域変数numberは2から順に1ずつ増加し、素数かどうかを調べるために使われる。配列の要素prime[n]にはn番目の素数が格納され、n_primeはこの配列の添字として使われる。

c_mainから実行を開始したプログラムが、初めに作る継続はp_cont[0]である。これは繰返し用のもので、大域変数candidateの値が1であれ

```
 1  #include <stdio.h>
 2  #include "cont.h"
 3
 4  #define N_PRIME 10                      /*  求める素数の個数          */
 5
 6  int     number;                         /*  素数の候補                */
 7
 8  int     n_prime;
 9  int     prime[N_PRIME+1];               /*  prime[n]はｎ番目の素数    */
10  cont    p_cont[N_PRIME+1];              /*  整除できるかを調べる継続  */
11  int     candidate;                      /*  素数の候補を示すフラグ    */
12
13  int     is_prime(int n)
14  {
15          cont    temp;
16          if (temp = c_get())
17                  p_cont[n] = temp;       /*  prime[n]で割るための継続  */
18          if (number % prime[n]) {
19                  c_cont(p_cont[n-1]);    /*  整除できない時            */
20          } else {
21                  candidate = 0;          /*  整除できた時              */
22                  c_cont(p_cont[0]);
23          }
24  }
25
26  void    c_main()
27  {
28          cont    temp;
29          n_prime = 1;
30          number = 2;
31          candidate = 1;
32          if (temp = c_get())
33                  p_cont[0] = temp;       /*  繰返し用の継続            */
34          if (candidate) {
35                  prime[n_prime] = number++;      /*  素数として登録    */
36                  if (n_prime < N_PRIME)
37                          is_prime(n_prime++);    /*  次の数を調べる    */
38          } else {
39                  number += 1;
40                  candidate = 1;
41                  c_cont(p_cont[n_prime-1]);      /*  次の数を調べる    */
42          }
43          for (n_prime=1; n_prime<=N_PRIME; n_prime+=1)
44                  printf("%d,", prime[n_prime]);  /*  求めた素数の表示  */
45          printf(" .....¥n");
46  }
```

ば関数is_primeを呼び出して、0であれば既に取得した継続を実行して（41行目）、処理を続行する。

関数is_primeは、引数で指定されるn番目の素数prime[n]でnumberが割り切れるかどうかを判定する。割り切れた場合は、その数は素数ではないのでcandidateを0にし継続p_cont[0]を実行する。割り切れなかった場合は1つ前の素数で割り切れるかどうかを調べるために既に用意してある継続p_cont[n-1]を実行する。なお、この処理を繰り返すために、is_primeの先頭で、新しい継続をp_cont[n]に取得する。

もし、numberが素数だった時はcandidateの値は1のままでp_cont[0]が実行される（19行目）。一方、素数でなかった時はcandidateの値を0に設定（21行目）してからp_cont[0]が実行される（22行目）。

継続p_cont[0]の実行は32行目のif文から始まる。この時c_getの戻り値は0なので、引き続き34行目のif文が実行される。candidateが1のまま制御が戻ってきた場合は、numberの値は素数なのでそれをprimeに登録する。また、求めた個数がまだN_PRIMEに達しない場合は、is_primeを呼び出して（今見つかった素数で割り切れるかどうかを調べる継続を作ると同時に）処理を続行する。なお、numberの値は1増加している（35行目）。

一方、34行目のif文のcandidateの値が0の場合、numberの値は素数ではないので、その値を1増やしcandidateを1に再設定した上で、継続p_cont[n_prime-1]を実行する。これにより、現在求まっている最大の素数でnumberが割り切れるかどうかを調べることになるが、この処理は16行目のc_getから始まる。この部分は既に説明した。

5. 継続ライブラリの評価および問題点

本継続ライブラリは、11行の取込みファイルcont.hと43行のソースコードファイルcont.cから構成される。cont.cには関数c_get（16行）、関数c_cont（11行）、c_mainを呼び出すmain（5行）、それに記号定数の定義などが含まれる。

ライブラリ自体は非常にコンパクトであるが、4章に示したような応用プログラムを書くこともでき、Cのセマンティクスを拡張するためのライブラリとしては有用である。実際、8クィーン問題の全解探索のバックトラッキングに本ライブラリを使うなどの、興味深いプログラムも（ライブラリの利用者である第3者によって）作成されている。

