(19)日本国特許庁(JP)　(12)公開特許公報(A)　(11)特許出願公開番号
特開2014-59692
(P2014-59692A)
(43)公開日　平成26年4月3日(2014.4.3)

(51)Int.Cl.		F I			テーマコード（参考）
G06F 9/44	(2006.01)	G06F	9/06	620A	5B376

審査請求 未請求　請求項の数 6　ＯＬ　（全 13 頁）

(21)出願番号　特願2012-203981 (P2012-203981)
(22)出願日　平成24年9月18日 (2012.9.18)

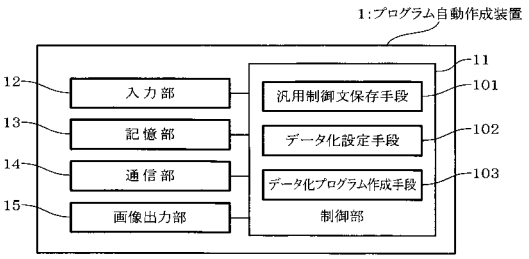
（特許庁注：以下のものは登録商標）
１．ＪＡＶＡ

(71)出願人　000006150
京セラドキュメントソリューションズ株式会社
大阪府大阪市中央区玉造１丁目２番２８号
(74)代理人　100086759
弁理士　渡辺　喜平
(74)代理人　100109128
弁理士　岡野　功
(72)発明者　上野　敏昭
大阪府大阪市中央区玉造１丁目２番２８号　京セラドキュメントソリューションズ株式会社内

Ｆターム(参考)　5B376 BC31　BC42　BC74

(54)【発明の名称】プログラム自動作成プログラム、プログラム自動作成方法及びプログラム自動作成装置

(57)【要約】
【課題】変更後の検証が不要で可読性及び保守性に優れたコンピュータプログラムを容易に作成する。
【解決手段】　コンピュータプログラムを自動的に作成するためのプログラム自動作成プログラムであって、コンピュータを、構造化定理にもとづく論理構造である順次、分岐、反復の制御文を汎用化した汎用制御文を保存する汎用制御文保存手段１０１、順次、分岐又は反復のうちのいずれかの論理構造と、設定した論理構造にしたがって実行させる関数との対応付けを設定するデータ化設定手段１０２、及び、設定した対応付けをデータとして定義するとともに、このデータを参照することによって、その論理構造に係る汎用制御文にしたがって対応した関数が実行されるコンピュータプログラムを作成するデータ化プログラム作成手段１０３、として機能させる構成とする。
【選択図】図１



【特許請求の範囲】
【請求項１】
コンピュータプログラムを自動的に作成するためのプログラム自動作成プログラムであって、
コンピュータを、
構造化定理にもとづく論理構造である順次、分岐、反復の制御文を汎用化した汎用制御文を保存する汎用制御文保存手段、
順次、分岐又は反復のうちのいずれかの論理構造と、当該論理構造にしたがって実行させる関数との対応付けを設定するデータ化設定手段、及び
前記対応付けをデータとして定義するとともに、当該データを参照することによって、その論理構造に係る前記汎用制御文にしたがって対応した関数が実行されるコンピュータプログラムを作成するデータ化プログラム作成手段、として機能させる
ことを特徴とするプログラム自動作成プログラム。
【請求項２】
前記データ化設定手段が、
順次を示す識別情報と順次実行させる関数の識別情報との対応付けを設定した場合に、
前記データ化プログラム作成手段に、
前記対応付けをデータとして定義するとともに、当該データを参照することによって、順次に係る前記汎用制御文にしたがって対応する関数を順次実行するコンピュータプログラムを作成させる
ことを特徴とする請求項１記載のプログラム自動作成プログラム。
【請求項３】
前記データ化設定手段が、
分岐を示す識別情報、及び、分岐の条件と当該分岐の条件に応じて実行させる一又は二以上の関数からなる対応付けを設定した場合に、
前記データ化プログラム作成手段に、
前記対応付けをデータとして定義するとともに、当該データを参照することによって、前記分岐の条件及び分岐に係る前記汎用制御文にしたがって対応する関数を実行するコンピュータプログラムを作成させる
ことを特徴とする請求項１又は２記載のプログラム自動作成プログラム。
【請求項４】
前記データ化設定手段が、
反復を示す識別情報、及び、反復の条件と当該反復の条件に応じて繰り返し実行させる関数からなる対応付けを設定した場合に、
前記データ化プログラム作成手段に、
前記対応付けをデータとして定義するとともに、当該データを参照することによって、前記反復の条件及び反復に係る前記汎用制御文にしたがって対応する関数を繰り返し実行するコンピュータプログラムを作成させる
ことを特徴とする請求項１～３のいずれか一項記載のプログラム自動作成プログラム。
【請求項５】
コンピュータプログラムを自動的に作成するためのプログラム自動作成方法であって、
構造化定理にもとづく論理構造である順次、分岐、反復の制御文を汎用化した汎用制御文を保存する工程と、
順次、分岐又は反復のうちのいずれかの論理構造と、当該論理構造にしたがって実行させる関数との対応付けを設定する工程と、
前記対応付けをデータとして定義するとともに、当該データを参照することによって、その論理構造に係る前記汎用制御文にしたがって対応した関数が実行されるコンピュータプログラムを作成する工程と、を有する
ことを特徴とするプログラム自動作成方法。
【請求項６】

