# 3R−6　Ｃ言語記述プログラムの移植性について

林　智定　岩田憲和　宮崎義之

ＮＴＴ情報通信処理研究所

## １．はじめに

プログラムの移植では、先ず移植阻害要因を抽出し、次いで移植阻害要因を取り除く作業を行う。従って、移植阻害要因を作り込まないことが移植作業全体の工数削減につながる。現在、著者らは(1) Ｃ言語により記述されたプログラムの移植　(2) 同一インタフェースを意図して開発された異なるＯＳ間の移植　(3) 移植先のコンパイラを使用する　という前提条件の元で、ＯＳのある機能ブロック（約４４ｋｓ）の移植を行っている。

本稿では、実際の移植作業を通して抽出された移植阻害要因、対処法および対処に要した工数等の分析結果について報告すると共に、移植阻害要因のチェック方法について示す。

## ２．移植手順

Ｃ言語で記述されたプログラムの移植は、①コンパイル～リンク環境の構築、②コンパイルの実施、③型の一貫性のチェック、④動作確認　の手順で行われる。以下に各フェーズについて考察する。

(1) コンパイル～リンク環境構築

コンパイルおよびリンクに必要な環境を構築し、コンパイル～リンク手順を明確にするフェーズである。ＵＮＩＸ環境であれば通常は、makefile中にこの手順が明記される。

(2) コンパイルの実施

実際にコンパイルを実施し、コンパイルエラーが取り除かれる。

(3) 型の一貫性のチェック

Ｃ言語で記述されたプログラムでは、変数、定数等に対して様々な型を宣言することができる。また、これらに対しては、型変換や算術演算等、様々な操作が可能である。従って、不適切なコーディングを行えば、エンディアンやパディングルールが異なる環境上に移植した場合、誤動作を起こす恐れがある。本フェーズでは、このような不適切コーディングが取り除かれる。

(4) 動作確認

シミュレータや実環境上で実際に走行させ、動作確認を行う。

上記(3) のフェーズまでを実施すれば、コンパイラの相違をクリアすることが可能となる。

## ３．移植結果

実際の移植を通じて得られた結果を示すと共に、各移植フェーズで抽出された移植阻害要因について、事前チェック方法を考察する。（表１参照）

### ３，１　移植阻害要因と工数

(1) コンパイル～リンク環境構築

本フェーズでは概ね以下の手順でｍａｋｅｆｉｌｅ環境を作成した。

移植対象プログラムをコンパイラが受理できるように、ヘッダファイルの配置を変えた。

今後の移植作業の効率化を考え、ソースも再配置を行った。本フェーズには、約１９６人時を費やした。

(2) コンパイル

コンパイラの制約値に抵触している項目に関しては、対処を行うと共に、構文チェッカーにより構文エラー部分を検出し修正した。本フェーズには、約１８３人時を費やした。尚、代表的なコンパイラの制約値について、いくつかのコンパイラ間で比較した結果を表２に示す。

(3) 型の一貫性のチェック

エンディアン、パディングルール等の相違による誤動作を防止するため、不適切なコーディング部を修正した。本フェーズには、約５０４人時を費やした。

(4) 動作確認

実機およびシミュレータ上で、ステップトレースしながら動作を確認中である。現在までのところ、ほぼ１ルートの動作確認を終了したが、ＯＳの相違による問題点は１点しか抽出されていない。

### ３，２　各フェーズの移植阻害要因の事前チェック

各フェーズ毎に以下の手順を実施すれば、プログラム作成時に移植阻害要因をチェックすることができる。

(1) コンパイル～リンク環境構築

プログラム作成時にｍａｋｅｆｉｌｅを作成しておけば、移植に際しての環境作成は最小限に押さえることができる。

(2) コンパイル

コンパイラの制約値および構文エラーは、ＰＣＣのｌｉｎｔ、ＡＮＳＩ−Ｃチェッカを併用することにより事前にチェックアウトすることが可能である。

(3) 型の一貫性の保証

ｌｉｎｔやＡＮＳＩ−Ｃチェッカは、構文エラーやコンパイラの制約値のチェックをターゲットとしてい

---

A Note on portability issuies.