本ライブラリは現在も改良が続けられている。この章では、以下、セマンティクス、実現法に関する問題点を議論し、その後、本ライブラリの移植性について評価する。

5.1. 継続ライブラリのセマンティクス再考

本継続ライブラリは現在も改良を続けており、そのため2章で示した各関数のインタフェースとセマンティクスとは変更されることもあり得る。特に現在、

1）大域変数のセマンティクス
2）c_getの戻り値

に関して代替案を検討している。以下、これらについて簡単に説明する。

［大域変数のセマンティクス］：2.3節で述べたように、本ライブラリでは大域変数の値を継続環境には含めない。これは実用的な見地からの決定であるが、このセマンティクスが最善というわけではない。バックトラックを含むある種のプログラムなどでは、大域変数の値も継続環境に保存すると都合が良い。

［c_getの戻り値］：現在、c_getの戻り値は取得した継続へのポインタ、または0である。しかし、実行を開始した継続にsetjmpのように任意の戻り値を渡すことができれば、プログラムでそれを活用することも可能である。たとえば、c_getの戻り値を使い分けることができれば、4章のプログラムの変数candidateを使う必要はなくなる。

5.2. 継続ライブラリの実現法再考

本継続ライブラリでは、継続をスタック領域

のコピーとして実現している。そのため、継続の保存には比較的大きなメモリ領域を必要とする。また、前節で議論したように大域変数の値を継続環境に含める場合には、継続あたりのメモリ使用量はさらに増加する。

実現という立場からは、継続の概念をどう定義するかによって継続あたりのメモリ使用量も変化する。Cという言語上の継続をどう捉えるかは、もう少し議論する必要があると思われる。

我々の考案した、継続を実行する際にc_contを再帰呼出ししてスタックフレームを追い出すという技法は、移植性確保の観点からは有効である。しかし、処理に時間が掛かるという欠点も合わせ持っている。

この問題点の実践的な解決法には、c_cont内で比較的大きな局所変数（配列など）を宣言するという方法が考えられる。これにより、再帰あたりのスタックフレーム移動量が大きくなり、この部分での処理時間を短縮できる。

5.3. 継続ライブラリの移植性

Cは、ライブラリ関数も言語仕様の一部と考えられるほど、ライブラリに依存した言語である。たとえば言語仕様自体には入出力に関する定義はなく、それらはread, write, getchar, printfといったライブラリ関数で実現されている。C言語としての一貫性、または、応用プログラムの移植性を確保するためには、これらライブラリ関数のインタフェースやセマンティクスをも統一する必要がある。

本継続ライブラリは、ライブラリ記述の移植性を上げることで各種C処理系のライブラリへの組込みを可能とし、同一のインタフェースとセマンティクスとを提供できることを目指している。この試みは十分に成功したと考えられる。

特に、我々が提案し実装した方法、つまり、

1) ライブラリの記述にはアセンブリ言語を使わずCのみを使用したこと；
2) 継続の取得に標準ライブラリ関数のsetjmpを利用したこと；
3) スタック領域の同定に局所変数のアドレスを利用したこと；
4) 関数フレームの移動を関数の再帰呼出しで実現したこと；

といった技法は、継続ライブラリの移植性を高めるのに十分に貢献した。これらの技法なしにはハードウェア、オペレーティングシステム、コンパイラなどと複雑に関係する継続ライブラリに移植性を持たせることはできなかったであろう。

実際、本継続ライブラリは、ハードウェア（SunとSONYの各種計算機（CPU: SPARC, 68030, R3000）やIBM-PC）、オペレーティングシステム（Unix（BSD, SystemV）やMS-DOS）、コンパイラ（cc, gcc, turbo-C）の種類を問わずに稼働することが確認されている。これらの結果からも、移植性向上の試みは十分に成功したと考えられる。

6. おわりに

本報告では、C言語ライブラリとして実現した継続について議論した。我々の実現したライブラリは、ライブラリプログラム自体の移植性を確保しているので、様々なC言語処理系に共通のライブラリ関数として実装することができる。本報告で述べたような技法が、今後のライブラリ実現法の参考になれば幸いである。

参考文献

[Stee78] G. L. Steele Jr. and G. J. Sussman: "The revised report on Scheme, a dialect of Lisp," MIT AI Memo 452, Jan., 1978.

[Tada90] 多田好克, 寺田実：“移植性・拡張性に優れたCのコルーチンライブラリ実現法”, 電子情報通信学会論文誌, Vol. J73-D-I, No. 12, pp. 961-970, 1990.

[Tada92] 多田好克：“機種に依存しない利用者threadsライブラリ”, 情報処理学会プログラミング－言語・基礎・実践－研究会予稿集, 92-PRG-8-22, pp. 171-178, 1992.