コンピュータプログラムを自動的に作成するプログラム自動作成装置であって、
構造化定理にもとづく論理構造である順次、分岐、反復の制御文を汎用化した汎用制御文を保存する汎用制御文保存手段と、
順次、分岐又は反復のうちのいずれかの論理構造と、当該論理構造にしたがって実行させる関数との対応付けを設定するデータ化設定手段と、
前記対応付けをデータとして定義するとともに、当該データを参照することによって、その論理構造に係る前記汎用制御文にしたがって対応した関数が実行されるコンピュータプログラムを作成するデータ化プログラム作成手段と、を備えた
ことを特徴とするプログラム自動作成装置。

【発明の詳細な説明】

【技術分野】

【０００１】

本発明は、コンピュータプログラムを自動的に作成するためのプログラム自動作成プログラム、プログラム自動作成方法及びプログラム自動作成装置に関し、より詳しくは、論理構造に係る制御文の一部をデータ化したコンピュータプログラムを自動的に作成することが可能なプログラム自動作成プログラム、プログラム自動作成方法及びプログラム自動作成装置に関する。

【背景技術】

【０００２】

従来からコンピュータプログラム（以下、プログラムという）の大規模化や複雑化に鑑み、Ｃ、Ｃ++、ｊａｖａ、Ｐａｓｃａｌ等のプログラム言語を用いた構造化定理にもとづく構造化プログラミングが推奨されている（例えば、特許文献１参照）。
構造化定理によれば、順次、分岐、反復の３つの基本的な論理構造だけでプログラムを構成するため、可読性及び保守性に優れたプログラムを実現することができる。

【先行技術文献】

【特許文献】

【０００３】

【特許文献１】特許第３６０２４１６号公報

【発明の概要】

【発明が解決しようとする課題】

【０００４】

しかしながら、構造化プログラミングはプログラムを作成するうえでの規範や作法であり、強制するものではないため、規範等が維持されない場合には上述の効果は得られない。
また、規範等が維持される場合であっても、各論理構造の制御文の構成はプログラマーに委ねられているため、プログラマーによっては制御文の構成が複雑になり、プログラムの作成が困難化し、可読性を低下させる問題が生じていた。
さらに、制御文はプログラムであり、内容を変更する場合には、その多少に拘わらず、正当性を検証する手間が生ずる。
図１０～１２は、従来の順次、分岐、反復の制御文の一例を示す図である。
例えば、図１０の制御文に別の関数（例えば、関数ｚ）を順次処理に追加する場合、図１１の制御文の分岐条件（ret==5：戻り値が５に等しいかどうか）を変更する（例えば、ret>10に変更する）場合、図１２の制御文の反復条件（x=1：初期値が１、ｘ+=2：刻み値が２）を変更する（例えば、x=0、x+=3に変更する）場合等、軽微な変更であってもプログラム自体を変更しているため、変更後のプログラムが全体として正しいかどうかを確認する煩わしさがあった。

【０００５】

本発明は、上記の事情に鑑みてなされたものであり、構造化定理にもとづく論理構造である順次、分岐、反復に関する制御文の一部をデータ化することで、変更後の検証が不要で、かつ、可読性及び保守性に優れたコンピュータプログラムを自動的に作成することが

できるプログラム自動作成プログラム、プログラム自動作成方法、及び、プログラム自動作成装置の提供を目的とする。

【課題を解決するための手段】

【０００６】

上記目的を達成するため本発明のプログラム自動作成プログラムは、コンピュータプログラムを自動的に作成するためのプログラム自動作成プログラムであって、コンピュータを、構造化定理にもとづく論理構造である順次、分岐、反復の制御文を汎用化した汎用制御文を保存する汎用制御文保存手段、順次、分岐又は反復のうちのいずれかの論理構造と、この論理構造にしたがって実行させる関数との対応付けを設定するデータ化設定手段、及び、設定した対応付けをデータとして定義するとともに、このデータを参照することによって、その論理構造に係る汎用制御文にしたがって対応した関数が実行されるコンピュータプログラムを作成するデータ化プログラム作成手段、として機能させるようにしている。

【０００７】

また、本発明のプログラム自動作成方法は、コンピュータプログラムを自動的に作成するためのプログラム自動作成方法であって、構造化定理にもとづく論理構造である順次、分岐、反復の制御文を汎用化した汎用制御文を保存する工程と、順次、分岐又は反復のうちのいずれかの論理構造と、この論理構造にしたがって実行させる関数との対応付けを設定する工程と、設定した対応付けをデータとして定義するとともに、このデータを参照することによって、その論理構造に係る汎用制御文にしたがって対応した関数が実行されるコンピュータプログラムを作成する工程と、を有するようにしている。

【０００８】

また、本発明のプログラム自動作成装置は、コンピュータプログラムを自動的に作成するプログラム自動作成装置であって、構造化定理にもとづく論理構造である順次、分岐、反復の制御文を汎用化した汎用制御文を保存する汎用制御文保存手段と、順次、分岐又は反復のうちのいずれかの論理構造と、この論理構造にしたがって実行させる関数との対応付けを設定するデータ化設定手段と、設定した対応付けをデータとして定義するとともに、このデータを参照することによって、その論理構造に係る前記制御文にしたがって対応した関数が実行されるコンピュータプログラムを作成するデータ化プログラム作成手段と、を備えるようにしている。

