14. C、C++に関する知識Ⅰ

1. 科目の概要

主要なオープンソースソフトウェアの記述言語であるC言語の基本的な知識を解説する。C 言語プログラムの構造、型、演算子、制御構文や標準的な関数、ポインタや構造体、ユーザや外部データとの入出力に求められるプログラミングなどについて説明する。

2. 習得ポイント

本科目の学習により習得することが期待されるポイントは以下の通り。

習得ポイント	説　明	シラバスの対応コマ
I-14-1. C言語の歴史と特徴、開発事例と開発手順	Cプログラミング学習の入り口として、C言語の歴史や特徴、他の言語との比較、Cによる開発事例を解説する。また、プログラムを開発する手順として、エディタによるプログラム作成からコンパイル、プログラム実行までの流れを説明する。	1
I-14-2. C言語の基本的な構造、型、演算子、制御構文	Cの基本的な仕組み、プログラムの構成、基本的な文法を説明する。基本的なプログラム記述の例を示し、変数、データ型、演算子、制御構造といったCプログラムの基本的な要素について説明を加える。	2
I-14-3. 関数の定義、標準関数の利用方法	関数とは何かを説明し、関数を活用したプログラム部品化の概念を示す。関数定義の方法と関数の利用方法、変数のスコープなど、関数の利用時に気をつけるべき点を列挙する。さらに代表的な標準関数を紹介し、標準で提供される関数で様々なことを実現できることを例示する。	4
I-14-4. ポインタを利用した効果的なプログラミング	メモリ上に配置されているデータの抽象化とアドレスの考え方を示し、ポインタの概念、特徴、利用方法について解説する。また配列や文字列、関数を利用する際にポインタを上手に利用する技法を示し、関数ポインタの概念についても言及する。	3, 5
I-14-5. 構造体の概念と構造体を使用したプログラミング	構造体の概要、定義と使い方を解説し、変数や配列などデータをまとめて扱う方法について説明する。また構造体を配列で利用する方法、構造体メンバへのポインタによるアクセス、関数への構造体データの受け渡し方法など、実際にプログラムで構造体を利用する際に必要となる技術を解説する。	6
I-14-6. マクロの利用（プリプロセッサ機能）	Cコンパイラにおけるプリプロセッサの位置づけと、プリプロセッサが備えている機能を説明する。またマクロによるプログラミング例を紹介し、マクロの効果的な使い方とマクロ利用時の留意点について解説する。	2
I-14-7. 標準入出力を利用したプログラミング	Cプログラムにおける入出力の概念を説明し、入出力ストリームに対するプログラミング手法を紹介する。ここではとくに標準入出力を対象とした文字や文字列の受け渡しについて解説する。	7
I-14-8. ファイル入出力とファイル/ディレクトリ操作	ファイル入出力、ファイル操作、ディレクトリ操作といったCプログラムからファイルにアクセスするファイル管理について説明する。またファイルに対する高水準ファイル入出力関数と低水準ファイル入出力があることを示し、具体的なデータ入出力方法を解説する。	8
I-14-9. データ構造	データ構造とは何か解説し、線形リスト、ツリー、スタック、キューといった一般的なデータ構造を紹介する。またメモリ管理とデータ構造の関係について触れ、Cプログラムによるデータ構造の実装例を紹介する。	9
I-14-10. アルゴリズム	用途に応じたデータ構造の使い方を説明し、それぞれのデータ構造を使用した代表的なアルゴリズムや効果的な利用方法を紹介する。	9

【学習ガイダンスの使い方】

1. 「習得ポイント」により、当該科目で習得することが期待される概念･知識の全体像を把握する。
2. 「シラバス」、「IT 知識体系との対応関係」、「OSS モデルカリキュラム固有知識」をもとに、必要に応じて、従来の IT 教育プログラム等との相違を把握した上で、具体的な講義計画を考案する。
3. 習得ポイント毎の「学習の要点」と「解説」を参考にして、講義で使用する教材等を準備する。

3. IT 知識体系との対応関係

「14. C、C++に関する知識Ⅰ」と IT 知識体系との対応関係は以下の通り。

科目名	基本レベル(Ⅰ)									応用レベル(Ⅱ)					
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
14. C、C++に関する知識	<Cの基本>	<Cの基本構造>	<文字列操作>	<関数>	<ポインタ>	<構造体>	<コンソール入出力>	<ファイル管理>	<データ構造>	<C++の基本>	<C++の基本構造>	<オブジェクト指向プログラミング>	<STL (Standard Template Library)>	<GUIアプリケーションの開発>	<開発ライブラリの使用>

[シラバス：http://www.ipa.go.jp/software/open/ossc/download/Model_Curriculum_05_14.pdf]

