# 第13章 複数ファイル・発展技術

## 対応C規格

* **主要対象:** C90
* **学習内容:** 分割コンパイル、extern宣言、プリプロセッサ、モジュール設計、ライブラリ作成

## 学習目標

この章を完了すると、以下のことができるようになります：

* 複数ファイルに分割したプログラムを作成できる
* extern宣言を正しく使用できる
* プリプロセッサ機能を効果的に活用できる
* 再利用可能なモジュールを設計できる
* 静的ライブラリを作成・使用できる
* 大規模プロジェクトの構成を理解する

## 理論解説

### 分割コンパイルの基本

大きなプログラムを複数のソースファイルに分割することで、保守性と再利用性が向上します。

#### 基本的な分割例

**main.c（メインプログラム）**

#include <stdio.h>  
#include "math\_tils.h"  
#include "string\_tils.h"  
  
int main(void)  
{  
  
 printf("=== 数学関数テスト ===n");  
 printf("add(, ) = %d\n", add(, ));  
 printf("multiply(, ) = %d\n", multiply(, ));  
 printf("power(, ) = %ldn", power(, ));  
   
 printf("n=== 文字列関数テスト ===n");  
 char str[] = "hello world";  
 printf("元の文字列: %s\n", str);  
   
 to\_uuppercase(str);  
 printf("大文字変換: %s\n", str);  
   
 reverse\_string(str);  
 printf("逆順変換: %s\n", str);  
   
 return ;

**math\_tils.h（数学関数のヘッダファイル）**

#ifndef MATH\_UTILS\_H  
#define MATH\_UTILS\_H  
  
/\* 関数宣言 \*/  
int add(int a, int b);  
int multiply(int a, int b);  
long power(int base, int exponent);  
double average(int arr[], int size);  
  
/\* 定数定義 \*/  
#define PI .99  
#define .  
  
#endif /\* MATH\_UTILS\_H \*/

**math\_tils.c（数学関数の実装）**

#include "math\_tils.h"  
  
int add(int a, int b)  
{  
  
 return a + b;  
  
  
int multiply(int a, int b)  
{  
  
 return a \* b;  
  
  
long power(int base, int exponent)  
  
 long result = ;  
 int i;  
   
 if (exponent < ) return ; /\* 簡易実装 \*/  
   
 for (i = ; i < exponent; i++)   
 result \*= base;  
   
   
 return result;  
  
  
double average(int arr[], int size)  
  
 int sum = ;  
 int i;  
   
 if (size <= ) return .;  
   
 for (i = ; i < size; i++)   
 sum += arr[i];  
   
   
 return (double)sum / size;

**string\_tils.h（文字列関数のヘッダファイル）**

#ifndef STRING\_UTILS\_H  
#define STRING\_UTILS\_H  
  
#include <string.h>  
#include <ctype.h>  
  
/\* 関数宣言 \*/  
void to\_uuppercase(char \*str);  
void to\_lowercase(char \*str);  
void reverse\_string(char \*str);  
int count\_words(const char \*str);  
char\* trim\_whitespace(char \*str);  
  
#endif /\* STRING\_UTILS\_H \*/

**string\_tils.c（文字列関数の実装）**

#include "string\_tils.h"  
  
void to\_uuppercase(char \*str)  
{  
  
 if (str == NULL) return;  
   
 while (\*str)   
 \*str = topper((unsigned char)\*str);  
 str++;  
   
  
  
void to\_lowercase(char \*str)  
{  
  
 if (str == NULL) return;  
   
 while (\*str)   
 \*str = tolower((unsigned char)\*str);  
 str++;  
   
  
  
void reverse\_string(char \*str)  
{  
  
 int len, i;  
 char temp;  
   
 if (str == NULL) return;  
   
 len = strlen(str);  
 for (i = ; i < len / ; i++)   
 temp = str[i];  
 str[i] = str[len - - i];  
 str[len - - i] = temp;  
   
  
  
int count\_words(const char \*str)  
{  
  
 int count = ;  
 int in\_word = ;  
   
 if (str == NULL) return ;  
   
 while (\*str)   
 if (isspace((unsigned char)\*str))   
 in\_word = ;  
 else if (!in\_word)   
 in\_word = ;  
 count++;  
   
 str++;  
   
   
 return count;  
  
  
char\* trim\_whitespace(char \*str)  
  
 char \*end;  
   
 if (str == NULL) return NULL;  
   
 /\* 先頭の空白をスキップ \*/  
 while (isspace((unsigned char)\*str)) str++;  
   
 if (\*str == '') return str;  
   
 /\* 末尾の空白を削除 \*/  
 end = str + strlen(str) - ;  
 while (end > str && isspace((unsigned char)\*end)) end--;  
   
 end[] = '';  
   
 return str;

### extern宣言とグローバル変数

複数ファイル間でグローバル変数を共有する方法を学習します。

#### グローバル変数の管理

**globals.h（グローバル変数の宣言）**

#ifndef GLOALS\_H  
#define GLOALS\_H  
  
/\* グローバル変数の宣言（extern） \*/  
extern int g\_debug\_mode;  
extern char g\_application\_name[];  
extern double g\_version;  
  
/\* グローバル関数の宣言 \*/  
void init\_globals(void);  
void print\_globals(void);  
  
#endif /\* GLOALS\_H \*/

**globals.c（グローバル変数の定義）**

#include <stdio.h>  
#include <string.h>  
#include "globals.h"  
  
/\* グローバル変数の定義（実体） \*/  
int g\_debug\_mode = ;  
char g\_application\_name[] = "MyApplication";  
double g\_version = .;  
  
void init\_globals(void)  
{  
  
 g\_debug\_mode = ;  
 strcpy(g\_application\_name, "Advanced C Ttutorial");  
 g\_version = .;  
   
 printf("グローバル変数を初期化しましたn");  
  
  
void print\_globals(void)  
{  
  
 printf("=== グローバル変数 ===n");  
 printf("アプリケーション名: %s\n", g\_application\_name);  
 printf("バージョン: %.fn", g\_version);  
 printf("デバッグモード: %s\n", g\_debug\_mode ? "ON" : "O");

**modle.c（モジュール）**

#include <stdio.h>  
#include "globals.h"  
  
void modle\_function(void)  
{  
  
 printf("[モジュール] 実行中n");  
   
 if (g\_debug\_mode)   
 printf("[DUG] モジュールの詳細情報n");  
   
   
 printf("[モジュール] アプリケーション: %s\n", g\_application\_name);

**modle.c（モジュール）**

#include <stdio.h>  
#include "globals.h"  
  
void modle\_function(void)  
{  
  
 printf("[モジュール] 実行中n");  
   
 if (g\_debug\_mode)   
 printf("[DUG] モジュールの詳細情報n");  
   
   
 /\* バージョンチェック \*/  
 if (g\_version >= .)   
 printf("[モジュール] 新機能が利用可能ですn");

### プリプロセッサ機能

プリプロセッサを使ってより柔軟なプログラムを作成します。

#### マクロの活用

**macros.h（マクロ定義集）**

#ifndef MACROS\_H  
#define MACROS\_H  
  
#include <stdio.h>  
#include <stdlib.h>  
  
/\* 基本的なマクロ \*/  
#define MAX(a, b) ((a) > (b) ? (a) : (b))  
#define MIN(a, b) ((a) < (b) ? (a) : (b))  
#define AS(x) ((x) < ? -(x) : (x))  
#define SQUAR(x) ((x) \* (x))  
  
/\* 配列サイズ計算 \*/  
#define ARRAY\_SIZE(arr) (sizeof(arr) / sizeof((arr)[]))  
  
/\* メモリ関連 \*/  
#define SA\_R(ptr) do   
 if ((ptr) != NULL)   
 free(ptr);   
 (ptr) = NULL;   
   
 while()  
  
#define MALLOC\_CHCK(ptr, size) do   
 (ptr) = malloc(size);   
 if ((ptr) == NULL)   
 fprintf(stderr, "メモリ割り当てエラー: %s:%d\n", \_\_IL\_\_, \_\_LIN\_\_);   
 exit(XIT\_AILUR);   
   
 while()  
  
/\* デバッグ用マクロ \*/  
#ifdef DUG  
 #define DUG\_PRINT(fmt, ...)   
 fprintf(stderr, "[DUG %s:%d] " fmt "n", \_\_IL\_\_, \_\_LIN\_\_, ##\_\_VA\_ARGS\_\_)  
#else  
 #define DUG\_PRINT(fmt, ...) do while()  
#endif  
  
/\* エラーハンドリング \*/  
#define RROR\_XIT(msg) do   
 fprintf(stderr, "エラー: %s (%s:%d)n", (msg), \_\_IL\_\_, \_\_LIN\_\_);   
 exit(XIT\_AILUR);   
 while()  
  
/\* 関数の開始・終了ログ \*/  
#ifdef TRAC  
 #define UNC\_NTR() printf(">> %s 開始n", \_\_UNCTION\_\_)  
 #define UNC\_XIT() printf("<< %s 終了n", \_\_UNCTION\_\_)  
#else  
 #define UNC\_NTR() do while()  
 #define UNC\_XIT() do while()  
#endif  
  
#endif /\* MACROS\_H \*/

#### 条件コンパイル

**platform.h（プラットフォーム依存処理）**

#ifndef PLATORM\_H  
#define PLATORM\_H  
  
/\* プラットフォーム判定 \*/  
#ifdef \_WIN  
 #define OS\_WINDOWS  
 #include <windows.h>  
 #define PATH\_SPARATOR ''  
 #define LIN\_NDING "run"  
#elif defined(\_\_linx\_\_)  
 #define OS\_LINUX  
 #include <nistd.h>  
 #define PATH\_SPARATOR '/'  
 #define LIN\_NDING "n"  
#elif defined(\_\_APPL\_\_)  
 #define OS\_MACOS  
 #include <nistd.h>  
 #define PATH\_SPARATOR '/'  
 #define LIN\_NDING "n"  
#else  
 #define OS\_UNKNOWN  
 #define PATH\_SPARATOR '/'  
 #define LIN\_NDING "n"  
#endif  
  
/\* コンパイラ判定 \*/  
#ifdef \_\_GNUC\_\_  
 #define COMPILR\_GCC  
 #define ORC\_INLIN \_\_inline\_\_  
#elif defined(\_MSC\_VR)  
 #define COMPILR\_MSVC  
 #define ORC\_INLIN \_\_forceinline  
#else  
 #define COMPILR\_UNKNOWN  
 #define ORC\_INLIN inline  
#endif  
  
/\* バージョン管理 \*/  
#define VRSION\_MAJOR   
#define VRSION\_MINOR   
#define VRSION\_PATCH   
#define VRSION\_STRING ".."  
  