【発明の効果】

【０００９】

本発明によれば、変更後の検証が不要で可読性及び保守性に優れたコンピュータプログラムを自動的に作成することができる。

【図面の簡単な説明】

【００１０】

【図１】本発明の実施形態に係るプログラム自動作成装置の構成及び制御部の機能ブロックを示す図である。

【図２】順次処理の汎用制御文を引用して作成されたプログラムを示す図である。

【図３】分岐処理の汎用制御文を引用して作成されたプログラムを示す図である。

【図４】反復処理の汎用制御文を引用して作成されたプログラムを示す図である。

【図５】プログラム作成時のメニュー画面の一例を示す図である。

【図６】順次処理のデータ化設定画面の一例を示す図である。

【図７】分岐処理のデータ化設定画面の一例を示す図である。

【図８】反復処理のデータ化設定画面の一例を示す図である。

【図９】本発明のプログラム自動作成方法を示すフローチャートである。

【図１０】従来の順次処理の制御文を示す図である。

【図１１】従来の分岐処理の制御文を示す図である。

【図１２】従来の反復処理の制御文を示す図である。

【発明を実施するための形態】

【００１１】
本発明の一実施形態に係るプログラム自動作成プログラム及びプログラム自動作成装置について説明する。
図１は、本発明の実施形態に係るプログラム自動作成装置の構成及び制御部の機能ブロックを示す図である。
図１に示すように、本実施形態のプログラム自動作成装置１は、キーボード等の入力デバイスと接続され文字、数字、記号などのデータ等を入力する入力部１２、アプリケーションプログラムや種々のファイルデータを保存する記憶部１３、通信回線と接続され外部装置との間でファイルデータ等の送受信を行う通信部１４、液晶モニター等のディスプレイ装置と接続され、操作画面や演算結果等を出力する画像出力部１５、及び、ＣＰＵ，ＲＯＭ，ＲＡＭ等を備えコンピュータとして動作する制御部１１を備えた情報処理装置である。
本実施形態のプログラム自動作成装置１は、プログラム自動作成プログラムの命令によりコンピュータで実行される処理，手段，機能によって実現され、プログラム自動作成プログラムが、ＣＰＵに読み込まれることにより、構成各部に指令が送られ、制御部１１として以下の動作を行う。
【００１２】
制御部１１は、プログラムの作成時において、キーボード等から入力された文字、数字、記号にもとづきソースプログラムを作成・保存する制御動作や、Ｃ言語などのプログラム言語のルールにもとづいてソースプログラムを翻訳して実行可能なプログラムを作成・保存する制御動作等、プログラムを作成、検証するための一般的な制御動作の他、以下の特徴的な動作を行う。
すなわち、制御部１１は、構造化定理に係る順次、分岐、反復ごとに汎用化した制御文を予め保存する動作や、実行すべき関数や必要な条件などをデータ化しつつ汎用制御文を引用してプログラムを作成する動作等を行う。
このため、本実施形態の制御部１１は、図１に示すように、汎用制御文保存手段１０１と、データ化設定手段１０２と、データ化プログラム作成手段１０３と、を機能ブロックとして備える。
このように、本実施形態のプログラム自動作成装置１における各構成手段は、プログラム自動作成プログラムとコンピュータとが協働した具体的手段によって実現される。
これによって、前記各動作は、ソフトウェアであるプログラム自動作成プログラムとハードウェア資源であるコンピュータの各構成手段とが協働することにより実現される。
【００１３】
汎用制御文保存手段１０１は、構造化定理にもとづく論理構造である順次、分岐、反復の制御文を汎用化した汎用制御文を保存する。
ここで、汎用制御文は、論理構造の従来の制御文を一般化・汎用化した命令文である。実行させる関数や諸条件に応じ、その都度プログラマーが個別に作成していた制御文を一般化することによって、データ化設定にもとづく様々なプログラムの作成において汎用的に引用できるようにしたものである。
具体的には、プログラマーが、予めキーボード等を介し作成したテキストデータ等で構成される汎用制御文を記憶部１３に保存し、または、このようなデータを外部の機器又は記憶装置からダウンロード又は入力して記憶部１３に保存する。
なお、「関数」とは、同じ目的が決まった小さなプログラムのまとまりをいい、「main()関数」をメインルーチンとする場合のサブルーチンに相当する。
以下、順次、分岐、反復の汎用制御文についてそれぞれ説明する。
【００１４】
図２は、順次処理の汎用制御文を引用して作成したプログラムを示す図であり、この中のＡ３部分（主にＡ３１部分）が、順次処理の汎用制御文を示す。
このような命令文によれば、for文における「i++」のインクリメント識別子によってiが１ずつ加算されながら、Ａ２１部分に示されるテーブルの[i][1]に指定されたfp（関数