<IT 知識体系上の関連部分>

分野		科目名	1	2	3	4	5	6	7	8	9	10	11	12	13
組織関連事項と情報システム	1	IT-IAS 情報保証と情報セキュリティ	IT-IAS1. 基礎的な問題	IT-IAS2. 情報セキュリティの仕組み(対策)	IT-IAS3. 運用上の問題	IT-IAS4. ポリシー	IT-IAS5. 攻撃	IT-IAS6. 情報セキュリティ分野	IT-IAS7. フォレンジック(情報証拠)	IT-IAS8. 情報の状態	IT-IAS9. 情報セキュリティサービス	IT-IAS10. 脅威分析モデル	IT-IAS11. 脆弱性		
	2	IT-SP 社会的な観点とプロフェッショナルとしての課題	IT-SP1. プロフェッショナルとしてのコミュニケーション	IT-SP2. コンピュータの歴史	IT-SP3. コンピュータを取り巻く社会環境	IT-SP4. チームワーク	IT-SP5. 知的財産権	IT-SP6. コンピュータの法的問題	IT-SP7. 組織の中のIT	IT-SP8. プロフェッショナルとしての倫理的な問題と責任	IT-SP9. プライバシーと個人の自由				
応用技術	3	IT-IM 情報管理	IT-IM1. 情報管理の概念と基礎	IT-IM2. データベース問合わせ言語	IT-IM3. データアーキテクチャ	IT-IM4. データモデリングとデータベース設計	IT-IM5. データと情報の管理	IT-IM6. データベースの応用分野							
	4	IT-WS Webシステムとその技術	IT-WS1. Web技術	IT-WS2. 情報アーキテクチャ	IT-WS3. デジタルメディア	IT-WS4. Web開発	IT-WS5. 脆弱性	IT-WS6. ソーシャルソフトウェア							
ソフトウェアの方法と技術	5	IT-PF プログラミング基礎	IT-PF1. 基本データ構造 [14-Ⅰ-3, 5, 6, 9]	IT-PF2. プログラミングの基本的構成要素	IT-PF3. オブジェクト指向プログラミング	IT-PF4. アルゴリズムと問題解決	IT-PF5. イベント駆動プログラミング	IT-PF6. 再帰							
	6	IT-IPT 技術を統合するためのプログラミング	IT-IPT1. システム間通信	IT-IPT2. データ割り当てと交換	IT-IPT3. 統合的コーディング	IT-IPT4. スクリプティング手法	IT-IPT5. ソフトウェアセキュリティの実現	IT-IPT6. 種々の問題	IT-IPT7. プログラミング言語の概要 [14-Ⅰ-1, 2, 4]						
	7	CE-SWE ソフトウェア工学	CE-SWE0. 歴史と概要	CE-SWE1. ソフトウェアプロセス	CE-SWE2. ソフトウェアの要求と仕様	CE-SWE3. ソフトウェアの設計	CE-SWE4. ソフトウェアのテストと検証	CE-SWE5. ソフトウェアの保守	CE-SWE6. ソフトウェア開発・保守ツールと環境	CE-SWE7. ソフトウェアプロジェクト管理	CE-SWE8. 言語翻訳	CE-SWE9. ソフトウェアのフォールトトレランス	CE-SWE10. ソフトウェアの構成管理	CE-SWE11. ソフトェアの標準化	
	8	IT-SIA システムインテグレーションとアーキテクチャ	IT-SIA1. 要求仕様	IT-SIA2. 調達/手配	IT-SIA3. インテグレーション	IT-SIA4. プロジェクト管理	IT-SIA5. テストと品質保証	IT-SIA6. 組織の特性	IT-SIA7. アーキテクチャ						
システム基盤	9	IT-NET ネットワーク	IT-NET1. ネットワークの基礎	IT-NET2. ルーティングとスイッチング	IT-NET3. 物理層	IT-NET4. セキュリティ	IT-NET5. アプリケーション分野	IT-NET6. ネットワーク管理							
	10	CE-NWK テレコミュニケーション	CE-NWK0. 歴史と概要	CE-NWK1. 通信ネットワークのアーキテクチャ	CE-NWK2. 通信ネットワークのプロトコル	CE-NWK3. LANとWAN	CE-NWK4. クライアントサーバコンピューティング	CE-NWK5. データのセキュリティと整合性	CE-NWK6. ワイヤレスコンピューティングとモバイルコンピューティング	CE-NWK7. データ通信	CE-NWK8. 組込み機器向けネットワーク	CE-NWK9. 通信技術とネットワーク概要	CE-NWK10. 性能評価	CE-NWK11. ネットワーク管理	CE-NWK12. 圧縮と伸張
			CE-NWK13. クラスタシステム	CE-NWK14. インターネットアプリケーション	CE-NWK15. 次世代インターネット	CE-NWK16. 放送									
	11	IT-PT プラットフォーム技術	IT-PT1. オペレーティングシステム	IT-PT2. アーキテクチャと機構	IT-PT3. コンピュータインフラストラクチャ	IT-PT4. デプロイメントソフトウェア	IT-PT5. ファームウェア	IT-PT6. ハードウェア							
	12	CE-OPS オペレーティングシステム	CE-OPS0. 歴史と概要	CE-OPS1. 並行性	CE-OPS2. スケジューリングとディスパッチ	CE-OPS3. メモリ管理	CE-OPS4. セキュリティと保護	CE-OPS5. ファイル管理	CE-OPS6. リアルタイムOS	CE-OPS7. OSの概要	CE-OPS8. 設計の原則	CE-OPS9. デバイス管理	CE-OPS10. システム性能評価		
コンピューターアーキテクチャ	13	CE-CAO コンピュータのアーキテクチャと構成	CE-CAO0. 歴史と概要	CE-CAO1. コンピュータアーキテクチャの基礎	CE-CAO2. メモリシステムの構成とアーキテクチャ	CE-CAO3. インタフェースと通信	CE-CAO4. デバイスサブシステム	CE-CAO5. CPUアーキテクチャ	CE-CAO6. 性能・コスト評価	CE-CAO7. 分散・並列処理	CE-CAO8. コンピュータによる計算	CE-CAO9. 性能向上			
複数領域にまたがるもの	14	IT-ITF IT基礎	IT-ITF1. ITの一般的なテーマ	IT-ITF2. 組織の問題	IT-ITF3. ITの歴史	IT-ITF4. IT分野(学科)とそれに関連のある分野(学科)	IT-ITF5. 応用領域	IT-ITF6. IT分野における数学と統計学の活用							
	15	CE-ESY 組込みシステム	CE-ESY0. 歴史と概要	CE-ESY1. 低電力コンピューティング	CE-ESY2. 高信頼性システムの設計	CE-ESY3. 組込み用アーキテクチャ	CE-ESY4. 開発環境	CE-ESY5. ライフサイクル	CE-ESY6. 要件分析	CE-ESY7. 仕様定義	CE-ESY8. 構造設計	CE-ESY9. テスト	CE-ESY10. プロジェクト管理	CE-ESY11. 並行設計(ハードウェア、ソフトウェア	CE-ESY12. 実装
			CE-ESY13. リアルタイムシステム設計	CE-ESY14. 組込みマイクロコントローラ	CE-ESY15. 組込みプログラム	CE-ESY16. 設計手法	CE-ESY17. ツールによるサポート	CE-ESY18. ネットワーク型組込みシステム	CE-ESY19. インタフェースシステムと混合信号システム	CE-ESY20. センサ技術	CE-ESY21. デバイスドライバ	CE-ESY22. メンテナンス	CE-ESY23. 専門システム	CE-ESY24. 信頼性とフォールトトレランス	

4. OSS モデルカリキュラム固有の知識

OSS モデルカリキュラム固有の知識として、Linux 上での標準入出力、ファイル入出力を扱うための知識がある。また、OSS である vi エディタ、gcc コンパイラ(GNU Compiler Collection)を用いた開発の流れを学ぶ。

科目名	第1回	第2回	第3回	第4回	第5回	第6回	第7回	第8回	第9回
14.C、C++に関する知識Ⅰ	(1)C の概要 (2)C による開発の流れ	(1)C の書き方の特徴 (2)基本的なプログラム記述の例	(1)配列と文字列 (2)文字列操作関数	(1)関数の説明 (2)ユーザ定義関数 (1) 標準関数	(1)ポインタの基礎 (2)ポインタによるプログラミング	(1)構造体の概要 (2)構造体を使用したプログラム	(1)コンソール入出力の説明 (2)標準入出力 (3)対話型入出力	(1)ファイル管理の説明 (2)ファイル入出力 (3)ファイル操作とディレクトリ操作	(1)ファイル操作とディレクトリ操作 (2)線形リストとは(データ構造の一例) (3)その他のデータ構造の種類と適用例(プログラム例)

(網掛け部分は IT 知識体系で学習できる知識を示し、それ以外は OSS モデルカリキュラム固有の知識を示している)

スキル区分	科目	レベル
プログラミング分野	14　C、C++に関する知識 I	基本
習得ポイント	I-14-1. C 言語の歴史と特徴、開発事例と開発手順	
対応する コースウェア	第 1 回 (C の基本)	

I-14-1. C 言語の歴史と特徴、開発事例と開発手順

C プログラミング学習の入り口として、C 言語の歴史や特徴、他の言語との比較、C による開発事例を解説する。また、プログラムを開発する手順として、エディタによるプログラム作成からコンパイル、プログラム実行までの流れを説明する。

【学習の要点】

* C は UNIX などのシステムソフトからアプリケーションまで、非常に幅広く使われている言語である。
* C は強力なポインタ機能を有し、OS やドライバの記述に必要な低レベル処理が記述できる。
* C は他の言語に比べるオーバーヘッドが少なく、一般に高速である。
* C のソースコードはテキストエディタなどを使って書き、コンパイルすることでプログラム開発の流れがつかめる。