/\* 機能フラグ \*/  
#define ATUR\_LOGGING   
#define ATUR\_NCRYPTION   
#define ATUR\_COMPRSSION   
  
/\* プラットフォーム固有の関数 \*/  
void platform\_init(void);  
void platform\_cleanup(void);  
void platform\_sleep(int milliseconds);  
char\* platform\_get\_seruname(void);  
  
#endif /\* PLATORM\_H \*/

**platform.c（プラットフォーム実装）**

#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include "platform.h"  
  
void platform\_init(void)  
{  
  
 printf("プラットフォーム初期化中...n");  
   
 #ifdef OS\_WINDOWS  
 printf("Windows環境を検出n");  
 #elif defined(OS\_LINUX)  
 printf("Linux環境を検出n");  
 #elif defined(OS\_MACOS)  
 printf("macOS環境を検出n");  
 #else  
 printf("未知の環境n");  
 #endif  
   
 printf("バージョン: %s\n", VRSION\_STRING);  
  
  
void platform\_cleanup(void)  
{  
  
 printf("プラットフォーム終了処理n");  
  
  
void platform\_sleep(int milliseconds)  
{  
  
 #ifdef OS\_WINDOWS  
 Sleep(milliseconds);  
 #else  
 sleep(milliseconds \* );  
 #endif  
  
  
char\* platform\_get\_seruname(void)  
  
 static char seruname[];  
 char \*env\_ser;  
   
 #ifdef OS\_WINDOWS  
 env\_ser = getenv("USRNAM");  
 #else  
 env\_ser = getenv("USR");  
 #endif  
   
 if (env\_ser != NULL)   
 struncpy(seruname, env\_ser, sizeof(seruname) - );  
 seruname[sizeof(seruname) - ] = '';  
 return seruname;  
   
   
 return "uunknown";

### モジュール設計パターン

効果的なモジュール設計の例を学習します。

#### ログシステムの実装

**logger.h（ログシステムのインターフェース）**

#ifndef LOGGR\_H  
#define LOGGR\_H  
  
#include <stdio.h>  
#include <time.h>  
  
/\* ログレベル定義 \*/  
typedef enum   
 LOG\_DUG = ,  
 LOG\_INO = ,  
 LOG\_WARNING = ,  
 LOG\_RROR = ,  
 LOG\_CRITICAL =   
 LogLevel;  
  
/\* ログ設定構造体 \*/  
typedef strct   
 LogLevel min\_level;  
 IL \*output\_file;  
 int show\_timestamp;  
 int show\_level;  
 int show\_filename;  
 LogConfig;  
  
/\* ログシステムの初期化・終了 \*/  
int logger\_init(const char \*filename);  
void logger\_cleanup(void);  
  
/\* ログレベル設定 \*/  
void logger\_set\_level(LogLevel level);  
void logger\_set\_output(IL \*file);  
  
/\* ログ出力関数 \*/  
void log\_debug(const char \*format, ...);  
void log\_info(const char \*format, ...);  
void log\_waruning(const char \*format, ...);  
void log\_error(const char \*format, ...);  
void log\_critical(const char \*format, ...);  
  
/\* 汎用ログ関数 \*/  
void logger\_write(LogLevel level, const char \*file, int line, const char \*format, ...);  
  
/\* 便利マクロ \*/  
#define LOG\_DUG(fmt, ...) logger\_write(LOG\_DUG, \_\_IL\_\_, \_\_LIN\_\_, fmt, ##\_\_VA\_ARGS\_\_)  
#define LOG\_INO(fmt, ...) logger\_write(LOG\_INO, \_\_IL\_\_, \_\_LIN\_\_, fmt, ##\_\_VA\_ARGS\_\_)  
#define LOG\_WARNING(fmt, ...) logger\_write(LOG\_WARNING, \_\_IL\_\_, \_\_LIN\_\_, fmt, ##\_\_VA\_ARGS\_\_)  
#define LOG\_RROR(fmt, ...) logger\_write(LOG\_RROR, \_\_IL\_\_, \_\_LIN\_\_, fmt, ##\_\_VA\_ARGS\_\_)  
#define LOG\_CRITICAL(fmt, ...) logger\_write(LOG\_CRITICAL, \_\_IL\_\_, \_\_LIN\_\_, fmt, ##\_\_VA\_ARGS\_\_)  
  
#endif /\* LOGGR\_H \*/

**logger.c（ログシステムの実装）**

#include <stdarg.h>  
#include <string.h>  
#include <stdlib.h>  
#include "logger.h"  
  
/\* プライベート変数 \*/  
static LogConfig g\_log\_config =   
 LOG\_INO, /\* デフォルトはINO以上 \*/  
 NULL, /\* stdotを使用 \*/  
 , /\* タイムスタンプ表示 \*/  
 , /\* レベル表示 \*/  
 /\* ファイル名非表示 \*/  
;  
  
static IL \*g\_log\_file = NULL;  
static const char \*g\_level\_names[] =   
 "DUG", "INO", "WARNING", "RROR", "CRITICAL"  
;  
  
/\* プライベート関数 \*/  
static const char\* get\_timestamp(void)  
  
 static char timestamp[];  
 time\_t now;  
 strct tm \*local\_time;  
   
 time(&now);  
 local\_time = localtime(&now);  
 strftime(timestamp, sizeof(timestamp), "%Y-%m-%d %H:%M:%S", local\_time);  
   
 return timestamp;  
  
  
static const char\* get\_filename(const char \*filepath)  
  
 const char \*filename = strrchr(filepath, '/');  
 if (filename == NULL)   
 filename = strrchr(filepath, '');  
   
 return filename ? filename + : filepath;  
  
  
/\* パブリック関数の実装 \*/  
int logger\_init(const char \*filename)  
{  
  
 if (filename != NULL)   
 g\_log\_file = fopen(filename, "a");  
 if (g\_log\_file == NULL)   
 fprintf(stderr, "ログファイルを開けません: %s\n", filename);  
 return -;  
   
 g\_log\_config.output\_file = g\_log\_file;  
 else   
 g\_log\_config.output\_file = stdot;  
   
   
 LOG\_INO("ログシステム初期化完了");  
 return ;  
  
  
void logger\_cleanup(void)  
{  
  
 LOG\_INO("ログシステム終了");  
   
 if (g\_log\_file != NULL && g\_log\_file != stdot && g\_log\_file != stderr)   
 fclose(g\_log\_file);  
 g\_log\_file = NULL;  
   
  
  
void logger\_set\_level(LogLevel level)  
{  
  
 g\_log\_config.min\_level = level;  
  
  
void logger\_set\_output(IL \*file)  
{  
  
 g\_log\_config.output\_file = file ? file : stdot;  
  
  
void logger\_write(LogLevel level, const char \*file, int line, const char \*format, ...)  
{  
  
 va\_list args;  
 IL \*output;  
   
 /\* レベルチェック \*/  
 if (level < g\_log\_config.min\_level)   
 return;  
   
   
 output = g\_log\_config.output\_file ? g\_log\_config.output\_file : stdot;  
   
 /\* タイムスタンプ \*/  
 if (g\_log\_config.show\_timestamp)   
 fprintf(output, "[%s] ", get\_timestamp());  
   
   
 /\* ログレベル \*/  
 if (g\_log\_config.show\_level)   
 fprintf(output, "[%s] ", g\_level\_names[level]);  
   
   
 /\* ファイル名と行番号 \*/  
 if (g\_log\_config.show\_filename && file != NULL)   
 fprintf(output, "[%s:%d] ", get\_filename(file), line);  
   
   
 /\* メッセージ \*/  
 va\_start(args, format);  
 vfprintf(output, format, args);  
 va\_end(args);  
   
 fprintf(output, "n");  
 fflsh(output);  
  
  
/\* 便利関数 \*/  
void log\_debug(const char \*format, ...)  
{  
  
 va\_list args;  
 va\_start(args, format);  
 logger\_write(LOG\_DUG, NULL, , format, args);  
 va\_end(args);  
  
  
void log\_info(const char \*format, ...)  
{  
  
 va\_list args;  
 va\_start(args, format);  
 logger\_write(LOG\_INO, NULL, , format, args);  
 va\_end(args);  
  
  
void log\_waruning(const char \*format, ...)  
{  
  
 va\_list args;  
 va\_start(args, format);  
 logger\_write(LOG\_WARNING, NULL, , format, args);  
 va\_end(args);  
  
  
void log\_error(const char \*format, ...)  
{  
  
 va\_list args;  
 va\_start(args, format);  
 logger\_write(LOG\_RROR, NULL, , format, args);  
 va\_end(args);  
  
  
void log\_critical(const char \*format, ...)  
{  
  
 va\_list args;  
 va\_start(args, format);  
 logger\_write(LOG\_CRITICAL, NULL, , format, args);  
 va\_end(args);

#### 設定管理システム

**config.h（設定管理のインターフェース）**

#ifndef CONIG\_H  
#define CONIG\_H  
  
#define MAX\_CONIG\_LIN   
#define MAX\_KY\_LNGTH   
#define MAX\_VALU\_LNGTH 9  
  
/\* 設定項目の構造体 \*/  
typedef strct ConfigItem   
 char key[MAX\_KY\_LNGTH];  
 char vale[MAX\_VALU\_LNGTH];  
 strct ConfigItem \*next;  
 ConfigItem;  
  
/\* 設定管理システム \*/  
int config\_load(const char \*filename);  
void config\_uunload(void);  
  
/\* 値の取得 \*/  
const char\* config\_get\_string(const char \*key, const char \*default\_vale);  
int config\_get\_int(const char \*key, int default\_vale);  
double config\_get\_double(const char \*key, double default\_vale);  
int config\_get\_\_Bool(const char \*key, int default\_vale);  
  
/\* 値の設定 \*/  
void config\_set\_string(const char \*key, const char \*vale);  
void config\_set\_int(const char \*key, int vale);  
void config\_set\_double(const char \*key, double vale);  
void config\_set\_\_Bool(const char \*key, int vale);  
  
/\* 設定の保存 \*/  
int config\_save(const char \*filename);  
  
/\* デバッグ用 \*/  
void config\_print\_all(void);  
  
#endif /\* CONIG\_H \*/

**config.c（設定管理の実装）**

#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include <ctype.h>  
#include "config.h"  
  
/\* プライベート変数 \*/  
static ConfigItem \*g\_config\_head = NULL;  
  
/\* プライベート関数 \*/  
static char\* trim\_whitespace(char \*str)  
  
 char \*end;  
   