ポインタ）に関する関数を実行させることができる。
このため、Ａ３１部分の命令文をもって、Ａ２１部分のテーブルにおいて任意の関数の実行を命令する順次処理の一般的なプログラム（汎用制御文）とすることができる。
【００１５】
図３は、分岐処理の汎用制御文を引用して作成したプログラムを示す図であり、この中のＢ３部分（主にＢ３１部分）が、分岐処理の汎用制御文を示す。なお、ここでは、ある関数によって求めた値が分岐条件（ret=*）を満たすかどうかによって、２つの関数の実行を切り分ける分岐のケースを例示している。
このような命令文によれば、まず、「BUNKI1」において、Ｂ２１部分に示されるテーブルの[i][2]に指定されたfpに関する関数ｄが実行され、その戻り値（ret）が求められる。「BUNKI1」の処理後、iは１加算される。
そして、「BUNKI2」において、「BUNKI1」で求めた戻り値がＢ２１部分に示されるテーブルの[i][1]に指定された判定値と等しければ、テーブルの[i][2]に指定されたfpに関する関数ｅを実行させ、「BUNKI1」で求めた戻り値が同テーブルの[i][1]に指定された判定値と等しくなければ、テーブルの[i][3]に指定されたfpに関する関数ｆを実行させることができる。
なお、「判定値と等しいか否か（==）」に限らず、「以上（>=）」，「以下（<=）」，「より大きい（>）」，「より小さい（<）」など、任意の判定条件を適用することができる。
このため、Ｂ３１部分の命令文をもって、Ｂ２１部分のテーブルにおいて任意の分岐条件にもとづき任意の関数の実行を命令する分岐処理の一般的なプログラム（汎用制御文）とすることができる。
【００１６】
図４は、反復処理の汎用制御文を引用して作成したプログラムを示す図であり、この中のＣ３部分（主にＣ３１部分）が、反復処理の汎用制御文を示す。
このような命令文によれば、初期値に相当する初期化式（x=table[i][1]）によって、変数ｘの初期値がテーブルの[i][1]に指定された値としている。そして、判定値に相当する判定条件式（x<table[i][1]）及び刻み値に相当する更新式（x=x+table[i][3]）によって、ｘがＣ２１部分に示されるテーブルの[i][2]に指定された値より小さい場合において、xがテーブルの[i][3]に指定された値が加算されるたびに、テーブルの[i][4]に指定されたfpに関する関数ｇを実行させるようにしている。
なお、初期化式、判定条件式、更新式などの反復条件は、任意の式を適用することができる。
このため、Ｃ３１部分の命令文をもって、Ｃ２１部分のテーブルにおいて任意の反復条件(初期化式、判定条件式、更新式)にもとづき任意の関数の実行を命令する反復処理の一般的なプログラム（汎用制御文）とすることができる。
【００１７】
データ化設定手段１０２は、順次、分岐又は反復のうちのいずれかの論理構造と、この論理構造にしたがって実行させる関数との対応付けを設定する。
図５は、プログラム作成時のメニュー画面の一例を示す図である。
また、図６～図８は、論理構造を設定する際に表示されるデータ化設定画面を示す図である。
画像出力部１５は、これらの画面を図示しないディスプレイ装置の表示画面Ｐに表示する。
例えば、図５に示すメニュー画面において、順次処理キーＫ１が選択されると図６の画面を表示し、分岐処理キーＫ２が選択されると図７の画面を表示し、反復処理キーＫ３が選択されると図８の画面を表示する。
以下、順次、分岐、反復のデータ化設定について図６～図８を参照しながら説明する。
【００１８】
順次処理のデータ化設定においては、順次を示す識別情報（識別子）と、順次実行させ

る関数の識別情報（関数のアドレス）との対応付けを設定する。
図６は、順次処理のデータ化設定画面を示す図である。
図６に示すように、順次処理のデータ化設定画面としては、識別子の入力欄と関数ポインタの入力欄とが対応したテーブルＴａが表示されるため、ユーザーはこれらの入力欄に必要事項を入力する。
例えば、識別子の入力欄に「順次」を示す識別子として任意の文字列（「JUNJI」）を入力し、関数ポインタの入力欄に関数のアドレス（関数ポインタ）を入力し、これらの入力を対象の関数の数分行う。
【００１９】
分岐処理のデータ化設定においては、分岐を示す識別情報（識別子）と、分岐条件とその分岐条件に応じて実行させる一又は二以上の関数の識別情報（関数ポインタ）との対応付けを設定する。
図７は、分岐処理のデータ化設定画面を示す図である。
図７に示すように、分岐処理のデータ化設定画面としては、識別子の入力欄、判定値の入力欄及びTRUE・FALSEの入力欄（関数ポインタの入力欄）が対応したテーブルＴｂが表示されるため、ユーザーはこれらの入力欄に必要事項を分岐条件として入力する。
例えば、図３に例示したタイプの分岐処理に対応する場合には、テーブルＴｂの１段目において、識別子の入力欄に「分岐」を示す識別子として任意の文字列（「BUNKI1」）を入力し、TRUEの入力欄に対象の戻り値を出力する関数の関数ポインタを入力し、判定値及びFALSEの入力欄にdummyの情報を入力する。そして、テーブルＴｂの２段目においては、識別子の入力欄に「分岐」を示す識別子として１段目とは異なる文字列（「BUNKI2」）を入力し、判定値に5を入力し、TRUEの入力欄に関数ｅの関数ポインタを入力し、FALSEの入力欄に関数ｆの関数ポインタを入力する。
【００２０】
反復処理のデータ化設定においては、反復を示す識別子（識別情報）と、反復条件とその反復条件に応じて繰り返し実行させる関数の識別情報（関数ポインタ）との対応付けを設定する。
図８は、反復処理のデータ化設定画面を示す図である。
図８に示すように、反復処理のデータ化設定画面としては、識別子の入力欄、初期値の入力欄、判定値の入力欄、刻み値の入力欄及び関数ポインタの入力欄が対応したテーブルＴｃが表示されるため、ユーザーはこれらの入力欄に必要事項を反復条件として入力する。
例えば、識別子の入力欄に「反復」を示す識別子として任意の文字列（「HANPUKU」）を入力し、初期値の入力欄に「1」を入力し、判定値の入力欄に「10」を入力し、刻み値の入力欄に「2」を入力し、関数ポインタの入力欄に関数ｇの関数ポインタを入力する。
【００２１】
データ化プログラム作成手段１０３は、設定された論理構造と関数の識別情報との対応付けをデータとして定義するとともに、このデータを参照することによって、その論理構造に係る汎用制御文にしたがって対応した関数が実行されるプログラムを作成する。
以下、順次、分岐、反復に関するプログラムの作成についてそれぞれ説明する。
【００２２】
順次処理に関するプログラムの作成においては、データ化設定手段１０２において設定された順次を示す識別子と関数ポインタとの対応付けをデータとして定義するとともに、このデータを参照することによって、順次に係る汎用制御文にしたがって対応する関数を順次実行するプログラムを作成する。
図２は、順次処理の汎用制御文を引用するとともに、図６に示すデータ化設定にもとづき作成されたプログラムを示す図である。以下、Ａ１、Ａ２及びＡ３のブロックごとにプログラムの内容を説明する。
【００２３】
Ａ１では、データ化設定に応じてテーブルのサイズを指定する命令文が作成される。図