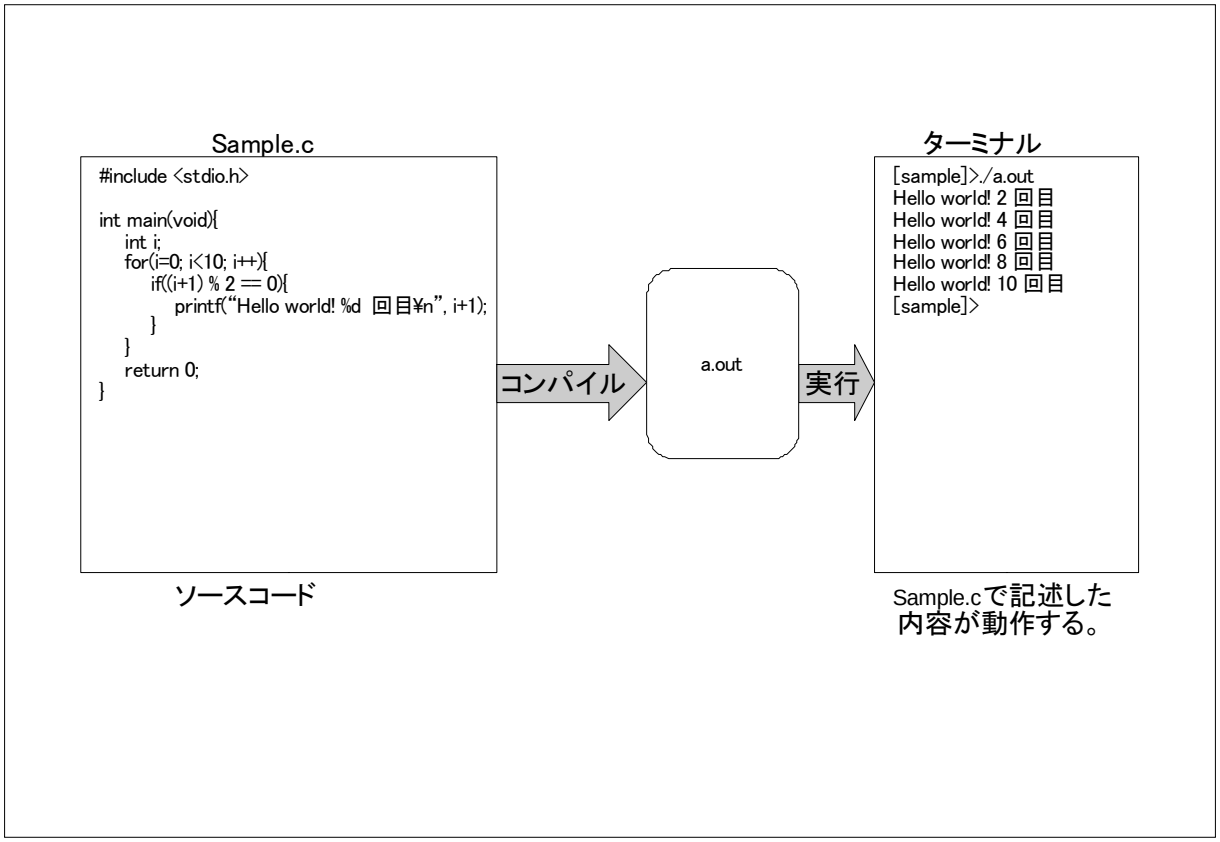

図 I-14-1.　C プログラミングの流れ

【解説】

1) C の歴史

C は 1970 年代前半に開発されたプログラミング言語である。
C の歴史は UNIX の歴史と切り離すことができない。UNIX は C で開発され、UNIX の普及が C を支え、C の普及が UNIX を支えてきた。他の様々なプログラミング言語が誕生し普及した現在においても、最も重要なプログラミング言語の一つとして位置づけられている。

2) C の特徴

C の一番の特徴といえるのが、強力なポインタ機能である。この強力なポインタ機能により、アセンブリで開発するような低レベル処理(ハードウェア制御などの処理)も可能にし、多くの OS、周辺機器のドライバソフトウェア、他のプログラミング言語の処理系などが C で開発されている。ソースコードの記述の自由度も高く、高度なプログラミングができる反面、プログラムの保守性やセキュリティなどの面でリスクを抱えている。

3) 他の言語との比較

アセンブリと比較すると、C は人間が使う言葉に非常に近い記述ができ、高級言語と呼ばれるが、低レベル処理が可能であることから、他の高級言語とアセンブリとの中間に位置づけられることが多い。
C++は C にオブジェクト指向を取り込んだ言語である。その他の C の後発言語の多くは、C のポインタ機能などの一部をあえて隠蔽しており、プログラムの保守性やセキュリティ面、学習の容易性の利点がある反面、ハードウェア制御といった低レベル処理には適さないものとなっている。これらの言語で書かれたプログラムでは、上記利点を得るために、ソースコードに記述されないさまざまな処理を内部で行うが、C ではそのような処理がほとんどなく、オーバヘッドが小さい。よって、C で書かれたプログラムは、他の言語で書かれたものよりも一般的に高速に動作する。

4) C のプログラム作成と実行の流れ

C のプログラミングの流れは、図Ⅰ-14-1 に示したような手順になる。

* テキストエディタを用いてソースコードを記述する。
* gcc などのコンパイラを用いてコンパイルを実行する。gcc の場合、a.out という実行ファイルができる。
* 実行ファイルを実行する。

スキル区分	OSS モデルカリキュラムの科目	レベル
プログラミング分野	14　C、C++に関する知識 I	基本
習得ポイント	I-14-2. C 言語の基本的な構造、型、演算子、制御構文	
対応する コースウェア	第 2 回（C の基本構造）	

I-14-2. C 言語の基本的な構造、型、演算子、制御構文

Cの基本的なしくみ、プログラムの構成、基本的な文法を説明する。基本的なプログラム記述の例を示し、変数、データ型、演算子、制御構造といった C プログラムの基本的な要素について説明を加える。

【学習の要点】

＊ 制御構文によって、繰り返しや分岐処理など、プログラムの実行順を制御することができる。

＊ C でプログラムを書く時は、データ型に気を付ける。

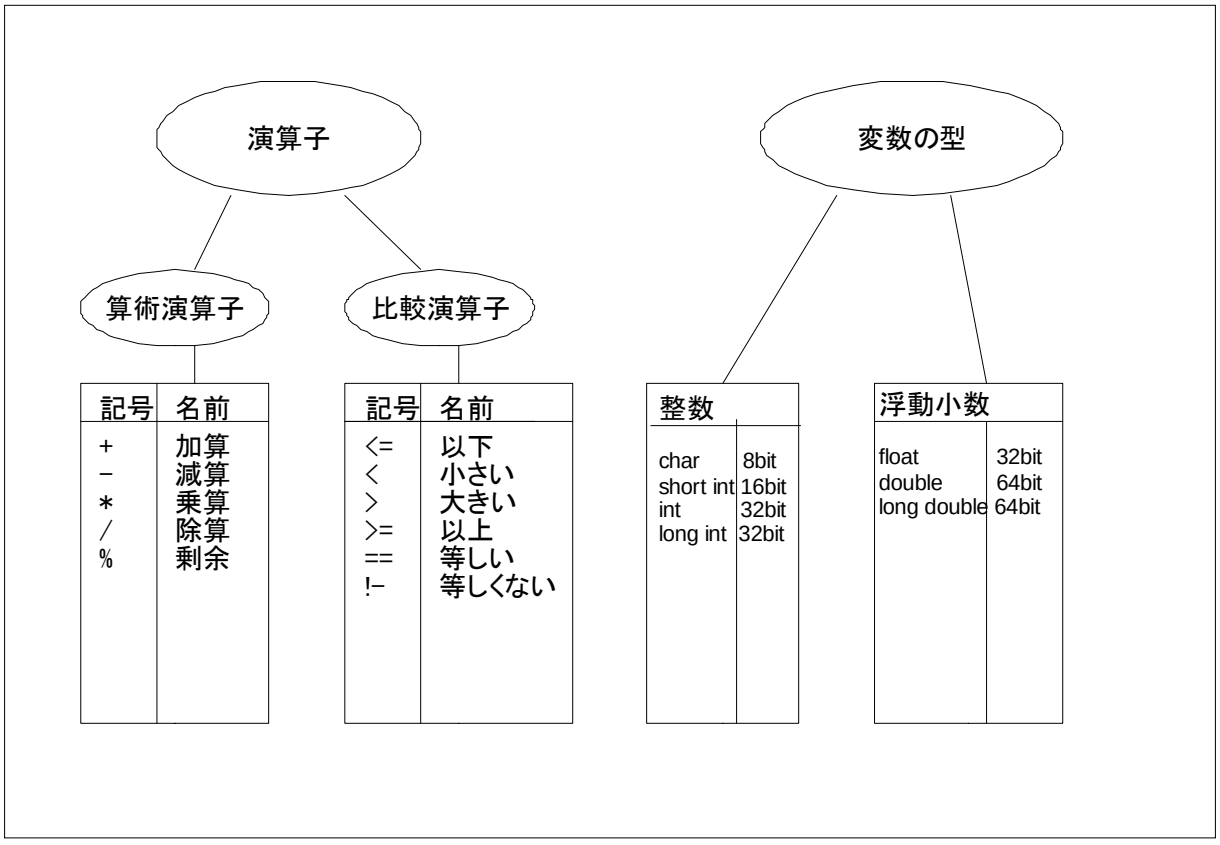

図 I-14-2.　C の基本的な演算子と変数型

【解説】

1) C の基本的な仕組み

C は手続き型言語と呼ばれている。プログラムの実行が、上から記述した順に行われていくからである。文の終わりには必ずセミコロン(;)を付ける。そうすることで、命令文が一文終わったことを示す。また、C ではソース内にコメントを書くことができる。 /* と */ とで囲まれた部分や、 // から改行までがコメントになる。