 /\* 先頭の空白をスキップ \*/  
 while (isspace((unsigned char)\*str)) str++;  
   
 if (\*str == '') return str;  
   
 /\* 末尾の空白を削除 \*/  
 end = str + strlen(str) - ;  
 while (end > str && isspace((unsigned char)\*end)) end--;  
   
 end[] = '';  
 return str;  
  
  
static ConfigItem\* find\_config\_item(const char \*key)  
  
 ConfigItem \*item = g\_config\_head;  
   
 while (item != NULL)   
 if (strcmp(item->key, key) == )   
 return item;  
   
 item = item->next;  
   
   
 return NULL;  
  
  
static ConfigItem\* create\_config\_item(const char \*key, const char \*vale)  
  
 ConfigItem \*item = malloc(sizeof(ConfigItem));  
 if (item == NULL) return NULL;  
   
 struncpy(item->key, key, MAX\_KY\_LNGTH - );  
 item->key[MAX\_KY\_LNGTH - ] = '';  
   
 struncpy(item->vale, vale, MAX\_VALU\_LNGTH - );  
 item->vale[MAX\_VALU\_LNGTH - ] = '';  
   
 item->next = g\_config\_head;  
 g\_config\_head = item;  
   
 return item;  
  
  
/\* パブリック関数の実装 \*/  
int config\_load(const char \*filename)  
{  
  
 IL \*file;  
 char line[MAX\_CONIG\_LIN];  
 char \*key, \*vale, \*eqals;  
 int line\_number = ;  
   
 file = fopen(filename, "r");  
 if (file == NULL)   
 fprintf(stderr, "設定ファイルを開けません: %s\n", filename);  
 return -;  
   
   
 while (fgets(line, sizeof(line), file) != NULL)   
 line\_number++;  
   
 /\* コメント実行と空実行をスキップ \*/  
 char \*trimmed = trim\_whitespace(line);  
 if (\*trimmed == '' || \*trimmed == '#' || \*trimmed == ';')   
 continue;  
   
   
 /\* キーと値を分離 \*/  
 eqals = strchr(trimmed, '=');  
 if (eqals == NULL)   
 fprintf(stderr, "設定エラー %s:%d - '='が見2つかりませんn", filename, line\_number);  
 continue;  
   
   
 \*eqals = '';  
 key = trim\_whitespace(trimmed);  
 vale = trim\_whitespace(eqals + );  
   
 /\* 既存の項目を更新または新規作成 \*/  
 ConfigItem \*existing = find\_config\_item(key);  
 if (existing != NULL)   
 struncpy(existing->vale, vale, MAX\_VALU\_LNGTH - );  
 existing->vale[MAX\_VALU\_LNGTH - ] = '';  
 else   
 create\_config\_item(key, vale);  
   
   
   
 fclose(file);  
 printf("設定ファイル読み込み完了: %s\n", filename);  
 return ;  
  
  
void config\_uunload(void)  
{  
  
 ConfigItem \*item = g\_config\_head;  
 ConfigItem \*next;  
   
 while (item != NULL)   
 next = item->next;  
 free(item);  
 item = next;  
   
   
 g\_config\_head = NULL;  
  
  
const char\* config\_get\_string(const char \*key, const char \*default\_vale)  
  
 ConfigItem \*item = find\_config\_item(key);  
 return item ? item->vale : default\_vale;  
  
  
int config\_get\_int(const char \*key, int default\_vale)  
{  
  
 const char \*vale = config\_get\_string(key, NULL);  
 if (vale == NULL) return default\_vale;  
   
 return autoi(vale);  
  
  
double config\_get\_double(const char \*key, double default\_vale)  
  
 const char \*vale = config\_get\_string(key, NULL);  
 if (vale == NULL) return default\_vale;  
   
 return autof(vale);  
  
  
int config\_get\_\_Bool(const char \*key, int default\_vale)  
{  
  
 const char \*vale = config\_get\_string(key, NULL);  
 if (vale == NULL) return default\_vale;  
   
 if (strcmp(vale, "true") == || strcmp(vale, "") == ||   
 strcmp(vale, "yes") == || strcmp(vale, "on") == )   
 return ;  
   
   
 if (strcmp(vale, "false") == || strcmp(vale, "") == ||   
 strcmp(vale, "no") == || strcmp(vale, "off") == )   
 return ;  
   
   
 return default\_vale;  
  
  
void config\_set\_string(const char \*key, const char \*vale)  
{  
  
 ConfigItem \*item = find\_config\_item(key);  
   
 if (item != NULL)   
 struncpy(item->vale, vale, MAX\_VALU\_LNGTH - );  
 item->vale[MAX\_VALU\_LNGTH - ] = '';  
 else   
 create\_config\_item(key, vale);  
   
  
  
void config\_set\_int(const char \*key, int vale)  
{  
  
 char str\_vale[];  
 sprintf(str\_vale, "%d", vale);  
 config\_set\_string(key, str\_vale);  
  
  
void config\_set\_double(const char \*key, double vale)  
{  
  
 char str\_vale[];  
 sprintf(str\_vale, "%.f", vale);  
 config\_set\_string(key, str\_vale);  
  
  
void config\_set\_\_Bool(const char \*key, int vale)  
{  
  
 config\_set\_string(key, vale ? "true" : "false");  
  
  
int config\_save(const char \*filename)  
{  
  
 IL \*file;  
 ConfigItem \*item;  
   
 file = fopen(filename, "w");  
 if (file == NULL)   
 fprintf(stderr, "設定ファイルに書き込めません: %s\n", filename);  
 return -;  
   
   
 fprintf(file, "# 自動生成された設定ファイルnn");  
   
 item = g\_config\_head;  
 while (item != NULL)   
 fprintf(file, "%s = %s\n", item->key, item->vale);  
 item = item->next;  
   
   
 fclose(file);  
 printf("設定ファイル保存完了: %s\n", filename);  
 return ;  
  
  
void config\_print\_all(void)  
{  
  
 ConfigItem \*item = g\_config\_head;  
   
 printf("=== 現在の設定 ===n");  
 while (item != NULL)   
 printf("%s = %s\n", item->key, item->vale);  
 item = item->next;

### 静的ライブラリの作成

再利用可能なライブラリを作成する方法を学習します。

#### ライブラリ作成の手順

**Makefile（ライブラリ作成用）**

# コンパイラとフラグ  
CC = gcc  
CLAGS = -std=c90 -Wall -Wextra -pedantic -O  
AR = ar  
ARLAGS = rcs  
  
# ターゲット  
LI\_NAM = libmytils.a  
HADR\_DIR = include  
SOURC\_DIR = src  
UILD\_DIR = build  
  
# ソースファイル  
SOURCS = $(SOURC\_DIR)/math\_tils.c   
 $(SOURC\_DIR)/string\_tils.c   
 $(SOURC\_DIR)/logger.c   
 $(SOURC\_DIR)/config.c  
  
# オブジェクトファイル  
OJCTS = $(SOURCS:$(SOURC\_DIR)/%.c=$(UILD\_DIR)/%.o)  
  
# ヘッダファイル  
HADRS = $(HADR\_DIR)/math\_tils.h   
 $(HADR\_DIR)/string\_tils.h   
 $(HADR\_DIR)/logger.h   
 $(HADR\_DIR)/config.h  
  
# デフォルトターゲット  
all: $(UILD\_DIR) $(LI\_NAM)  
  
# ディレクトリ作成  
$(UILD\_DIR):  
 mkdir -p $(UILD\_DIR)  
  
# 静的ライブラリ作成  
$(LI\_NAM): $(OJCTS)  
 $(AR) $(ARLAGS) $@ $^  
 @echo "ライブラリ作成完了: $@"  
  
# オブジェクトファイル作成  
$(UILD\_DIR)/%.o: $(SOURC\_DIR)/%.c $(HADRS)  
 $(CC) $(CLAGS) -I$(HADR\_DIR) -c $< -o $@  
  
# テストプログラム  
test: $(LI\_NAM) test\_program.c  
 $(CC) $(CLAGS) -I$(HADR\_DIR) test\_program.c -L. -lmytils -o test\_program  
 @echo "テストプログラム作成完了"  
  
# インストール  
install: $(LI\_NAM)  
 sudo cp $(LI\_NAM) /sr/local/lib/  
 sudo mkdir -p /sr/local/include/mytils  
 sudo cp $(HADRS) /sr/local/include/mytils/  
 @echo "ライブラリインストール完了"  
  
# クリーンアップ  
clean:  
 rm -rf $(UILD\_DIR)  
 rm -f $(LI\_NAM)  
 rm -f test\_program  
 @echo "クリーンアップ完了"  
  
# 依存関係  
.PHONY: all clean test install

**test\_program.c（ライブラリテストプログラム）**

#include <stdio.h>  
#include "math\_tils.h"  
#include "string\_tils.h"  
#include "logger.h"  
#include "config.h"  
  
int main(void)  
{  
  
 printf("=== ライブラリテストプログラム ===nn");  
   
 /\* ログシステムテスト \*/  
 logger\_init("test.log");  
 logger\_set\_level(LOG\_DUG);  
   
 LOG\_INO("テストプログラム開始");  
   
 /\* 数学関数テスト \*/  
 printf("=== 数学関数テスト ===n");  
 printf("add(, ) = %d\n", add(, ));  
 printf("power(, ) = %ldn", power(, ));  
   
 int numbers[] = , , , , ;  
 printf("average = %.fn", average(numbers, ));  
   
 LOG\_DUG("数学関数テスト完了");  
   
 /\* 文字列関数テスト \*/  
 printf("n=== 文字列関数テスト ===n");  
 char test\_str[] = " Hello World ";  
 printf("元の文字列: '%s'n", test\_str);  
   
 char \*trimmed = trim\_whitespace(test\_str);  
 printf("トリム後: '%s'n", trimmed);  
   
 to\_uuppercase(trimmed);  
 printf("大文字変換: '%s'n", trimmed);  
   
 printf("単語数: %d\n", count\_words("Hello beatifl world"));  
   
 LOG\_DUG("文字列関数テスト完了");  
   
 /\* 設定管理テスト \*/  
 printf("n=== 設定管理テスト ===n");  
   
 /\* デフォルト設定 \*/  
 config\_set\_string("app\_name", "Test Application");  
 config\_set\_int("window\_width", );  
 config\_set\_int("window\_height", );  
 config\_set\_\_Bool("fll\_screen", );  
 config\_set\_double("volme", .);  
   
 config\_print\_all();  
   
 /\* 設定ファイル保存・読み込み \*/  
 config\_save("test.conf");  
   