６に示すように、テーブルＴａにおいて３段にわたり入力が行われている場合には、「#define SIZE 3」と記述される。

また、テーブルの型宣言を示す命令文が作成される。具体的には、テーブル「fp_tbl」は、「shikibetsushi」と「*fp」を構成要素とする構造体「struct」でありデータのまとまりであることが定義付けされる。

【００２４】

Ａ２では、データ化設定値に応じ、構造体（Ａ２１部分のテーブルデータ）の実体を宣言する命令文が作成される。

具体的には、「i」が整数変数であることが宣言される。また、図６に示すデータ化設定に応じ、「func_a」、「func_b」、「func_c」がそれぞれ整数変数であること、及び、「fp_tbl」の構成データが、{JUNJI,&func_a},{JUNJI,&func_b},{JUNJI,&func_c}であることが宣言される。

【００２５】

Ａ３では、宣言した構造体を参照することによって、その構造体に示された関数を順次処理を実行する命令文が作成される。

具体的には、「for～switch～case JUNJI:～case BUNKI:～case HANPUKU～」における「case JUNJI:～」（Ａ３１部分）において、順次処理に関する制御文を汎用化した命令文が作成される。

この場合、データ化プログラム作成手段１０３は、汎用制御文保存手段１０１により予め記憶部１３に保存された汎用制御文の中から順次に関する汎用制御文を取り出して引用する。

【００２６】

このようにすると、順次処理に関するプログラムを自動的に作成することができるだけでなく、作成したプログラムを容易に変更することができる。

例えば、関数ｚを関数ｃの次に実行させる関数として加える場合には、図６に示すデータ化設定画面において、識別子「JUNJI」と関数ｚの関数ポインタとの対応付けを加えるだけでよく、関数ｃを順次処理から外す場合には、関数ｃに関する設定データを削除するだけでよい。

また、関数ａ～ｃの順序を入れ替えたり、関数ａ～ｃの一部又は全部を他の関数と置き換えたりする場合も、設定データを変更するだけでできる。

また、データ変更のみであり、プログラムの変更は伴わないため、作成当初のプログラムの検証が完了していれば、変更後の検証を省くことができる。

【００２７】

分岐処理に関するプログラムの作成においては、データ化設定手段１０２において設定された分岐を示す識別子と関数ポインタと分岐条件との対応付けをデータとして定義するとともに、このデータを参照することによって、分岐条件及び分岐に係る汎用制御文にしたがって対応する関数を実行するプログラムを作成する。

図３は、分岐処理の汎用制御文を引用するとともに、図７のデータ化設定にもとづき作成されたプログラムを示す図である。以下、Ｂ１、Ｂ２及びＢ３のブロックごとにプログラムの内容を説明する。

【００２８】

Ｂ１では、データ化設定に応じてテーブルのサイズを指定する命令文が作成される。図７に示すように、テーブルＴｂにおいて２段にわたり入力が行われている場合には、「#define SIZE 2」と記述される。

また、テーブルの型宣言を示す命令文が作成される。具体的には、テーブル「fp_tbl」は、「shikibetsushi」、「hanteichi」、「*fp」を構成要素とする構造体「struct」でありデータのまとまりであることが定義付けされる。

【００２９】

Ｂ２では、データ化設定値に応じ、構造体（Ｂ２１部分のテーブルデータ）の実体を宣言する命令文が作成される。

具体的には、「i」が整数変数であることが宣言される。また、図７に示すデータ化設定に応じ、「ret」、「func_d」、「func_e」、「func_f」がそれぞれ整数変数であること、及び、「fp_tbl」の構成データが、{BUNKI1,DUMMY_VALUE,&func_d,DUMMY_FUNC},{BUNKI2,5,&func_e,&func_f}であることが宣言される。

【００３０】

Ｂ３では、宣言した構造体を参照することによって、その構造体に示された分岐条件にしたがって分岐処理を実行する命令文が作成される。

具体的には、「for～switch～case JUNJI:～case BUNKI:～case HANPUKU～」における「case BUNKI1」,「case BUNKI2」（Ｂ３１部分）において、分岐処理に関する制御文を汎用化した命令文が作成される。

この場合、データ化プログラム作成手段１０３は、汎用制御文保存手段１０１により予め記憶部１３に保存された汎用制御文の中から分岐に関する汎用制御文を取り出して引用する。