2) C の基本的な制御構文

C は制御構文によってプログラムの挙動を制御する。制御構文は、プログラムの状態と指定された条件を比べて、その条件が当てはまったとき、または当てはまらなかったときの挙動を変えるために指定するものである。この制御構文を用いることで、繰り返しや分岐処理ができる。以下に代表的な制御構文を紹介する。

* if
  ある条件に当てはまった場合の挙動を制御する。
* for
  指示した回数だけ挙動を繰り返す。
* return
  関数を終了する時点を指示する。

3) C の基本的なデータ型

C で変数を宣言するには、次の通りに書く。

型名 変数名;

基本的なデータ型と格納されるデータサイズは一般には以下の通りである。

* 整数型
  - int : 32bit　short : 16bit　long : 32bit　char : 8bit (文字型)
* 浮動小数型
  - float : 32bit　double : 64bit

4) C の基本的な演算子

* 算術演算子
  + 加算　　- 減算　　* 乗算　　/ 除算　　% 剰余
* 比較演算子
  > より大きい　　< 未満　　>= 以上　　<= 以下　　== 等価
  != 等しくない
* ビット演算子
  & 積　　| 和　　~ 補数　　<< 左シフト　　>> 右シフト

スキル区分	OSS モデルカリキュラムの科目	レベル
プログラミング分野	14　C、C++に関する知識 I	基本
習得ポイント	I-14-3. 関数の定義、標準関数の利用方法	
対応する コースウェア	第 4 回（関数）	

I-14-3. 関数の定義、標準関数の利用方法

関数とは何かを説明し、関数を活用したプログラム部品化の概念を示す。関数定義の方法と関数の利用方法、変数のスコープなど、関数の利用時に気をつけるべき点を列挙する。さらに代表的な標準関数を紹介し、標準で提供される関数で様々なことを実現できることを例示する。

【学習の要点】

＊ 関数とは、ある手続きをひとまとめにしたものである。

＊ 関数を活用するとプログラムを構造的に書くことができ、読みやすく、保守性の高いプログラムを作ることができる。

```
#include<stdio.h>

//関数の宣言
float cal_tri_area(float width, float height);

int main(void){
    float ans1, ans2;

    ans1 = cal_tri_area(1.0,2.0);
    printf("%f¥n", ans1);

    ans2 = cal_tri_area(3.0,4.0);
    printf("%f¥n", ans2);

    return 0;
}

//関数定義
float cal_tri_area(float width, float height){
    float area;
    area = (width*height)/2.0;
    return area;
}
```

↑三角形の面積を
計算していることが
明白になっている

```
#include <stdio.h>

int main(void){
    float ans1, ans2;

    /*ここで三角形の面積の計算を
    しているが、分かりづらい*/
    ans1 = (1.0*2.0)/2.0;
    printf("%f¥n",ans1);

    ans2 = (3.0*4.0)/2.0;
    printf("%f¥n",ans2);

    return 0;
}
```

↑何をやっているか
分かりにくい

図 I-14-3.　関数を使ったプログラム

【解説】

1) 関数とは何か

関数とは、ある手続きをひとまとめにしたものである。関数には引数と返り値というものがあり、関数実行時に関数に渡す値を引数、そして計算結果として出てきた値として呼び出し元に返す値を返り値と呼ぶ。頻繁に使う手順などを関数化することで、コードの削減と可読性を高めることができる。また、汎用性の高い関数を作り、一つのプログラムだけではなく複数のプログラムで活用することで、開発時間の短縮を図れる。こうした汎用性の高い関数を作り、活用していくことを、プログラム部品化と呼んでいる。

2) 関数の宣言

関数は、次のように宣言することで使うことができるようになる。

```
返り値の型 関数名(引数の型 引数の名前, ...){
    関数で行いたい手続きの記述
    return 返り値;
}
```

例) 引数の 2 乗を返す関数

```
float square(float a){
    float s;
    s = a * a;
    return s;
}
```

3) 関数の利用

関数を利用する場合は、「関数名(引数)」のように記述する。

例)

```
r = square(1.1);
```

この場合は 1.1 が引数となり、返り値は 1.21 となるので、r には 1.21 が代入される。

4) 変数のスコープ

変数には、グローバル変数とローカル変数の二種類がある。グローバル変数はどこからでも利用でき、ローカル変数は使える場所が制限されている。ローカル変数が使える場所は、そのローカル変数が宣言された関数の中である。ある変数を参照できるプログラム中の範囲のことを、その変数のスコープという。

5) 標準関数

C の規格として定められている、標準ライブラリというファイルに定義されている関数。利用するときにはプログラムの頭に #include〈標準ライブラリ名〉と書いて、ライブラリを読み込んで利用する。代表的なものは以下の通り。

* printf() 文字列などを出力する関数（定義されているライブラリ:stdio.h）
* fgets() 文字列を読み込む関数（定義されているライブラリ:stdio.h）
* strcpy() 文字列をコピーする関数（定義されているライブラリ:string.h）

スキル区分	OSS モデルカリキュラムの科目	レベル
プログラミング分野	14　C、C++に関する知識 I	基本
習得ポイント	I-14-4. ポインタを利用した効果的なプログラミング	
対応する コースウェア	第 5 回（ポインタ）	

I-14-4. ポインタを利用した効果的なプログラミング

メモリ上に配置されているデータの抽象化とアドレスの考え方を示し、ポインタの概念、特徴、利用方法について解説する。また配列や文字列、関数を利用する際にポインタを上手に利用する技法を示し、関数ポインタの概念についても言及する。