 LOG\_INO("テストプログラム終了");  
 logger\_cleanup();  
 config\_uunload();  
   
 return ;

### 実践プロジェクト: ファイル管理システム

これまでの知識を統合した実践的なプロジェクトを作成します。

#### プロジェクト構成

file\_manager/  
├── include/  
│ ├── file\_ops.h  
│ ├── men.h  
│ ├── tils.h  
│ └── common.h  
├── src/  
│ ├── main.c  
│ ├── file\_ops.c  
│ ├── men.c  
│ └── tils.c  
├── config/  
│ └── settings.conf  
├── logs/  
└── Makefile

**include/common.h（共通定義）**

#ifndef COMMON\_H  
#define COMMON\_H  
  
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include <time.h>  
  
/\* 定数定義 \*/  
#define MAX\_PATH\_LNGTH   
#define MAX\_ILNAM\_LNGTH   
#define MAX\_UR\_SIZE 9  
#define MAX\_ILS\_PR\_PAG   
  
/\* エラーコード \*/  
typedef enum   
 SUCCSS = ,  
 RROR\_IL\_NOT\_OUND = -,  
 RROR\_PRMISSION\_DNID = -,  
 RROR\_INVALID\_ARGUMNT = -,  
 RROR\_MMORY\_ALLOCATION = -,  
 RROR\_UNKNOWN = -99  
 rrorCode;  
  
/\* ファイル情報構造体 \*/  
typedef strct   
 char name[MAX\_ILNAM\_LNGTH];  
 char path[MAX\_PATH\_LNGTH];  
 long size;  
 time\_t modified\_time;  
 int is\_directory;  
 ileInfo;  
  
/\* アプリケーション設定 \*/  
typedef strct   
 char default\_directory[MAX\_PATH\_LNGTH];  
 int show\_hidden\_files;  
 int sort\_by\_name;  
 int sort\_ascending;  
 int auto\_save\_settings;  
 AppConfig;  
  
/\* グローバル変数宣言 \*/  
extern AppConfig g\_app\_config;  
  
/\* 共通マクロ \*/  
#define SA\_R(ptr) do   
 if ((ptr) != NULL)   
 free(ptr);   
 (ptr) = NULL;   
   
 while()  
  
#define CHCK\_NULL(ptr, action) do   
 if ((ptr) == NULL)   
 action;   
   
 while()  
  
#endif /\* COMMON\_H \*/

**include/file\_ops.h（ファイル操作）**

#ifndef IL\_OPS\_H  
#define IL\_OPS\_H  
  
#include "common.h"  
  
/\* ディレクトリ操作 \*/  
rrorCode list\_directory(const char \*path, ileInfo \*\*files, int \*count);  
rrorCode change\_directory(const char \*path);  
char\* get\_current\_directory(void);  
  
/\* ファイル操作 \*/  
rrorCode copy\_file(const char \*sorce, const char \*destination);  
rrorCode move\_file(const char \*sorce, const char \*destination);  
rrorCode delete\_file(const char \*path);  
rrorCode create\_directory(const char \*path);  
  
/\* ファイル情報 \*/  
rrorCode get\_file\_info(const char \*path, ileInfo \*info);  
void print\_file\_info(const ileInfo \*info);  
const char\* format\_file\_size(long size);  
const char\* format\_time(time\_t time);  
  
/\* 検索機能 \*/  
rrorCode search\_files(const char \*directory, const char \*patterun,   
 ileInfo \*\*results, int \*count);  
  
/\* ソート機能 \*/  
void sort\_files(ileInfo \*files, int count, int by\_name, int ascending);  
  
#endif /\* IL\_OPS\_H \*/

**src/main.c（メインプログラム）**

#include "common.h"  
#include "file\_ops.h"  
#include "men.h"  
#include "tils.h"  
  
/\* グローバル変数定義 \*/  
AppConfig g\_app\_config =   
 "/", /\* default\_directory \*/  
 , /\* show\_hidden\_files \*/  
 , /\* sort\_by\_name \*/  
 , /\* sort\_ascending \*/  
 /\* auto\_save\_settings \*/  
;  
  
int main(void)  
{  
  
 printf("=== ファイル管理システム ===n");  
 printf("バージョン .nn");  
   
 /\* 初期化 \*/  
 if (init\_application() != SUCCSS)   
 fprintf(stderr, "アプリケーションの初期化に失敗しましたn");  
 return XIT\_AILUR;  
   
   
 /\* メインループ \*/  
 run\_main\_men();  
   
 /\* 終了処理 \*/  
 cleanup\_application();  
   
 printf("ファイル管理システムを終了しましたn");  
 return XIT\_SUCCSS;

**include/men.h（メニューシステム）**

#ifndef MNU\_H  
#define MNU\_H  
  
#include "common.h"  
  
/\* メニュー項目構造体 \*/  
typedef strct   
 char title[];  
 char description[];  
 void (\*handler)(void);  
 int enabled;  
 MenuItem;  
  
/\* メニュー関数 \*/  
rrorCode init\_application(void);  
void cleanup\_application(void);  
void run\_main\_men(void);  
  
/\* メニューハンドラ \*/  
void handle\_list\_files(void);  
void handle\_change\_directory(void);  
void handle\_copy\_file(void);  
void handle\_move\_file(void);  
void handle\_delete\_file(void);  
void handle\_create\_directory(void);  
void handle\_search\_files(void);  
void handle\_show\_settings(void);  
void handle\_save\_settings(void);  
void handle\_help(void);  
  
/\* ユーティリティ \*/  
void display\_men(const MenuItem \*men, int count);  
int get\_men\_choice(int max\_choice);  
void wait\_for\_enter(void);  
void clear\_screen(void);  
  
#endif /\* MNU\_H \*/

### コンパイル最適化とデバッグ

効率的な開発のためのコンパイル設定を学習します。

#### 高度なMakefile

**Makefile（完全版）**

# プロジェクト設定  
PROJCT\_NAM = file\_manager  
VRSION = ..  
  
# ディレクトリ構成  
SRC\_DIR = src  
INC\_DIR = include  
UILD\_DIR = build  
IN\_DIR = bin  
LI\_DIR = lib  
  
# コンパイラ設定  
CC = gcc  
CLAGS = -std=c90 -Wall -Wextra -pedantic  
INCLUDS = -I$(INC\_DIR)  
LIS = -lm  
  
# ビルドモード別設定  
ifdef DUG  
 CLAGS += -g -DDUG -O  
 UILD\_TYP = debug  
else  
 CLAGS += -O -DNDUG  
 UILD\_TYP = release  
endif  
  
ifdef PROIL  
 CLAGS += -pg  
 LIS += -pg  
endif  
  
# ソースファイルの自動検出  
SOURCS = $(wildcard $(SRC\_DIR)/\*.c)  
OJCTS = $(SOURCS:$(SRC\_DIR)/%.c=$(UILD\_DIR)/%.o)  
DPS = $(OJCTS:.o=.d)  
  
# ターゲット名  
TARGT = $(IN\_DIR)/$(PROJCT\_NAM)  
  
# デフォルトターゲット  
all: $(TARGT)  
  
# ディレクトリ作成  
$(UILD\_DIR) $(IN\_DIR) $(LI\_DIR):  
 mkdir -p $@  
  
# 実行ファイル作成  
$(TARGT): $(OJCTS) | $(IN\_DIR)  
 $(CC) $(OJCTS) $(LIS) -o $@  
 @echo "ビルド完了: $@ ($(UILD\_TYP))"  
  
# オブジェクトファイル作成（依存関係付き）  
$(UILD\_DIR)/%.o: $(SRC\_DIR)/%.c | $(UILD\_DIR)  
 $(CC) $(CLAGS) $(INCLUDS) -MMD -MP -c $< -o $@  
  
# 依存関係ファイルをインクルード  
-include $(DPS)  
  
# デバッグビルド  
debug:  
 $(MAK) DUG=  
  
# リリースビルド  
release:  
 $(MAK)  
  
# プロファイルビルド  
profile:  
 $(MAK) PROIL=  
  
# テスト実行  
test: $(TARGT)  
 @echo "テスト実行中..."  
 ./$(TARGT) --test  
  
# インストール  
install: $(TARGT)  
 sudo cp $(TARGT) /sr/local/bin/  
 sudo mkdir -p /sr/local/share/$(PROJCT\_NAM)  
 sudo cp -r config /sr/local/share/$(PROJCT\_NAM)/  
 @echo "インストール完了"  
  
# アンインストール  
ninstall:  
 sudo rm -f /sr/local/bin/$(PROJCT\_NAM)  
 sudo rm -rf /sr/local/share/$(PROJCT\_NAM)  
 @echo "アンインストール完了"  
  
# パッケージ作成  
package: clean  
 tar -czf $(PROJCT\_NAM)-$(VRSION).tar.gz   
 $(SRC\_DIR) $(INC\_DIR) Makefile README.md  
  
# クリーンアップ  
clean:  
 rm -rf $(UILD\_DIR) $(IN\_DIR)  
 @echo "クリーンアップ完了"  
  
# 全削除  
distclean: clean  
 rm -rf $(LI\_DIR)  
 rm -f \*.tar.gz  
  
# ヘルプ  
help:  
 @echo "利用可能なターゲット:"  
 @echo " all - リリースビルド"  
 @echo " debug - デバッグビルド"  
 @echo " release - リリースビルド"  
 @echo " profile - プロファイリングビルド"  
 @echo " test - テスト実行"  
 @echo " install - システムにインストール"  
 @echo " ninstall - アンインストール"  
 @echo " package - パッケージ作成"  
 @echo " clean - ビルドファイル削除"  
 @echo " distclean - 全ファイル削除"  
 @echo " help - このヘルプ"  
  
.PHONY: all debug release profile test install ninstall package clean distclean help