Tomosada HAYASHI,Norikazu IWATA and Yoshiyuki MIYAZAKI

NTT Communications and Information Processing Laboratories

るため、型の一貫性についてはチェックすることは不可能である。従って、実際には手作業でソースのチェックを行った。現在、著者らは型の一貫性のチェックを可能とするチェッカを検討中である。

(4) 動作確認

現在は、シミュレータや実機上でステップトレースしながら動作確認を実施しているが、効率的に動作を確認するプログラムについては、今後の課題である。

## 4．考察

今回の移植作業を通して、確認できたことをまとめる。

(1) 従来よりプログラムの移植に際しては、ＯＳインタフェースが同一であり、移植対象プログラムが高級言語で記述されていれば、ＯＳの相違（実行環境の相違）への整合に殆どの移植工数が費やされると言われてきた。しかし、今回の移植作業の結果、ＯＳインタフェースが同一であり、かつ高級言語で記述されていても、言語処理系の相違への整合に非常に多くの工数（８８３人時）が必要であることが明かとなった。

従って、この部分に要する工数を削減することは、移植作業全体からみても非常に効果は大きいと思われる。

(2) (1) に関連して、コンパイラの制約値や構文エラーに起因する移植阻害要因は、ｌｉｎｔやＡＮＳＩ−Ｃチェッカと言った、既存のツールを用いることによりかなりチェックが可能である。また、型の一貫性の保証に関しても、チェッカを作成することにより充分可能である。従って、これらのチェッカ類をプログラム開発段階で利用すれば、移植阻害要因の混入を防止でき、移植性の高いプログラムの実現が可能となる。

## 5．おわりに

著者らは引続き移植作業を進めており、本検討結果も含めて移植作業を通して得られた結果を、今後開発されるプログラムにおいて移植性確保のための指針として行く予定である。

[参考文献]

[１] 林ほか：移植阻害要因要因の定量化手法の検討，情報処理学会第４１回全国大会予稿（1990）

表１　各移植フェーズの概要

| 移植ﾌｪｰｽﾞ | 移植阻害要因あるいは作業項目 | 工数 | 修正規模 | チェックあるいは自動化方法 |
|---|---|---|---|---|
| ｺﾝﾊﾟｲﾙ〜ﾘﾝｸ環境の構築 | ﾍｯﾀﾞﾌｧｲﾙの再配置<br>ｿｰｽの再配置 | 196人時 | | makefile(UNIX)で対処可能。 |
| ｺﾝﾊﾟｲﾙ | ｺﾝﾊﾟｲﾗの制約値の回避<br>・ｿｰｽ１行中の最大文字数<br>・ﾏｸﾛ呼び出し部の複数行記述<br>・ｺﾝﾊﾟｲﾙ単位中のﾏｸﾛ数 | 106人時 | 377ｽﾃｯﾌﾟ | ANSI_Cﾁｪｯｶｰ |
| | 構文ｴﾗｰの修正<br>・引数宣言列中に型定義が混在<br>・関数の引数の個数の不一致<br>・ﾚｼﾞｽﾀ変数のｱﾄﾞﾚｽ参照<br>・関数の返り値が未使用<br>・ﾏﾙﾁﾊﾞｲﾄｷｬﾗｸﾀの使用<br>・ﾏｸﾛ定義内での変数定義のｽｺｰﾌﾟ | 77人時 | 100ｽﾃｯﾌﾟ | ANSI-Cﾁｪｯｶｰ、lint |
| 型の一貫性のﾁｪｯｸ | ・実態参照の一貫性<br>・関数の引数の型の不一致<br>・基本整数型が３２ｂを前提<br>・NULLの扱い | 504人時 | 4kｽﾃｯﾌﾟ | 新規にﾁｪｯｶｰを開発 |
| 動作確認 | ｼﾐｭﾚｰﾀあるいは実環境での試験<br>・論理空間確保法の相違 | | 1ｽﾃｯﾌﾟ | 今後の検討課題 |

表２　代表的なコンパイラの制約値

| 項　目 | 移植元 | 移植先 | その他１ | その他２ | ANSI-C |
|---|---|---|---|---|---|
| ｉｆ文のﾈｽﾄ数 | １６ | ３２ | 8 | ３２ | 8 |
| 外部識別子の有効先頭文字数 | ３１ | | ３１ | ３１ | 6 |
| ｺﾝﾊﾟｲﾙ単位中に定義可能なﾏｸﾛ数 | ４０９６ | １０２４ | ４０９６ | ４０９６ | １０２４ |
| １つのｿｰｽ行に許される最大文字数 | ４０００ | ５１１ | ４０９３ | ４０９３ | ５０９ |
| #includeﾌｧｲﾙのﾈｽﾄ数 | １６ | 6 | １２ | １２ | 8 |

（注）　　は実際の移植作業において問題となった項目を示す。また、ANSI-Cは最低保証値を示す。