【学習の要点】

＊ すべての変数はアドレスを割り当てられ、そのアドレスが指すメモリに保存されている。

＊ ポインタとは、あるデータが格納されているメモリのアドレスを保存したものである。

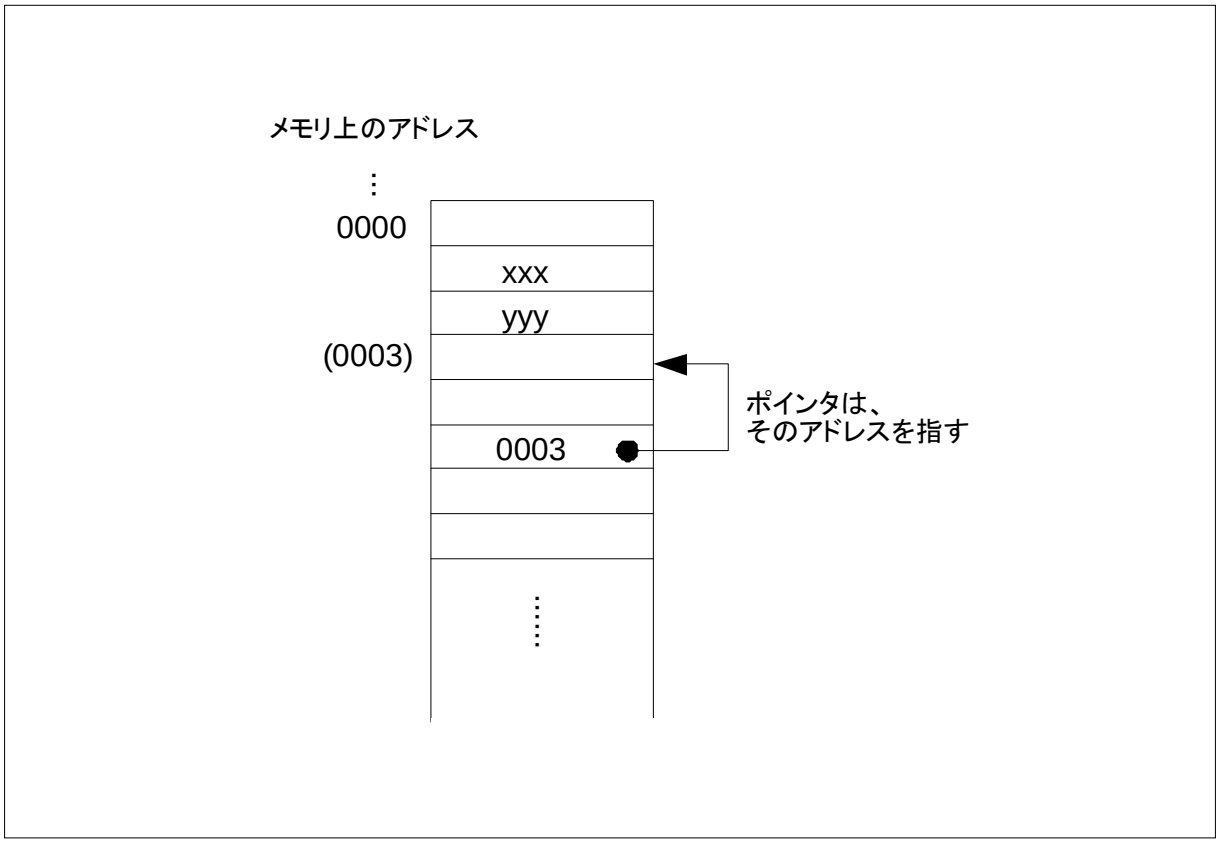

図 I-14-4.　ポインタの概念

【解説】

1) データ格納アドレスの抽象化

メモリ上のデータは、本来ならばアクセスするのにメモリのアドレスを指定する必要がある。しかし、データにアクセスするためにいちいち人が手でメモリのアドレスを指定するのは、非常に困難である。そのため、データに変数名をつけて、その名前を指定することでデータが入っているメモリにアクセスできるようにした。こうすることで、データをメモリの実アドレスを指定することなく、抽象的に扱うことができるようになる。

2) ポインタの概念、特徴、利用方法

変数はそれぞれコンパイラによって自動的にアドレスを割り当てられる。通常はこのアドレスを意識する必要はないが、ポインタを使うことでアドレスを利用することができる。ポインタは、メモリのアドレスを保存しておくためのものである。ポインタ型の変数を利用するには、変数名の前に「*」をつけて宣言をする必要がある。また、ポインタに関する演算子として、アドレスを表す「&」、逆参照を表す「*」がある。例えば、「&var」は変数 var のアドレスを指すポインタを示し、「*ptr」はポインタ変数 ptr が指すアドレスに格納されている値を示す。

3) 関数引数としてのポインタ

C では引数は値呼び出しで関数に渡される。変数を引数として関数に渡しても、その変数の値を関数内から変更することはできない。。しかし、ポインタを引数として関数を宣言し、関数呼び出し時にアドレスを渡すと、関数内でポインタが差す値（実際に引数として使用される変数の値）を変更することができるようになる。

4) ポインタと配列

C での配列の宣言の仕方は、次のとおりである。

```
型名 配列名[確保したい配列の要素数];
```

配列の要素を指定するときは、「 配列名[要素番号] 」のように記述する。

例）

```
int arr[100];
arr[3] = 21;
```

この例の場合、C では単に「 arr 」と書くことで、配列 arr の先頭要素を指すポインタを表すことになる。つまり、「 arr 」は「 &arr[0] 」と等価である。

5) ポインタと文字列

C には文字列型がなく、文字列は文字を配列にしたものであらわされる。配列で表せるということは、ポインタでも表せるということである。

6) 関数ポインタについて

関数も変数と同じくメモリ上にアドレスを持っている。これは関数のポインタということができ、ポインタ変数で扱うことができる。これを使用することで、動的に呼び出す関数を変えることができる。

スキル区分	OSS モデルカリキュラムの科目	レベル
プログラミング分野	14　C、C++に関する知識 I	基本
習得ポイント	I-14-5. 構造体の概念と構造体を使用したプログラミング	
対応する コースウェア	第 6 回（構造体）	

I-14-5. 構造体の概念と構造体を使用したプログラミング

構造体の概要、定義と使い方を解説し、変数や配列などデータをまとめて扱う方法について説明する。また構造体を配列で利用する方法、構造体メンバへのポインタによるアクセス、関数への構造体データの受け渡し方法など、実際にプログラムで構造体を利用する際に必要となる技術を解説する。

【学習の要点】

＊ 構造体とは、複数のデータ型をひとまとめに扱えるようにしたものである。

＊ 構造体は配列とは違い、柔軟性に富んだデータの扱いができる。

```
struct person{                  /* 構造体の定義 */
    char firstname[20];         /* フィールドの定義 */
    char familyname[20];
    char zipcode[9];
    char address[80];
    char phone[20];
}

struct person p1;               /* 構造体personの変数の宣言 */

p1.firstname = "太郎";          /* フィールドへの値の代入 */
p1.familyname = "山田";
p1.zipcode = "123-0123";
p1.address = "東京都千代田区〇〇〇１－２－３";
p1.phone = "03-1234-5678"
```

図 I-14-5.　構造体の定義

【解説】

1) 構造体とその定義

構造体とは、異なるデータ型をグループとして扱うために用いる機能である。同じデータ型をグループとして用いる配列とは違い、柔軟性に富んだデータの扱いができる。
構造体を定義するためには、以下のような文法を用いる。

```
struct 構造体名 {
    フィールドのデータ型 フィールドの名前;
    フィールドのデータ型 フィールドの名前;
    ……
};
```

フィールドとは、構造体を構成する要素のこと。構造体では、フィールドをそれぞれ違うデータ型にできる。

2) 構造体の使い方

構造体の各フィールドにアクセスするには、次のような構文を利用する。

```
変数名.フィールド名
```

3) 構造体と配列

構造体は配列にしても用いることができる。配列として用いる場合の宣言方法と、メンバへのアクセス方法は下記のようにする。

＊ 宣言例

```
struct elem{
    int f1;
    int f2;
};
struct elem a[10];
```

＊ アクセス例

```
a[0].f1 = 0;
```

4) 構造体のポインタ

ポインタを用いて構造体のフィールドにアクセスするには、演算子「 -> 」を利用する。

```
変数名->フィールド名
```

スキル区分	OSS モデルカリキュラムの科目	レベル
プログラミング分野	14　C、C++に関する知識 I	基本
習得ポイント	I-14-6. マクロの利用(プリプロセッサ機能)	
対応する コースウェア	第 2 回 (C の基本構造)	

I-14-6. マクロの利用(プリプロセッサ機能)

C コンパイラにおけるプリプロセッサの位置づけと、プリプロセッサが備えている機能を説明する。また、マクロによるプログラミング例を紹介し、マクロの効果的な使い方とマクロ利用時の留意点について解説する。