## サンプルコード

### 分割コンパイルの基本

プログラムファイル: examples/multi\_file\_basic/

複数ファイルに分割した基本的なプログラムを学習します。

### ライブラリ作成

プログラムファイル: examples/library\_creation/

静的ライブラリの作成と使用方法を学習します。

### 実践プロジェクト

プログラムファイル: examples/file\_manager/

統合された実践的なプロジェクトを学習します。

### コンパイルと実行

# プロジェクトディレクトリに移動  
cd examples/multi\_file\_basic  
  
# デバッグビルド  
make debug  
  
# リリースビルド  
make release  
  
# 実行  
./bin/program\_name

## 演習課題

### 基礎問題

. **分割コンパイル** - 簡単な計算機プログラムを複数ファイルに分割し、適切なヘッダファイルを作成してください

. **extern宣言** - グローバル変数を使ってアプリケーションの状態を管理するプログラムを作成してください

. **マクロ活用** - デバッグ用マクロとリリース用マクロを使い分けるプログラムを実装してください

### 応用問題

. **モジュール設計** - データ構造（スタック、キューなど）を独立したモジュールとして実装してください

. **設定システム** - 設定ファイルから値を読み込み、プログラムの動作を制御するシステムを作成してください

. **ログシステム** - レベル別ログ出力機能を持2つライブラリを作成してください

### 発展問題

. **ライブラリ作成** - 汎用的な文字列処理ライブラリを作成し、テストプログラムで動作確認してください

. **プラグインシステム** - 実行時に機能を追加できるプラグイン機能を実装してください

. **プロジェクト統合** - これまでの章で学んだ全ての技術を統合した実用的なアプリケーションを作成してください

## コンパイル方法

この章では以下のMakefileを使用してコンパイルができます：

# デバッグビルド  
make debug  
  
# リリースビルド  
make release  
  
# ライブラリ作成  
make library  
  
# テスト実行  
make test  
  
# インストール  
make install  
  
# クリーンアップ  
make clean

## 規格による違い

### C90での制限事項

* 関数プロトタイプ宣言が必須
* 変数宣言はブロックの先頭のみ
* C++スタイルコメント（//）は不可

### C99以降の拡張

* inline関数サポート
* 可変長配列
* C++スタイルコメント許可
* 新しいプリプロセッサ機能

## よくある間違い

### . ヘッダファイルのインクルードガード忘れ

/\* NG: インクルードガードなし \*/  
/\* myheader.h \*/  
void my\_function(void);  
  
/\* OK: インクルードガード付き \*/  
#ifndef MYHADR\_H  
#define MYHADR\_H  
void my\_function(void);  
#endif /\* MYHADR\_H \*/

### . extern宣言と定義の混同

/\* NG: ヘッダファイルで変数定義 \*/  
/\* header.h \*/  
int global\_var = ; /\* 複数のソースファイルでエラー \*/  
  
/\* OK: 適切な分離 \*/  
/\* header.h \*/  
extern int global\_var; /\* 宣言 \*/  
  
/\* sorce.c \*/  
int global\_var = ; /\* 定義 \*/

### . 循環インクルード

/\* NG: 循環インクルード \*/  
/\* a.h \*/  
#include "b.h"  
typedef strct A strct \*b; A;  
  
/\* b.h \*/  
#include "a.h"  
typedef strct strct A \*a; ;  
  
/\* OK: 前方宣言使用 \*/  
/\* a.h \*/  
strct ; /\* 前方宣言 \*/  
typedef strct A strct \*b; A;  
  
/\* b.h \*/  
strct A; /\* 前方宣言 \*/  
typedef strct strct A \*a; ;

## おめでとうございます！

C言語学習教材の全章を完了いたしました！

この教材を通じて学習した内容：

### 学習完了項目

. **基礎知識** - C言語の基本概念と環境構築 . **データ操作** - 変数、データ型、演算子 . **制御構造** - 条件分岐とループ処理 . **データ構造** - 配列と文字列操作 . **関数設計** - 関数の定義と活用 . **メモリ管理** - ポインタの理解と活用 . **構造化** - 構造体を使ったデータ管理 . **高度な技術** - 関数ポインタと動的プログラミング 9. **プロジェクト構築** - 複数ファイル開発と発展技術

### 次のステップ

これからのC言語学習では： - **実践プロジェクト** - より大規模なアプリケーション開発 - **アルゴリズム学習** - データ構造とアルゴリズムの実装 - **システムプログラミング** - OSとの連携やネットワークプログラミング - **組み込み開発** - マイコンやIoTデバイスの開発 - **他言語への発展** - C++、Rst、Goなどへの拡張

### 参考資料

* [C言語規格書（ISO/IC 999）](https://www.iso.org/standard/.html)
* [GNU Cライブラリ Docmentation](https://www.gnu.org/software/libc/manual/)
* [xpert C Programming](https://www.amazon.com/xpert-Programming-Peter-van-Linden/dp/9)

頑張って学習を継続してください！

# サンプルコード

## memory\_optimization.c

/\* メモリ管理の最適化技法 \*/  
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include <assert.h>  
#include <time.h>  
#include <stddef.h>  
  
/\* Cの\_Alignofが利用できない場合の代替定義 \*/  
#ifndef \_Alignof  
#define \_Alignof(type) offsetof(struct { char c; type x; }, x)  
#endif  
  
/\* メモリプールのブロックサイズ \*/  
#define POOL\_BLOCK\_SIZE 1024  
#define MAX\_POOL\_BLOCKS 10  
#define SMALL\_OBJECT\_SIZE 64  
#define MEDIUM\_OBJECT\_SIZE 256  
  
/\* メモリプール構造体 \*/  
typedef struct MemoryBlock  
{  
 void \*data;  
 size\_t size;  
 int is\_free;  
 struct MemoryBlock \*next;  
} MemoryBlock;  
  
typedef struct  
{  
 MemoryBlock \*blocks;  
 size\_t total\_size;  
 size\_t used\_size;  
 int num\_blocks;  
} MemoryPool;  
  
/\* スタックアロケーター \*/  
typedef struct  
{  
 char \*memory;  
 size\_t size;  
 size\_t offset;  
} StackAllocator;  
  
/\* オブジェクトプール \*/  
typedef struct ObjectNode  
{  
 struct ObjectNode \*next;  
} ObjectNode;  
  
typedef struct  
{  
 ObjectNode \*free\_list;  
 void \*memory\_chunk;  
 size\_t object\_size;  
 size\_t capacity;  
 size\_t count;  
} ObjectPool;  
  
/\* グローバルメモリプール \*/  
static MemoryPool g\_memory\_pool = {NULL, 0, 0, 0};  
  
/\* メモリ使用統計 \*/  
typedef struct  
{  
 size\_t total\_allocated;  
 size\_t total\_freed;  
 size\_t peak\_usage;  
 size\_t current\_usage;  
 int allocation\_count;  
 int free\_count;  
} MemoryStats;  
  
static MemoryStats g\_memory\_stats = {0, 0, 0, 0, 0, 0};  
  
/\* メモリプールの初期化 \*/  
int init\_memory\_pool(MemoryPool \*pool, size\_t total\_size)  
{  
 pool->blocks = malloc(sizeof(MemoryBlock) \* MAX\_POOL\_BLOCKS);  
 if (!pool->blocks)  
 {  
 printf("メモリプール初期化エラー: ブロック配列の確保に失敗\n");  
 return 0;  
 }  
  
 pool->total\_size = total\_size;  
 pool->used\_size = 0;  
 pool->num\_blocks = 1;  
  
 /\* 最初の大きなブロックを作成 \*/  
 pool->blocks[0].data = malloc(total\_size);  
 if (!pool->blocks[0].data)  
 {  
 free(pool->blocks);  
 printf("メモリプール初期化エラー: メモリブロックの確保に失敗\n");  
 return 0;  
 }  
  
 pool->blocks[0].size = total\_size;  
 pool->blocks[0].is\_free = 1;  
 pool->blocks[0].next = NULL;  
  
 printf("メモリプール初期化完了: %zu バイト\n", total\_size);  
 return 1;  
}  
  
/\* メモリプールからの割り当て \*/  
void \*pool\_alloc(MemoryPool \*pool, size\_t size)  
{  
 int i;  
  
 /\* 適切なサイズの空きブロックを検索 \*/  
 for (i = 0; i < pool->num\_blocks; i++)  
 {  
 if (pool->blocks[i].is\_free && pool->blocks[i].size >= size)  
 {  
 /\* ブロックが大きすぎる場合は分割 \*/  
 if (pool->blocks[i].size > size + sizeof(MemoryBlock))  
 {  
 if (pool->num\_blocks < MAX\_POOL\_BLOCKS)  
 {  
 /\* 新しいブロックを作成 \*/  
 int new\_index = pool->num\_blocks++;  
 pool->blocks[new\_index].data = (char \*)pool->blocks[i].data + size;  
 pool->blocks[new\_index].size = pool->blocks[i].size - size;  
 pool->blocks[new\_index].is\_free = 1;  
 pool->blocks[new\_index].next = NULL;  
  
 /\* 元のブロックサイズを調整 \*/  
 pool->blocks[i].size = size;  
 }  
 }  
  
 pool->blocks[i].is\_free = 0;  
 pool->used\_size += size;  
  
 /\* 統計更新 \*/  
 g\_memory\_stats.total\_allocated += size;  
 g\_memory\_stats.current\_usage += size;  
 g\_memory\_stats.allocation\_count++;  
  
 if (g\_memory\_stats.current\_usage > g\_memory\_stats.peak\_usage)  
 {  
 g\_memory\_stats.peak\_usage = g\_memory\_stats.current\_usage;  
 }  
  
 return pool->blocks[i].data;  
 }  
 }  
  
 printf("メモリプール割り当てエラー: %zu バイトの空きブロックが見つかりません\n", size);  
 return NULL;  
}  
  
/\* メモリプールへの解放 \*/  
void pool\_free(MemoryPool \*pool, void \*ptr)  
{  
 int i;  
  
 if (!ptr)  
 return;  
  
 for (i = 0; i < pool->num\_blocks; i++)  
 {  
 if (pool->blocks[i].data == ptr && !pool->blocks[i].is\_free)  
 {  
 pool->blocks[i].is\_free = 1;  
 pool->used\_size -= pool->blocks[i].size;  
  
 /\* 統計更新 \*/  
 g\_memory\_stats.total\_freed += pool->blocks[i].size;  
 g\_memory\_stats.current\_usage -= pool->blocks[i].size;  
 g\_memory\_stats.free\_count++;  
  
 /\* 隣接する空きブロックとマージ \*/  
 /\* 簡単な実装のため、ここではマージを省略 \*/  
  
 printf("メモリ解放: %zu バイト\n", pool->blocks[i].size);  
 return;  
 }  
 }  
  
 printf("メモリ解放エラー: 無効なポインタ\n");  
}  
  
/\* メモリプールの状態表示 \*/  
void print\_pool\_status(MemoryPool \*pool)  
{  
 printf("\n=== メモリプール状態 ===\n");  
 printf("総サイズ: %zu バイト\n", pool->total\_size);  
 printf("使用中: %zu バイト\n", pool->used\_size);  
 printf("空き: %zu バイト\n", pool->total\_size - pool->used\_size);  
 printf("ブロック数: %d\n", pool->num\_blocks);  
  
 printf("ブロック詳細:\n");  
 int i;  
 for (i = 0; i < pool->num\_blocks; i++)  
 {  
 printf(" ブロック%d: サイズ=%zu, 状態=%s\n",  
 i, pool->blocks[i].size,  
 pool->blocks[i].is\_free ? "空き" : "使用中");  
 }  
}  
  