【００３１】

このようにすると、分岐処理に関するプログラムを自動的に作成することができるだけでなく、作成したプログラムを容易に変更することができる。

例えば、分岐条件を変更する場合には、図７に示すデータ化設定画面において、判定値を変更するだけで足りる。また、分岐処理を追加する場合には、重複しない識別子（「BUNKI3」「BUNKI4」等）、判定値及び関数の対応付けを追加するだけでよい。

また、関数ｄ～ｆを相互に入れ替えたり、関数ｄ～ｆの一部又は全部を他の関数と置き換えたりする場合も、設定データを変更するだけでできる。

また、順次処理の場合と同様、変更後の検証を省くことができる。

【００３２】

反復処理に関するプログラムの作成においては、データ化設定手段１０２において設定された反復を示す識別子と関数ポインタと反復の条件との対応付けをデータとして定義するとともに、このデータを参照することによって、反復条件及び反復に係る汎用制御文にしたがって対応する関数を繰り返し実行するプログラムを作成する。

図４は、反復処理の汎用制御文を引用するとともに、図８のデータ化設定にもとづき作成されたプログラムを示す図である。以下、Ｃ１、Ｃ２及びＣ３のブロックごとにプログラムの内容を説明する。

【００３３】

Ｃ１では、データ化設定に応じてテーブルのサイズを指定する命令文が作成される。図８に示すように、テーブルＴｃの１段のみに入力が行われている場合には、「#define SIZE 1」と記述される。

また、テーブルの型宣言を示す命令文が作成される。具体的には、テーブル「fp_tbl」は、「shikibetsushi」、「shokichi」、「hanteichi」、「kizamichi」、「*fp」を構成要素とする構造体「struct」でありデータのまとまりであることが定義付けされる。

【００３４】

Ｃ２では、データ化設定値に応じ、構造体（Ｃ２１部分のテーブルデータ）の実体を宣言する命令文が作成される。

具体的には、「i」が整数変数であることが宣言される。また、図８に示すデータ化設定に応じ、「func_g」が整数変数であること、及び、「fp_tbl」の構成データが、{HANPUKU,1,10,2,&func_g}であることが宣言される。

【００３５】

Ｃ３では、宣言した構造体を参照することによって、その構造体に示された反復条件にしたがって反復処理を実行する命令文が作成される。

具体的には、「for～switch～case JUNJI:～case BUNKI:～case HANPUKU～」における「case HANPUKU」（Ｃ３１部分）において、反復処理に関する制御文を汎用化した命令文が作成される。

この場合、データ化プログラム作成手段１０３は、汎用制御文保存手段１０１により予

め記憶部１３に保存された汎用制御文の中から反復に関する汎用制御文を取り出して引用する。

【００３６】

このようにすると、反復処理に関するプログラムを自動的に作成することができるだけでなく、作成したプログラムを容易に変更することができる。

例えば、反復条件を変更する場合には、図８に示すデータ化設定画面において、判定値、初期値、刻み値を変更するだけでよい。また、関数ｇを他の関数に変更したい場合には、その関数ポインタに置き換えればよい。また、反復処理を追加する場合には、識別子「HANPUKU」と所望の関数の関数ポインタとの対応付けを加えるだけでよい。

また、順次処理、分岐処理の場合と同様、変更後の検証を省くことができる。

【００３７】

ところで、図２のＡ３部分、図３のＢ３部分、及び、図４のＣ３部分に示すように、「for～switch～case　JUNJI:～case　BUNKI:～case　HANPUKU～」の命令文は、データ化プログラムの作成に際し、順次、分岐、反復の別に拘わらず共通に用いるようにしている。

このようにすると、構造化定理にもとづく順次、分岐、反復に関する汎用制御文を任意に組み合わせたプログラムを作成することができる。

例えば、図６に示すデータ化設定に続けて、図７に示すデータ化設定をした場合には、図２のＡ２部分に図３のＢ２部分が追加され、Ａ３部分の「case　BUNKI:～break」がＢ３１部分に置き換わったプログラムが作成される。

このため、所望のプログラムを構造化プログラミングの作法にしたがって適切に作成することができる。

【００３８】

このように、本実施形態のプログラム自動作成プログラム及びプログラム自動作成装置１によれば、汎用制御文保存手段１０１が、順次、分岐、反復ごとに汎用化した制御文を予め保存するようにしている。

また、データ化設定手段１０２が、順次、分岐又は反復のうちのいずれかの論理構造を設定するとともに、その論理構造にしたがって実行させる関数を設定し、データ化プログラム作成手段１０３が、設定された論理構造と関数との対応付けをデータとして定義するとともに、このデータを参照することによって、その論理構造に係る汎用制御文にしたがって対応した関数が実行されるプログラムを作成するようにしている。

【００３９】

このため、順次、分岐、反復の３つの論理構造のみによって構成すべきとする構造化定理にもとづくプログラムを、誰もが容易に作成することができる。

また、従来のように、プログラム構成が複雑化することがなく、プログラムの作成に時間を要したりプログラムの可読性を悪化させる問題を解消することができる。

また、実行すべき関数や関数を実行するうえでの必要な条件等、従来プログラムであった制御文の一部をデータ化してプログラムを構成するようにしている。

このため、簡単なデータ設定だけで所望のプログラムを作成することができる。

また、プログラムの作成後においては、データ設定を変更するだけで所望のプログラムに変更することができる。しかも、この場合、プログラムの構文自体は変更しないため、従来変更後に生じていたプログラムの正当性を検証する手間を無くすことができる。