【学習の要点】

＊ プリプロセッサとは、コンパイルの前処理をするプログラムである。

＊ プリプロセッサを用いれば、文字列の置換や条件付きコンパイル、外部ファイルの挿入などができる。

＊ プリプロセッサ用の構文は C の構文とは別物なので、その扱いには注意しなければならない。

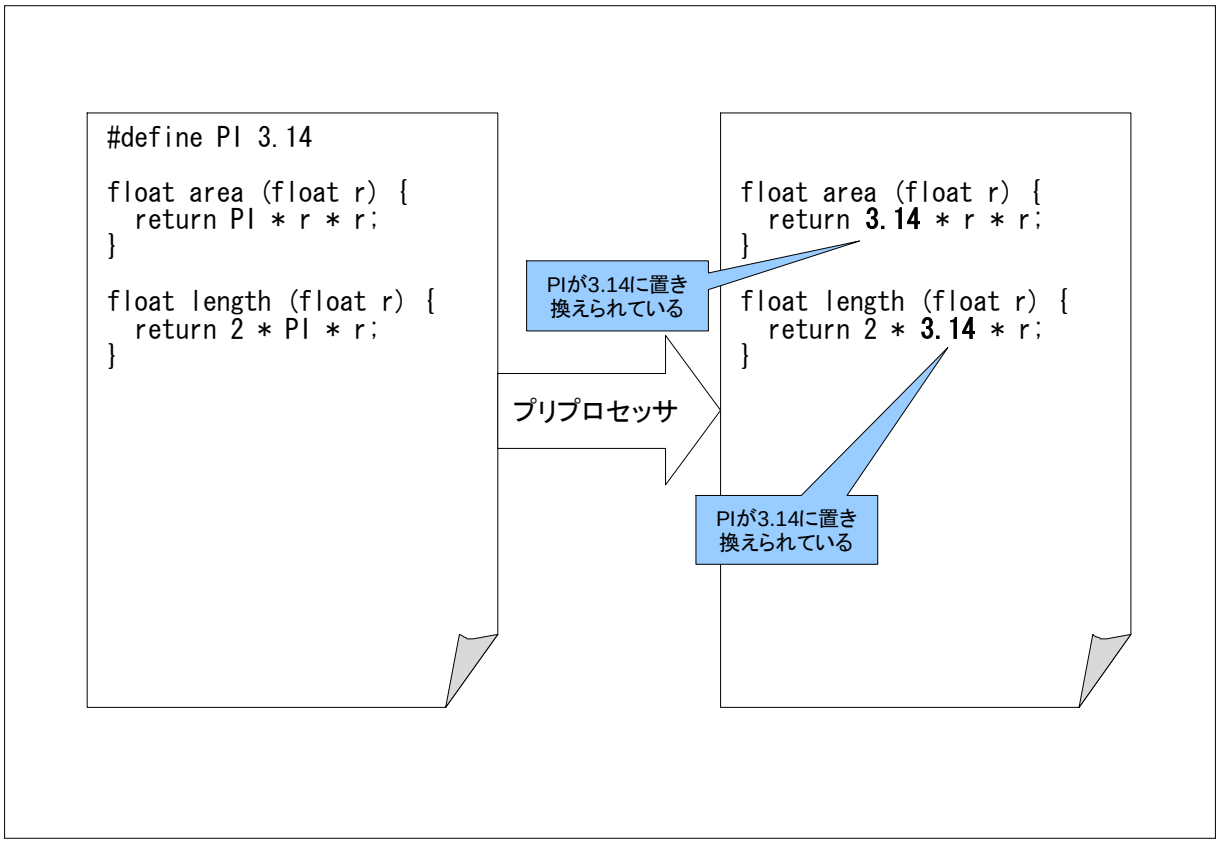

図 I-14-6.　プリプロセッサに通す前と後

【解説】

1) C コンパイラにおけるプリプロセッサの位置づけ

プリプロセッサとは、コンパイルの前に実行し、ソースコードに変更を加えるプログラムである。C では、#から始まる行がプリプロセッサへの指示となっている。

2) プリプロセッサが備える機能

プリプロセッサが備える機能の一部を紹介する。

＊ マクロ

「#define」命令により、マクロを定義できる。例えば、次のコマンドにより、プリプロセッサはソースコード中のすべての「PI」という文字列を「3.14」に置き換える。

例） #define PI 3.14

＊ 条件付きコンパイル

デバッグ用のコードをプログラムに埋め込み、出荷時には取り除きたい場合には、条件付きコンパイル #ifdef/#endif を用いる。

例）

```
#define DEBUG
#ifdef DEBUG
        printf (″ステータスは%d¥n″, status);
#endif
```

デバッグ中はプログラム先頭にマクロ DEBUG を定義しておくと、変数 status が表示される。

```
#undef DEBUG
#ifdef DEBUG
printf (″ステータスは%d¥n″, status);
#endif
```

とマクロ DEBUG を未定義に変えると、#ifdef～#endif の間はコンパイル対象から外れる。

＊ インクルードファイル

#include 命令を使用すると、プログラム中で別のファイルのソースコードを挿入できる。マクロ定義を複数のファイルで共通に使いたい場合などに用いる。

3) マクロ利用時の留意点

プリプロセッサとコンパイラとはソースコードの解釈が異なるので、プリプロセッサ命令中で C 言語の構文は使えないことに注意する必要がある。また、プリプロセッサは適切な C の構文であるかどうかをチェックしないため、特にマクロで書き換えたため予期せぬ問題が生じることがあることにも注意が必要である。

スキル区分	OSS モデルカリキュラムの科目	レベル
プログラミング分野	14　C、C++に関する知識 I	基本
習得ポイント	I-14-7. 標準入出力を利用したプログラミング	
対応する コースウェア	第 7 回（コンソール入出力）	

I-14-7. 標準入出力を利用したプログラミング

C プログラムにおける入出力の概念を説明し、入出力ストリームに対するプログラミング手法を紹介する。ここでは特に標準入出力を対象とした文字や文字列の受け渡しについて解説する。

【学習の要点】

＊ C では標準入出力もファイルとして扱われるので、ファイルの入出力と同様の方法で標準入出力にアクセスできる。

＊ fgets()関数や fputs()関数の引数に標準ファイルを指定することで、標準入出力を対象とした文字列の受け渡しができる。

```
#include <stdio.h>
#include <stdlib.h>
const char IN_FILE_NAME[] = "input.txt";
const char OUT_FILE_NAME[] = "output.txt";

int main(){
    int count = 0;
    FILE *in_file, *out_file;
    char string[100];
    char *ch;

    in_file = fopen(IN_FILE_NAME, "r");
    if (in_file == NULL) {
        printf("Cannot open %s¥n", IN_FILE_NAME);
        exit(8);
    }
    out_file = fopen(OUT_FILE_NAME, "w");

    while (1) {
        ch = fgets(string, sizeof(string),in_file);
        if(ch == NULL)
            break;
      fputs(string, out_file);
      ++count;
    }
    printf("Number of line in %s id %d¥n",
            IN_FILE_NAME, count);

    fclose(in_file);
    fclose(out_file);
    return (0);
}
```

input.txtから一行ずつ読み取り、output.txtにコピーし、行数を表示するプログラム

ファイルをオープンしている

ファイルを一行ずつ読み込んでいる

ファイルを一行ずつ書きだしている

図 I-14-7.　入出力ストリーム

【解説】

1) C プログラムにおける入出力

C プログラムにおける入出力は、すべてファイルに対して行われる。キー入力と画面出力（標準入出力と呼ばれる）もファイルとして扱われる。

* stdin　　標準入力(通常はキー入力)
* stdout　　標準出力(通常は画面表示)
* stderr　　標準エラー出力(通常は画面表示)

これらのファイルはＣでは常にオープン状態となっているため、明示的にオープンする命令は不要である。

2) 標準入力からの文字列の読み込み

標準関数 fgets()を使用すると、標準入力から文字列を読み込むことができる。fgets()呼び出しの一般的な形は以下のとおりである。

```
fgets (name, sizeof(name), stdin);
```

ここで、name は文字列変数を表す。引数は以下のようになる。

* name
  文字配列の名前を指定する。1 行（文字列終端文字を含む）が、この配列に読み込まれる。
* sizeof(name)
  読み込む文字の最大数（文字列終端文字の 1 文字分を加えた数）を指定する。関数 sizeof()を使用すると、読み込む文字数を変数が保持可能な数に制限できる。
* stdin
  読み込むべきファイルを指定する。ここでは、このファイルは標準入力となる。