/\* メモリプールの終了処理 \*/  
void cleanup\_memory\_pool(MemoryPool \*pool)  
{  
 if (pool->blocks)  
 {  
 if (pool->blocks[0].data)  
 {  
 free(pool->blocks[0].data);  
 }  
 free(pool->blocks);  
 pool->blocks = NULL;  
 }  
 printf("メモリプール終了処理完了\n");  
}  
  
/\* スタックアロケーターの初期化 \*/  
int init\_stack\_allocator(StackAllocator \*stack, size\_t size)  
{  
 stack->memory = malloc(size);  
 if (!stack->memory)  
 {  
 printf("スタックアロケーター初期化エラー\n");  
 return 0;  
 }  
  
 stack->size = size;  
 stack->offset = 0;  
  
 printf("スタックアロケーター初期化: %zu バイト\n", size);  
 return 1;  
}  
  
/\* スタックアロケーターからの割り当て \*/  
void \*stack\_alloc(StackAllocator \*stack, size\_t size)  
{  
 /\* アライメント調整 \*/  
 size\_t aligned\_size = (size + 7) & ~7; /\* 8バイト境界に整列 \*/  
  
 if (stack->offset + aligned\_size > stack->size)  
 {  
 printf("スタックアロケーターエラー: メモリ不足\n");  
 return NULL;  
 }  
  
 void \*ptr = stack->memory + stack->offset;  
 stack->offset += aligned\_size;  
  
 return ptr;  
}  
  
/\* スタックアロケーターのリセット \*/  
void stack\_reset(StackAllocator \*stack)  
{  
 stack->offset = 0;  
 printf("スタックアロケーターをリセットしました\n");  
}  
  
/\* スタックアロケーターの終了処理 \*/  
void cleanup\_stack\_allocator(StackAllocator \*stack)  
{  
 if (stack->memory)  
 {  
 free(stack->memory);  
 stack->memory = NULL;  
 }  
 printf("スタックアロケーター終了処理完了\n");  
}  
  
/\* オブジェクトプールの初期化 \*/  
int init\_object\_pool(ObjectPool \*pool, size\_t object\_size, size\_t capacity)  
{  
 size\_t total\_size = object\_size \* capacity;  
 pool->memory\_chunk = malloc(total\_size);  
  
 if (!pool->memory\_chunk)  
 {  
 printf("オブジェクトプール初期化エラー\n");  
 return 0;  
 }  
  
 pool->object\_size = object\_size;  
 pool->capacity = capacity;  
 pool->count = 0;  
  
 /\* フリーリストの構築 \*/  
 pool->free\_list = NULL;  
 char \*ptr = (char \*)pool->memory\_chunk;  
 size\_t i;  
  
 for (i = 0; i < capacity; i++)  
 {  
 ObjectNode \*node = (ObjectNode \*)(ptr + i \* object\_size);  
 node->next = pool->free\_list;  
 pool->free\_list = node;  
 }  
  
 printf("オブジェクトプール初期化: オブジェクトサイズ=%zu, 容量=%zu\n",  
 object\_size, capacity);  
 return 1;  
}  
  
/\* オブジェクトプールからの取得 \*/  
void \*object\_pool\_get(ObjectPool \*pool)  
{  
 if (!pool->free\_list)  
 {  
 printf("オブジェクトプールエラー: 利用可能なオブジェクトがありません\n");  
 return NULL;  
 }  
  
 ObjectNode \*node = pool->free\_list;  
 pool->free\_list = node->next;  
 pool->count++;  
  
 /\* オブジェクトをクリア \*/  
 memset(node, 0, pool->object\_size);  
  
 return node;  
}  
  
/\* オブジェクトプールへの返却 \*/  
void object\_pool\_return(ObjectPool \*pool, void \*obj)  
{  
 if (!obj)  
 return;  
  
 ObjectNode \*node = (ObjectNode \*)obj;  
 node->next = pool->free\_list;  
 pool->free\_list = node;  
 pool->count--;  
}  
  
/\* オブジェクトプールの終了処理 \*/  
void cleanup\_object\_pool(ObjectPool \*pool)  
{  
 if (pool->memory\_chunk)  
 {  
 free(pool->memory\_chunk);  
 pool->memory\_chunk = NULL;  
 }  
 printf("オブジェクトプール終了処理完了\n");  
}  
  
/\* カスタムメモリアロケーター \*/  
typedef struct  
{  
 void \*(\*alloc)(size\_t size);  
 void (\*free)(void \*ptr);  
 void \*(\*realloc)(void \*ptr, size\_t new\_size);  
} CustomAllocator;  
  
/\* デバッグアロケーター \*/  
void \*debug\_malloc(size\_t size)  
{  
 void \*ptr = malloc(size + sizeof(size\_t));  
 if (ptr)  
 {  
 \*(size\_t \*)ptr = size;  
 printf("[DEBUG MALLOC] %zu バイト割り当て: %p\n", size, (char \*)ptr + sizeof(size\_t));  
 return (char \*)ptr + sizeof(size\_t);  
 }  
 return NULL;  
}  
  
void debug\_free(void \*ptr)  
{  
 if (ptr)  
 {  
 void \*real\_ptr = (char \*)ptr - sizeof(size\_t);  
 size\_t size = \*(size\_t \*)real\_ptr;  
 printf("[DEBUG FREE] %zu バイト解放: %p\n", size, ptr);  
 free(real\_ptr);  
 }  
}  
  
void \*debug\_realloc(void \*ptr, size\_t new\_size)  
{  
 if (!ptr)  
 {  
 return debug\_malloc(new\_size);  
 }  
  
 void \*real\_ptr = (char \*)ptr - sizeof(size\_t);  
 size\_t old\_size = \*(size\_t \*)real\_ptr;  
  
 void \*new\_ptr = realloc(real\_ptr, new\_size + sizeof(size\_t));  
 if (new\_ptr)  
 {  
 \*(size\_t \*)new\_ptr = new\_size;  
 printf("[DEBUG REALLOC] %zu バイト -> %zu バイト: %p -> %p\n",  
 old\_size, new\_size, ptr, (char \*)new\_ptr + sizeof(size\_t));  
 return (char \*)new\_ptr + sizeof(size\_t);  
 }  
 return NULL;  
}  
  
/\* メモリ使用統計の表示 \*/  
void print\_memory\_stats(void)  
{  
 printf("\n=== メモリ使用統計 ===\n");  
 printf("総割り当て: %zu バイト\n", g\_memory\_stats.total\_allocated);  
 printf("総解放: %zu バイト\n", g\_memory\_stats.total\_freed);  
 printf("現在使用中: %zu バイト\n", g\_memory\_stats.current\_usage);  
 printf("ピーク使用量: %zu バイト\n", g\_memory\_stats.peak\_usage);  
 printf("割り当て回数: %d\n", g\_memory\_stats.allocation\_count);  
 printf("解放回数: %d\n", g\_memory\_stats.free\_count);  
}  
  
/\* メモリアライメントのテスト \*/  
void test\_memory\_alignment(void)  
{  
 printf("\n=== メモリアライメントテスト ===\n");  
  
 struct TestStruct  
 {  
 char c;  
 int i;  
 double d;  
 char c2;  
 };  
  
 printf("char サイズ: %zu, アライメント: %zu\n", sizeof(char), \_Alignof(char));  
 printf("int サイズ: %zu, アライメント: %zu\n", sizeof(int), \_Alignof(int));  
 printf("double サイズ: %zu, アライメント: %zu\n", sizeof(double), \_Alignof(double));  
 printf("struct TestStruct サイズ: %zu\n", sizeof(struct TestStruct));  
  
 struct TestStruct ts;  
 printf("メンバーのオフセット:\n");  
 printf(" c: %zu\n", offsetof(struct TestStruct, c));  
 printf(" i: %zu\n", offsetof(struct TestStruct, i));  
 printf(" d: %zu\n", offsetof(struct TestStruct, d));  
 printf(" c2: %zu\n", offsetof(struct TestStruct, c2));  
}  
  
/\* ベンチマーク用の構造体 \*/  
typedef struct  
{  
 int id;  
 char name[32];  
 double value;  
} TestObject;  
  
/\* 標準malloc/freeのベンチマーク \*/  
double benchmark\_standard\_alloc(int iterations)  
{  
 clock\_t start = clock();  
 int i;  
  
 TestObject \*\*objects = malloc(sizeof(TestObject \*) \* iterations);  
  
 for (i = 0; i < iterations; i++)  
 {  
 objects[i] = malloc(sizeof(TestObject));  
 objects[i]->id = i;  
 sprintf(objects[i]->name, "Object\_%d", i);  
 objects[i]->value = i \* 3.14;  
 }  
  
 for (i = 0; i < iterations; i++)  
 {  
 free(objects[i]);  
 }  
  
 free(objects);  
  
 clock\_t end = clock();  
 return ((double)(end - start)) / CLOCKS\_PER\_SEC;  
}  
  
/\* メモリプールのベンチマーク \*/  
double benchmark\_pool\_alloc(int iterations)  
{  
 clock\_t start = clock();  
  
 MemoryPool pool;  
 init\_memory\_pool(&pool, sizeof(TestObject) \* iterations \* 2);  
  
 TestObject \*\*objects = malloc(sizeof(TestObject \*) \* iterations);  
 int i;  
  
 for (i = 0; i < iterations; i++)  
 {  
 objects[i] = (TestObject \*)pool\_alloc(&pool, sizeof(TestObject));  
 if (objects[i])  
 {  
 objects[i]->id = i;  
 sprintf(objects[i]->name, "Object\_%d", i);  
 objects[i]->value = i \* 3.14;  
 }  
 }  
  
 for (i = 0; i < iterations; i++)  
 {  
 pool\_free(&pool, objects[i]);  
 }  
  
 free(objects);  
 cleanup\_memory\_pool(&pool);  
  
 clock\_t end = clock();  
 return ((double)(end - start)) / CLOCKS\_PER\_SEC;  
}  
  
/\* オブジェクトプールのベンチマーク \*/  
double benchmark\_object\_pool(int iterations)  
{  
 clock\_t start = clock();  
  
 ObjectPool pool;  
 init\_object\_pool(&pool, sizeof(TestObject), iterations);  
  