【００４０】

（プログラム自動作成方法）

次に、本発明のプログラム自動作成方法について説明する。

図９は、プログラム自動作成方法を説明するためのフローチャートである。

まず、汎用制御文保存手段１０１が、汎用化制御文を保存する（Ｓ１）。

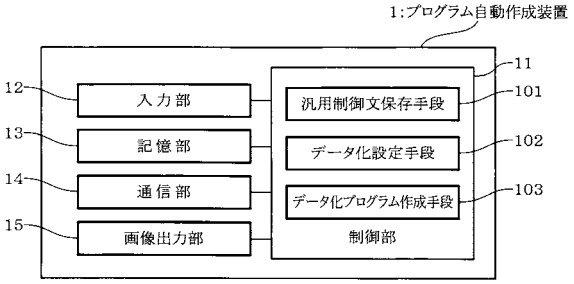
具体的には、図５のメニュー画面において、「読込」キーＫ４を選択して順次、分岐、反復の汎用制御文を記憶部１３に保存する。

【００４１】

次に、データ化設定手段１０２が、データ化設定を行う（Ｓ２）。

具体的には、順次処理のデータ化設定を行う場合、メニュー画面の順次処理キーＫ１を選択することにより表示画面Ｐに表示されるテーブルＴａにおいて、必要な設定データ（識別子、関数ポインタ）を入力する（図６参照）。
分岐処理のデータ化設定を行う場合、メニュー画面の分岐処理キーＫ２を選択することによって表示画面Ｐに表示されるテーブルＴｂにおいて、必要な設定データ（識別子、判定値、関数ポインタ）を入力する（図７参照）。
反復処理のデータ化設定を行う場合、メニュー画面の反復処理キーＫ３を選択することによって表示画面Ｐに表示されるテーブルＴｃにおいて、必要な設定データ（識別子、初期値、判定値、刻み値、関数ポインタ）を入力する（図８参照）。
ＯＫキーＫ８が選択されると、入力した設定データが設定登録されてメニュー画面に遷移し、キャンセルキーＫ９が選択されると、そのままメニュー画面に遷移する。
【００４２】
続いて、データ化プログラム作成手段１０３が、データ化プログラムの作成を行う（Ｓ３）。
具体的には、メニュー画面のデータ化処理実行キーＫ５を選択すると、ステップＳ２において行われたデータ化設定の内容がデータ化され、かつ、ステップＳ１にて保存された汎用化制御文の中からステップＳ２において設定した識別子に応じた汎用化制御文が引用されたプログラムが作成される。
保存キーＫ６を選択するとプログラムが保存され、終了キーＫ７を選択するとプログラム作成に関する処理を終了する。
【００４３】
このように、プログラム自動作成方法によっても本発明の目的を達成することができる。
このため、プログラム自動作成装置１のような構成に限らず、様々な態様にて本発明を実施することができる。
【００４４】
以上、本発明について、実施形態を示して説明したが、本発明は、上述した実施形態にのみ限定されるものではなく、種々の変更が可能であることは言うまでもない。
例えば、データ変更キーを選択可能に表示させることによって、データ化設定に係る現在の設定データを変更できるようにしたり、検証実行キーを選択可能に表示させることによって、読み込んだ汎用化制御文や作成したプログラムの正当性を検証できるようにして、拡張性を高めることもできる。
また、本実施形態においては、Ｃ言語の場合を例示して説明したが、他のプログラム言語（例えば、C++、ｊａｖａ、Ｐａｓｃａｌ等）においても本発明を適用することができる。
例えば、ｊａｖａやＣ＃を用いる場合、分岐の汎用制御文におけるｉｆ文を三項演算子やcase文に置き換え、反復の汎用制御文におけるfor文をwhile文に置き換えることができる。
【産業上の利用可能性】
【００４５】
本発明は、パーソナルコンピュータ等、プログラムの作成が可能な情報処理装置に適用することができる。
【符号の説明】
【００４６】
１　プログラム自動作成装置
１１　制御手段
１０１　汎用制御文保存部
１０２　データ化設定手段
１０３　データ化プログラム作成手段

【図１】



【図２】

```
#define SIZE 3

struct fp_tbl {
        int shikibetsushi ;
        int (*fp) ( ) ;
} ;

main( )
{
        int i ;
        int func_a( ) ;
        int func_b( ) ;
        int func_c( ) ;
        static struct fp_tbl table[ SIZE ] {
                { JUNJI, &func_a } ,
                { JUNJI, &func_b } ,
                { JUNJI, &func_c }
        } ;

        for ( i = 0 ; i < SIZE ; i++ ) {
                switch( table[i][0] ) {
                        case JUNJI:
                                fp = table[i][1] ;
                                (*fp)( ) ;
                                break ;
                        case BUNKI:
                                break ;
                        case HANPUKU ;
                                break ;
                        default:
                                break ;
                }
        }
}
```

A1 A2 A21 A3 A31

【図３】

```
#define SIZE 2

struct fp_tbl {
        int shikibetsushi ;
        int hanteichi ;
        int (*fp) ( ) ;
        int (*fp) ( ) ;
} ;

main( )
{
        int i ;
        int ret ;
        int func_d( ) ;
        int func_e( ) ;
        int func_f( ) ;
        static struct fp_tbl table[ SIZE ] {
                { BUNKI 1, DUMMY_VALUE, &func_d, DUMMY_FUNC } ,
                { BUNKI 2, 5,           &func_e, &func_f      }
        } ;

        for ( i = 0 ; i < SIZE ; i++ ) {
                switch( table[i][0] ) {
                        case JUNJI:
                                break ;
                        case BUNKI 1:
                                fp = table[i][2] ;
                                ret = (*fp)( ) ;
                                break ;
                        case BUNKI 2:
                                if( ret == table[i][1] ) {
                                        fp = table[i][2] ;
                                        (*fp)( ) ;
                                } else {
                                        fp = table[i][3] ;
                                        (*fp)( ) ;
                                }
                                break ;
                        case HANPUKU ;
                                break ;
                        default:
                                break ;
                }
        }
}
```