3) 標準出力への文字列の出力

文字列は fputs()を使って標準出力に書きだすことができる。

例)

```
fputs(name, stdout);
```

4) 標準エラー出力への文字列の出力

標準出力と同様である。

例)

```
fputs(name, stderr);
```

スキル区分	OSS モデルカリキュラムの科目	レベル
プログラミング分野	14　C、C++に関する知識 I	基本
習得ポイント	I-14-8. ファイル入出力とファイル／ディレクトリ操作	
対応する コースウェア	第 8 回 (ファイル管理)	

I-14-8. ファイル入出力とファイル／ディレクトリ操作

ファイル入出力、ファイル操作、ディレクトリ操作といった C プログラムからファイルにアクセスするファイル管理について説明する。またファイルに対する高水準ファイル入出力関数と低水準ファイル入出力があることを示し、具体的なデータ入出力方法を解説する。

【学習の要点】

* ファイルの入出力には、ファイル変数を使う。
* ディレクトリ操作をするには、direct.h に入っている標準関数を使う。
* 通常はシステムの頻繁な呼び出しを防ぐためバッファリングの仕組みを備える高水準ファイル入出力を用いる。

```
printf("Starting");

do_step_1();
printf("Step 1 complete");

do_step_2();
printf("Step 2 complete");

do_step_3();
printf("Step 3 complete");

↑
バッファリングをやるI/O
```

```
printf("Starting");

do_step_1();
fprintf("Step 1 complete");


do_step_2();
fprintf("Step 2 complete");


do_step_3();
fprintf("Step 3 complete");

↑
バッファリングをやらないI/O
```

図 I-14-8.　ファイルアクセスの例

【解説】

1) ファイルの入出力

ファイルの入出力を行うには、以下のような手順が必要となる。

＊ ファイル変数を宣言する。

```
FILE *infile、*outfile;
```

＊ ファイルをオープンにする。

```
infile = fopen("infile.dat", "r");      //読み取りモードでオープン
outfile = fopen("outfile.dat", "w"); //書き込みモードでオープン
```

＊ ファイルを読み書きする。

```
fgets(str, sizeof(str), infile);        //文字列変数 str に 1 行読み込む場合
fputs(str, outfile);                    //文字列変数 str の内容を書き込む場合
```

＊ ファイルをクローズする。

```
fclose(outfile);
fclose(infile);
```

2) ディレクトリ操作の方法

UNIX 系 OS でディレクトリを操作する関数には、次のようなものがある。

＊ mkdir()　　ディレクトリの作成

＊ rmdir()　　ディレクトリの削除

＊ opendir()　　ディレクトリのオープン

＊ readdir()　　オープンしたディレクトリのファイル一覧の取得

＊ closedir()　　ディレクトリのクローズ

3) 高水準ファイル入出力と低水準ファイル入出力の使い分け

一時的にデータをため込む領域のことを、バッファという。fopen()、fclose()などの高水準ファイル入出力関数はバッファを使用した入出力を行う。open()、close()などの低水準ファイル入出力関数ではバッファを使用しない。低水準ファイル入出力関数を利用すると、システムコールが頻繁に発生するので、通常は高水準ファイル入出力関数を利用する。

4) 具体的なファイル入力方法

ファイル入力例を図 I-14-8 に示す。

スキル区分	OSS モデルカリキュラムの科目	レベル
プログラミング分野	14　C、C++に関する知識 I	基本
習得ポイント	I-14-9. データ構造	
対応する コースウェア	第 9 回（データ構造）	

I-14-9. データ構造

データ構造とは何か説明し、線形リスト、ツリー、スタック、キューといった一般的なデータ構造を紹介する。またメモリ管理とデータ構造の関係に触れ、C プログラムによるデータ構造の実装例を紹介する。

【学習の要点】

＊ データ構造はアルゴリズムと深く関わる、プログラムの根幹をなすものである。

＊ 線形リスト、ツリー、スタック、キューといったデータ構造は、ポインタを用いることで実現できる。

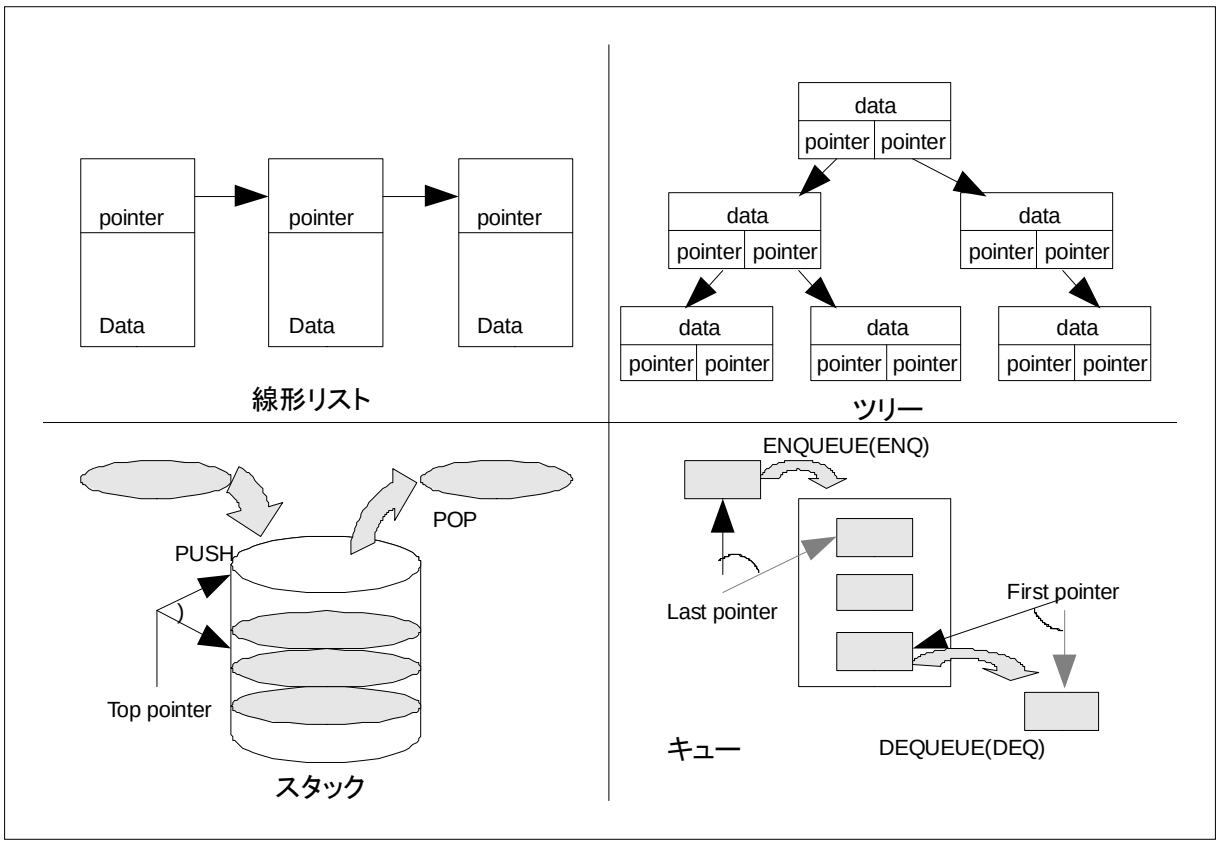

図 I-14-9.　各データ構造のイメージ図

【解説】

1) データ構造とは

データ構造とは、複数のデータの関連性を表現するモデルである。用いるアルゴリズムによって適切なデータ構造は異なり、その逆でデータ構造によって適切なアルゴリズムも異なる。プログラミングをする上でデータ構造は非常に重要な要素である。