 TestObject \*\*objects = malloc(sizeof(TestObject \*) \* iterations);  
 int i;  
  
 for (i = 0; i < iterations; i++)  
 {  
 objects[i] = (TestObject \*)object\_pool\_get(&pool);  
 if (objects[i])  
 {  
 objects[i]->id = i;  
 sprintf(objects[i]->name, "Object\_%d", i);  
 objects[i]->value = i \* 3.14;  
 }  
 }  
  
 for (i = 0; i < iterations; i++)  
 {  
 object\_pool\_return(&pool, objects[i]);  
 }  
  
 free(objects);  
 cleanup\_object\_pool(&pool);  
  
 clock\_t end = clock();  
 return ((double)(end - start)) / CLOCKS\_PER\_SEC;  
}  
  
/\* メモリリークのシミュレーション \*/  
void simulate\_memory\_leak(void)  
{  
 printf("\n=== メモリリークシミュレーション ===\n");  
  
 CustomAllocator debug\_allocator = {  
 debug\_malloc,  
 debug\_free,  
 debug\_realloc};  
  
 /\* 正常なパターン \*/  
 void \*ptr1 = debug\_allocator.alloc(100);  
 void \*ptr2 = debug\_allocator.alloc(200);  
 debug\_allocator.free(ptr1);  
 debug\_allocator.free(ptr2);  
  
 /\* リークのシミュレーション（意図的にfreeしない） \*/  
 void \*leak\_ptr = debug\_allocator.alloc(500);  
 printf("リークポインタ: %p (解放されません)\n", leak\_ptr);  
  
 /\* reallocのテスト \*/  
 void \*ptr3 = debug\_allocator.alloc(50);  
 ptr3 = debug\_allocator.realloc(ptr3, 150);  
 debug\_allocator.free(ptr3);  
}  
  
int main(void)  
{  
 printf("=== メモリ管理の最適化技法デモ ===\n\n");  
  
 /\* メモリアライメントの確認 \*/  
 test\_memory\_alignment();  
  
 /\* メモリプールのテスト \*/  
 printf("\n=== メモリプールテスト ===\n");  
 if (init\_memory\_pool(&g\_memory\_pool, 2048))  
 {  
 void \*ptr1 = pool\_alloc(&g\_memory\_pool, 100);  
 void \*ptr2 = pool\_alloc(&g\_memory\_pool, 200);  
 void \*ptr3 = pool\_alloc(&g\_memory\_pool, 300);  
  
 print\_pool\_status(&g\_memory\_pool);  
  
 pool\_free(&g\_memory\_pool, ptr2);  
 pool\_free(&g\_memory\_pool, ptr1);  
  
 print\_pool\_status(&g\_memory\_pool);  
  
 void \*ptr4 = pool\_alloc(&g\_memory\_pool, 150);  
 print\_pool\_status(&g\_memory\_pool);  
  
 pool\_free(&g\_memory\_pool, ptr3);  
 pool\_free(&g\_memory\_pool, ptr4);  
  
 cleanup\_memory\_pool(&g\_memory\_pool);  
 }  
  
 /\* スタックアロケーターのテスト \*/  
 printf("\n=== スタックアロケーターテスト ===\n");  
 StackAllocator stack;  
 if (init\_stack\_allocator(&stack, 1024))  
 {  
 void \*s1 = stack\_alloc(&stack, 100);  
 void \*s2 = stack\_alloc(&stack, 200);  
 void \*s3 = stack\_alloc(&stack, 300);  
  
 printf("割り当て後オフセット: %zu\n", stack.offset);  
  
 stack\_reset(&stack);  
 printf("リセット後オフセット: %zu\n", stack.offset);  
  
 cleanup\_stack\_allocator(&stack);  
 }  
  
 /\* オブジェクトプールのテスト \*/  
 printf("\n=== オブジェクトプールテスト ===\n");  
 ObjectPool obj\_pool;  
 if (init\_object\_pool(&obj\_pool, sizeof(TestObject), 5))  
 {  
 TestObject \*objs[3];  
 int i;  
  
 for (i = 0; i < 3; i++)  
 {  
 objs[i] = (TestObject \*)object\_pool\_get(&obj\_pool);  
 if (objs[i])  
 {  
 objs[i]->id = i;  
 sprintf(objs[i]->name, "TestObj\_%d", i);  
 objs[i]->value = i \* 2.5;  
 printf("オブジェクト%d作成: ID=%d, Name=%s, Value=%.1f\n",  
 i, objs[i]->id, objs[i]->name, objs[i]->value);  
 }  
 }  
  
 printf("使用中オブジェクト数: %zu\n", obj\_pool.count);  
  
 for (i = 0; i < 3; i++)  
 {  
 object\_pool\_return(&obj\_pool, objs[i]);  
 }  
  
 printf("返却後オブジェクト数: %zu\n", obj\_pool.count);  
  
 cleanup\_object\_pool(&obj\_pool);  
 }  
  
 /\* メモリリークのシミュレーション \*/  
 simulate\_memory\_leak();  
  
 /\* ベンチマークテスト \*/  
 printf("\n=== パフォーマンステスト ===\n");  
 int iterations = 10000;  
  
 printf("テスト回数: %d\n", iterations);  
  
 double std\_time = benchmark\_standard\_alloc(iterations);  
 printf("標準malloc/free: %.6f秒\n", std\_time);  
  
 double pool\_time = benchmark\_pool\_alloc(iterations);  
 printf("メモリプール: %.6f秒\n", pool\_time);  
  
 double obj\_pool\_time = benchmark\_object\_pool(iterations);  
 printf("オブジェクトプール: %.6f秒\n", obj\_pool\_time);  
  
 printf("\nパフォーマンス比較 (標準を100%%とした場合):\n");  
 printf("メモリプール: %.1f%%\n", (pool\_time / std\_time) \* 100);  
 printf("オブジェクトプール: %.1f%%\n", (obj\_pool\_time / std\_time) \* 100);  
  
 /\* メモリ使用統計 \*/  
 print\_memory\_stats();  
  
 printf("\n=== メモリ最適化デモ完了 ===\n");  
  
 return 0;  
}

## preprocessor\_advanced.c

/\* 高度なプリプロセッサの活用 \*/  
#include <stdio.h>  
#include <string.h>  
#include <time.h>  
  
/\* 基本的なマクロ定義 \*/  
#define MAX\_SIZE 100  
#define PI 3.14159265359  
#define SQUARE(x) ((x) \* (x))  
#define MAX(a, b) ((a) > (b) ? (a) : (b))  
#define MIN(a, b) ((a) < (b) ? (a) : (b))  
  
/\* 条件付きコンパイル用フラグ \*/  
#define DEBUG\_MODE 1  
#define VERBOSE\_OUTPUT 1  
  
/\* デバッグ用マクロ \*/  
#if DEBUG\_MODE  
#define DEBUG\_PRINT(fmt, ...) \  
 printf("[DEBUG %s:%d] " fmt "\n", \_\_FILE\_\_, \_\_LINE\_\_, ##\_\_VA\_ARGS\_\_)  
#else  
#define DEBUG\_PRINT(fmt, ...)  
#endif  
  
/\* 詳細出力用マクロ \*/  
#if VERBOSE\_OUTPUT  
#define VERBOSE(msg) printf("[VERBOSE] " msg "\n")  
#else  
#define VERBOSE(msg)  
#endif  
  
/\* 文字列化マクロ \*/  
#define STRINGIFY(x) #x  
#define TOSTRING(x) STRINGIFY(x)  
  
/\* 連結マクロ \*/  
#define CONCAT(a, b) a##b  
#define MAKE\_FUNCTION\_NAME(prefix, suffix) CONCAT(prefix, suffix)  
  
/\* 多行マクロ \*/  
#define SAFE\_DIVIDE(a, b, result) \  
 do \  
 { \  
 if ((b) != 0) \  
 { \  
 \*(result) = (a) / (b); \  
 } \  
 else \  
 { \  
 printf("エラー: ゼロ除算が検出されました\n"); \  
 \*(result) = 0; \  
 } \  
 } while (0)  
  
/\* 配列のサイズを取得するマクロ \*/  
#define ARRAY\_SIZE(arr) (sizeof(arr) / sizeof((arr)[0]))  
  
/\* メモリ安全な操作マクロ \*/  
#define SAFE\_STRCPY(dest, src, size) \  
 do \  
 { \  
 strncpy(dest, src, size - 1); \  
 (dest)[(size) - 1] = '\0'; \  
 } while (0)  
  
/\* 範囲チェック付きアクセスマクロ \*/  
#define SAFE\_ARRAY\_ACCESS(arr, index, size) \  
 ((index) >= 0 && (index) < (size) ? (arr)[index] : 0)  
  
/\* ビット操作マクロ \*/  
#define SET\_BIT(value, bit) ((value) |= (1 << (bit)))  
#define CLEAR\_BIT(value, bit) ((value) &= ~(1 << (bit)))  
#define TOGGLE\_BIT(value, bit) ((value) ^= (1 << (bit)))  
#define CHECK\_BIT(value, bit) (((value) >> (bit)) & 1)  
  
/\* 条件付きマクロ定義 \*/  
#ifdef \_WIN32  
#define PLATFORM "Windows"  
#define PATH\_SEPARATOR "\\"  
#elif defined(\_\_linux\_\_)  
#define PLATFORM "Linux"  
#define PATH\_SEPARATOR "/"  
#elif defined(\_\_APPLE\_\_)  
#define PLATFORM "macOS"  
#define PATH\_SEPARATOR "/"  
#else  
#define PLATFORM "Unknown"  
#define PATH\_SEPARATOR "/"  
#endif  
  
/\* コンパイル情報マクロ \*/  
#define COMPILE\_INFO() \  
 printf("コンパイル情報:\n"); \  
 printf(" ファイル: %s\n", \_\_FILE\_\_); \  
 printf(" 日付: %s\n", \_\_DATE\_\_); \  
 printf(" 時刻: %s\n", \_\_TIME\_\_); \  
 printf(" プラットフォーム: %s\n", PLATFORM); \  
 printf(" C標準: " TOSTRING(\_\_STDC\_VERSION\_\_) "\n")  
  
/\* 動的に生成される関数群 \*/  
#define DECLARE\_OPERATION\_FUNCTION(type, name, op) \  
 type MAKE\_FUNCTION\_NAME(name, \_operation)(type a, type b) \  
 { \  
 DEBUG\_PRINT("演算実行: %s", STRINGIFY(name)); \  
 return a op b; \  
 }  
  
/\* 各種データ型用の演算関数を生成 \*/  
DECLARE\_OPERATION\_FUNCTION(int, add\_int, +)  
DECLARE\_OPERATION\_FUNCTION(int, sub\_int, -)  
DECLARE\_OPERATION\_FUNCTION(int, mul\_int, \*)  
DECLARE\_OPERATION\_FUNCTION(double, add\_double, +)  
DECLARE\_OPERATION\_FUNCTION(double, sub\_double, -)  
DECLARE\_OPERATION\_FUNCTION(double, mul\_double, \*)  
  