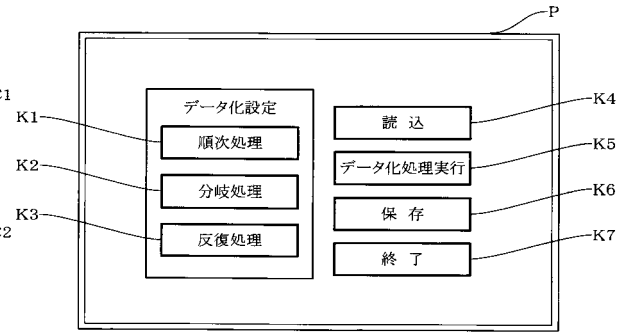
B1 B2 B21 B3 B31

【図４】

```
#define SIZE 1

struct fp_tbl {
        int shikibetsushi ;
        int syokichi ;
        int hanteichi ;
        int kizamichi ;
        int (*fp) ( ) ;
} ;

main( )
{
        int i ;
        int func_g( ) ;
        static struct fp_tbl table[ SIZE ] {
                { HANPUKU, 1, 10, 2, &func_g }
        } ;

        for ( i = 0 ; i < SIZE ; i++ ) {
                switch( table[i][0] ) {
                        case JUNJI:
                                break ;
                        case BUNKI:
                                break ;
                        case HANPUKU ;
                                for( x = table[i][1] ; x < table[i][2] ; x = x + table[i][3] ) {
                                        fp = table[i][4] ;
                                        (*fp)( ) ;
                                }
                                break ;
                        default:
                                break ;
                }
        }
}
```

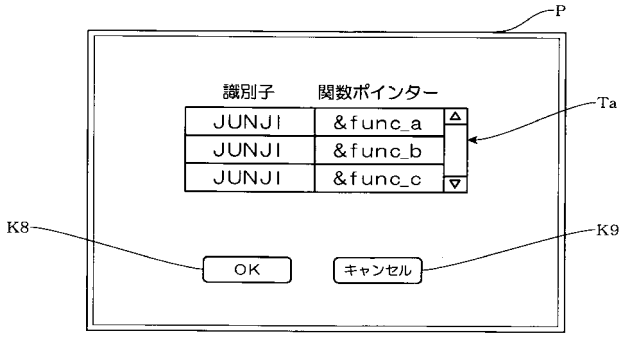
C1 C2 C21 C3 C31

【図５】



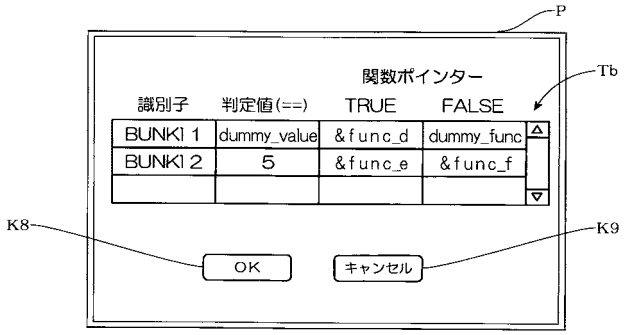
メニュー画面

【図６】



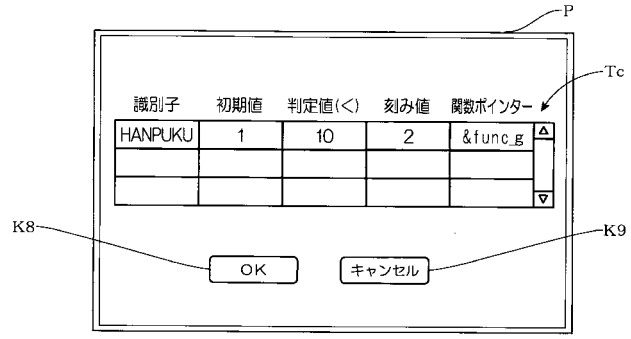
順次処理のデータ化設定画面

【図７】



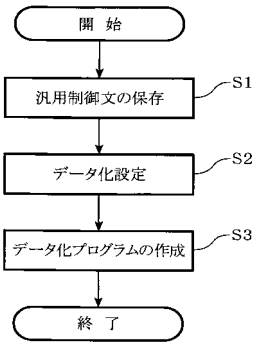
分岐処理のデータ化設定画面

【図８】



反復処理のデータ化設定画面

【図９】



【図１０】

```
main( )
{
        func_a( ) ;
        func_b( ) ;
        func_c( ) ;
}
```

関数

従来の順次処理の制御文

【図１１】

```
main( )
{
        int ret ;

        ret = func_d( ) ;
        if( ret == 5 ) {
                func_e( ) ;
        } else {
                func_f( ) ;
        }
}
```

戻り値　分岐条件

従来の分岐処理の制御文

【図１２】

```
main( )
{
        int x;
        for( x = 1; x < 10 ; x += 2 ) {
                func_g( x ) ;
        }
}
```

反復条件

従来の反復処理の制御文