2) 代表的なデータ構造

＊ 線形リスト
各ノードが次のノードへのポインタを持ち、データが直線状に並んだデータ構造。

＊ ツリー
各ノードがポインタを二つ以上持ち、それぞれのポインタが下位のノードを指す、データが木のように枝分かれするデータ構造。

＊ スタック
各ノードが積み重なっており、最後に追加されたデータが最初に取り出される、重ね餅のようなデータ構造。

＊ キュー
各ノードが積み重なっており、最初に追加されたデータが最初に読みだされる、パイプのようなデータ構造。

3) メモリ管理とデータ構造

各種のデータ構造に適したメモリ割り当てをすることにより、必要最小限のメモリで実装できる。データ構造がメモリ上でどう割り当てられているかを把握することが重要である。

4) データ構造の実装例

ここでは例として、線形リストの実装例を紹介する。

＊ 構造体のフィールドで、同じ構造体へのポインタを定義

```
struct linearelem {
        char data[100];   // 実際のデータ
        struct linearelem *next; // 次の構造体へのポインタを格納
};
```

＊ 構造体を3つと構造体へのポインタを宣言

```
struct linearelem first, second, third, *curr;
```

＊ ポインタでつないで線形リストを作成

```
first.next = &second;
second.next = &third;
```

＊ 最初の構造体を指し示す場合

```
curr = &first;
```

＊ 次の構造体を指し示す場合

```
curr = curr->next;
```

スキル区分	OSS モデルカリキュラムの科目	レベル
プログラミング分野	14　C、C++に関する知識 I	基本
習得ポイント	I-14-10. アルゴリズム	
対応する コースウェア	第 9 回（データ構造）	

I-14-10. アルゴリズム

用途に応じたデータ構造の使い方を説明し、それぞれのデータ構造を使用した代表的なアルゴリズムや効果的な利用方法を紹介する。

【学習の要点】

＊ データ構造は、用途によって使い分けることで、効率的なアルゴリズムを実行することができる。

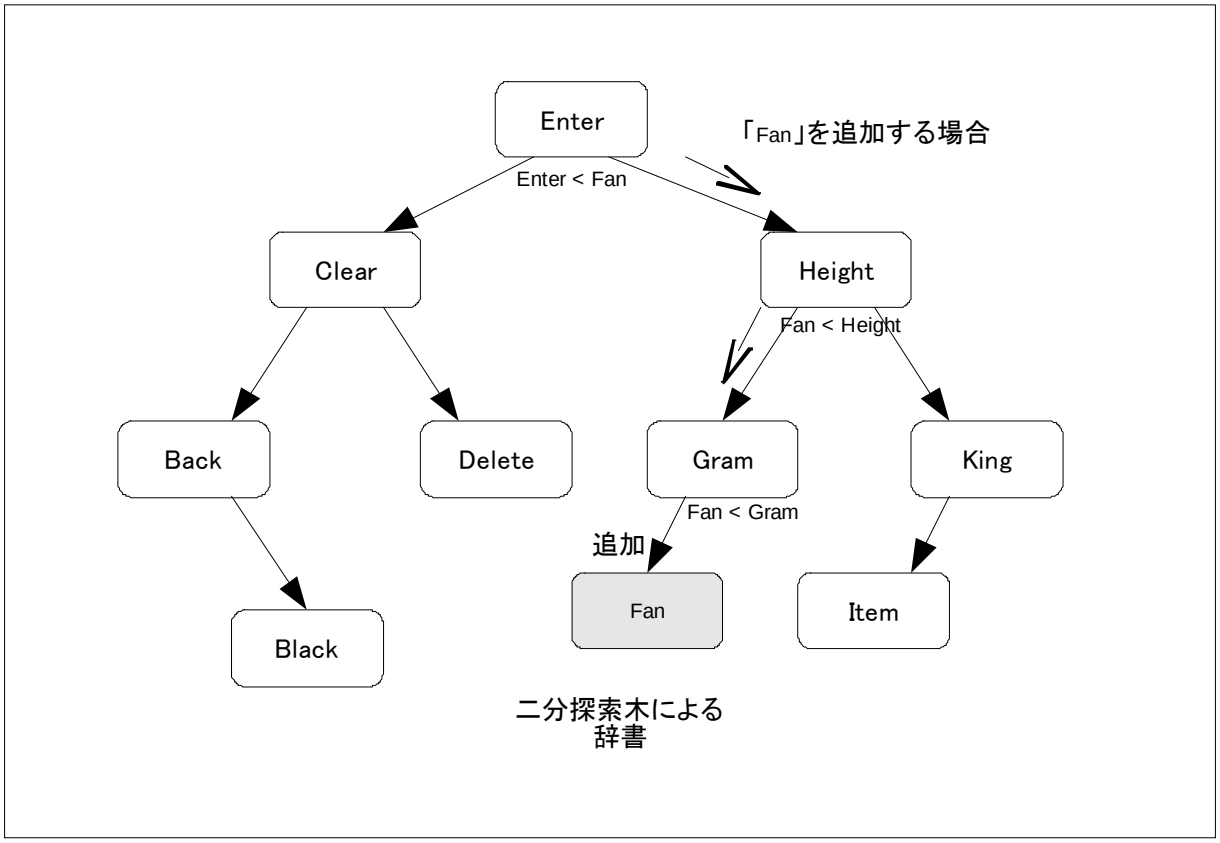

図 I-14-10.　二分探索木のデータ構造

【解説】

1) 用途に応じたデータ構造の使い方

＊ テスト結果のリストを点数順にして作りたい時など、リストに挿入、削除を頻繁に行いたい場合
線形リストが使える。もしこのリストに配列を使ったとしたら、リストの途中に挿入を行うと、その挿入を行った場所より後のデータも並びなおさなければならないが、線形リストだとポインタのつながり方を直すだけで済み、挿入した場所よりも後ろのデータの並び替えをする必要がない。これにより、データを作る時点で点数順に並べたリストを作ることができる。

＊ 辞書を作ってそれを活用したい時など、データの検索を頻繁に行いたい場合
二分探索木が使える。各ノードに一つ単語を割り当て、ノードを追加する際に後述する規則を守ることで、検索する際にその規則を利用して検索することができるので、線形検索よりも早く検索することができる。

2) 代表的なアルゴリズム

ここでは、例として二分探索木を用いて辞書を作るアルゴリズムを紹介する。

＊ 二分探索木とは
二分探索木とは、ツリー状のデータ構造の一種である。ツリー状のデータ構造では、ルートノードから枝分かれしていく構造をとり、あるノードから枝分かれした先のノードをそのノードの子ノードと呼び、そのノードは子ノードからみた親ノードという。ツリー構造では、ルートを除く各ノードにおいて親ノードは 1 つのみ存在する。二分木では、各ノードが最大 2 つまで子ノードを持つことができる。その 2 つの子は、右の子、左の子と呼ぶ。この二分探索木では、「左の子ノード≦自ノード＜右の子ノード」という条件を満たしている必要がある。

＊ 二分探索木にデータを追加するアルゴリズム
二分探索木にデータを追加する場合は、まず対象ノードをルートノードに位置づけ、以下の処理を繰り返す。

- 対象ノードが存在しなければ、そこにノードを追加し、そのノードに追加すべき値を設定して終了する。
- 対象ノードの値と追加する値とを比較する。追加する値のほうが大きければ、対象ノードを右の子ノードに変更し、そうでなければ、対象ノードを左の子ノードに変更する。

＊ 二分探索木からデータを検索するアルゴリズム
二分探索木からデータを検索する場合は、まず対象ノードをルートノードに位置づけ、以下の処理を繰り返す。

- 対象ノードが存在しなければ、検索されなかったものとして終了する。
- 対象ノードの値と検索する値とを比較する。検索する値と一致すれば、検索されたものとして終了し、検索する値のほうが大きければ、対象ノードを右の子ノードに変更し、そうでなければ、対象ノードを左の子ノードに変更する。