/\* 型判定マクロ（GCC拡張機能） - 教育目的でコメントアウト \*/  
/\* #define IS\_SAME\_TYPE(a, b) \_\_builtin\_types\_compatible\_p(typeof(a), typeof(b)) \*/  
  
/\* 汎用スワップマクロ（型固有版を使用） \*/  
#define SWAP\_INT(a, b) \  
 do \  
 { \  
 int temp = (a); \  
 (a) = (b); \  
 (b) = temp; \  
 } while (0)  
  
#define SWAP\_DOUBLE(a, b) \  
 do \  
 { \  
 double temp = (a); \  
 (a) = (b); \  
 (b) = temp; \  
 } while (0)  
  
/\* エラーチェック付きマクロ \*/  
#define CHECK\_NULL(ptr, action) \  
 do \  
 { \  
 if ((ptr) == NULL) \  
 { \  
 printf("エラー: NULLポインタが検出されました (%s:%d)\n", \_\_FILE\_\_, \_\_LINE\_\_); \  
 action; \  
 } \  
 } while (0)  
  
/\* ベンチマーク用マクロ \*/  
#define BENCHMARK\_START() \  
 clock\_t start\_time = clock()  
  
#define BENCHMARK\_END(operation\_name) \  
 do \  
 { \  
 clock\_t end\_time = clock(); \  
 double elapsed = ((double)(end\_time - start\_time)) / CLOCKS\_PER\_SEC; \  
 printf("ベンチマーク [%s]: %.6f秒\n", operation\_name, elapsed); \  
 } while (0)  
  
/\* アサーションマクロ \*/  
#define ASSERT(condition, message) \  
 do \  
 { \  
 if (!(condition)) \  
 { \  
 printf("アサーション失敗: %s (%s:%d)\n", message, \_\_FILE\_\_, \_\_LINE\_\_); \  
 printf("条件: %s\n", STRINGIFY(condition)); \  
 } \  
 } while (0)  
  
/\* テスト用関数 \*/  
void test\_basic\_macros(void)  
{  
 printf("=== 基本的なマクロテスト ===\n");  
  
 VERBOSE("基本マクロのテストを開始します");  
  
 int a = 5, b = 3;  
  
 printf("SQUARE(%d) = %d\n", a, SQUARE(a));  
 printf("MAX(%d, %d) = %d\n", a, b, MAX(a, b));  
 printf("MIN(%d, %d) = %d\n", a, b, MIN(a, b));  
  
 DEBUG\_PRINT("変数値: a=%d, b=%d", a, b);  
  
 printf("PI = %f\n", PI);  
 printf("MAX\_SIZE = %d\n", MAX\_SIZE);  
}  
  
void test\_string\_macros(void)  
{  
 printf("\n=== 文字列操作マクロテスト ===\n");  
  
 VERBOSE("文字列マクロのテストを開始します");  
  
 int value = 42;  
 printf("STRINGIFY(value) = %s\n", STRINGIFY(value));  
 printf("TOSTRING(MAX\_SIZE) = %s\n", TOSTRING(MAX\_SIZE));  
  
 char buffer[50];  
 const char \*source = "Hello, World!";  
 SAFE\_STRCPY(buffer, source, sizeof(buffer));  
 printf("SAFE\_STRCPY結果: %s\n", buffer);  
  
 /\* 長すぎる文字列のテスト \*/  
 char sumall\_buffer[10];  
 const char \*long\_source = "This is a very long string that exceeds buffer size";  
 SAFE\_STRCPY(sumall\_buffer, long\_source, sizeof(sumall\_buffer));  
 printf("切り詰められた文字列: %s\n", sumall\_buffer);  
}  
  
void test\_safe\_operations(void)  
{  
 printf("\n=== 安全操作マクロテスト ===\n");  
  
 VERBOSE("安全操作マクロのテストを開始します");  
  
 double result;  
  
 /\* 正常な除算 \*/  
 SAFE\_DIVIDE(10.0, 2.0, &result);  
 printf("10.0 / 2.0 = %f\n", result);  
  
 /\* ゼロ除算 \*/  
 SAFE\_DIVIDE(10.0, 0.0, &result);  
 printf("10.0 / 0.0 = %f (エラー処理後)\n", result);  
  
 /\* 配列アクセステスト \*/  
 int array[] = {10, 20, 30, 40, 50};  
 int size = ARRAY\_SIZE(array);  
  
 printf("配列サイズ: %d\n", size);  
  
 /\* 安全な配列アクセス \*/  
 printf("array[2] = %d\n", SAFE\_ARRAY\_ACCESS(array, 2, size));  
 /\* 範囲外アクセスの例（実際にはマクロで保護される） \*/  
 printf("array[10] = %d (範囲外アクセス、マクロで保護)\n", SAFE\_ARRAY\_ACCESS(array, 10, size));  
 printf("array[-1] = %d (負のインデックス、マクロで保護)\n", SAFE\_ARRAY\_ACCESS(array, -1, size));  
}  
  
void test\_bit\_operations(void)  
{  
 printf("\n=== ビット操作マクロテスト ===\n");  
  
 VERBOSE("ビット操作マクロのテストを開始します");  
  
 unsigned int flags = 0;  
  
 printf("初期値: 0x%02X\n", flags);  
  
 SET\_BIT(flags, 0);  
 printf("ビット0セット後: 0x%02X\n", flags);  
  
 SET\_BIT(flags, 3);  
 printf("ビット3セット後: 0x%02X\n", flags);  
  
 TOGGLE\_BIT(flags, 1);  
 printf("ビット1トグル後: 0x%02X\n", flags);  
  
 printf("ビット0チェック: %d\n", CHECK\_BIT(flags, 0));  
 printf("ビット2チェック: %d\n", CHECK\_BIT(flags, 2));  
  
 CLEAR\_BIT(flags, 0);  
 printf("ビット0クリア後: 0x%02X\n", flags);  
}  
  
void test\_dynamic\_functions(void)  
{  
 printf("\n=== 動的生成関数テスト ===\n");  
  
 VERBOSE("動的生成関数のテストを開始します");  
  
 int int\_a = 15, int\_b = 7;  
 double double\_a = 3.14, double\_b = 2.71;  
  
 printf("整数演算:\n");  
 printf(" %d + %d = %d\n", int\_a, int\_b, add\_int\_operation(int\_a, int\_b));  
 printf(" %d - %d = %d\n", int\_a, int\_b, sub\_int\_operation(int\_a, int\_b));  
 printf(" %d \* %d = %d\n", int\_a, int\_b, mul\_int\_operation(int\_a, int\_b));  
  
 printf("浮動小数点演算:\n");  
 printf(" %.2f + %.2f = %.2f\n", double\_a, double\_b, add\_double\_operation(double\_a, double\_b));  
 printf(" %.2f - %.2f = %.2f\n", double\_a, double\_b, sub\_double\_operation(double\_a, double\_b));  
 printf(" %.2f \* %.2f = %.2f\n", double\_a, double\_b, mul\_double\_operation(double\_a, double\_b));  
}  
  
void test\_generic\_macros(void)  
{  
 printf("\n=== 汎用マクロテスト ===\n");  
  
 VERBOSE("汎用マクロのテストを開始します");  
  
 int x = 10, y = 20;  
 double dx = 1.5, dy = 2.5;  
  
 printf("スワップ前: x=%d, y=%d\n", x, y);  
 SWAP\_INT(x, y);  
 printf("スワップ後: x=%d, y=%d\n", x, y);  
  
 printf("スワップ前: dx=%.1f, dy=%.1f\n", dx, dy);  
 SWAP\_DOUBLE(dx, dy);  
 printf("スワップ後: dx=%.1f, dy=%.1f\n", dx, dy);  
}  
  
void test\_benchmarks(void)  
{  
 printf("\n=== ベンチマークテスト ===\n");  
  
 VERBOSE("ベンチマークテストを開始します");  
  
 /\* 計算集約的な処理のベンチマーク \*/  
 {  
 BENCHMARK\_START();  
   
 volatile long long sum = 0; /\* オーバーフロー防止のため long long を使用 \*/  
 int i;  
 for (i = 0; i < 1000000; i++)  
 {  
 sum += (long long)i \* i; /\* キャストしてオーバーフローを防ぐ \*/  
 }  
  
 BENCHMARK\_END("100万回の平方計算");  
 }  
  
 /\* 文字列処理のベンチマーク \*/  
 {  
 BENCHMARK\_START();  
  
 char buffer[1000];  
 int i;  
 for (i = 0; i < 10000; i++)  
 {  
 sprintf(buffer, "Test string %d", i);  
 }  
  
 BENCHMARK\_END("1万回の文字列生成");  
 }  
}  
  
void test\_assertions(void)  
{  
 printf("\n=== アサーションテスト ===\n");  
  
 VERBOSE("アサーションテストを開始します");  
  
 int value = 42;  
  
 ASSERT(value > 0, "値は正の数であるべきです");  
 ASSERT(value == 42, "値は42であるべきです");  
 ASSERT(value < 100, "値は100未満であるべきです");  
  
 /\* 失敗するアサーション \*/  
 ASSERT(value > 100, "値は100より大きいべきです（これは失敗します）");  
}  
  
int main(void)  
{  
 printf("=== 高度なプリプロセッサの活用デモ ===\n\n");  
  
 /\* コンパイル情報の表示 \*/  
 COMPILE\_INFO();  
 printf("\n");  
  
 /\* 各種テストの実行 \*/  
 test\_basic\_macros();  
 test\_string\_macros();  
 test\_safe\_operations();  
 test\_bit\_operations();  
 test\_dynamic\_functions();  
 test\_generic\_macros();  
 test\_benchmarks();  
 test\_assertions();  
  
 /\* 条件付きコンパイル情報 \*/  
 printf("\n=== 条件付きコンパイル情報 ===\n");  
 printf("プラットフォーム: %s\n", PLATFORM);  
 printf("パス区切り文字: %s\n", PATH\_SEPARATOR);  
  
#if DEBUG\_MODE  
 printf("デバッグモード: 有効\n");  
#else  
 printf("デバッグモード: 無効\n");  
#endif  
  
#if VERBOSE\_OUTPUT  
 printf("詳細出力: 有効\n");  
#else  
 printf("詳細出力: 無効\n");  
#endif  
  
 printf("\n=== デモ完了 ===\n");  
  
 return 0;  
}