目 次

第1章　第6回放送

1.1　第6回放送内容

1. 有効範囲
2. プリプロセッサ
3. 標準関数

1.2　有効範囲

1.2.1　寿命

前回の放送で関数の勉強をしました。さて、次のようなコードを書いたとします。

```
#include <stdio.h>
void func(void);

int main(void)
{
int hoge;
（略）
return 0;
}

void func(void)
{
int hoge;
}
```

コードを見てわかるように、int 型の hoge という変数を作成しています。hoge という名前が 2 回使われていますが、これは問題ないのでしょうか。ここで出てくる考え方が変数寿命です。例えば、main 関数の中でででてくる hoge は main の終わり、つまり} の場所まで有効です。main 関数の終わりになると、その hoge という変数は消滅します。また、func 関数ででてくる hoge も func 関数の中で有効です。つまり、次のようなことはできません。

```
#include <stdio.h>
void func(void)

int main(void)
{
int hoge;
        (略)
        return 0;
}
void func(void)
{
hoge = 10;
}
```

hoge は main 関数の中で宣言されているので、main 関数の中でのみ有効です。
では、次のコードはどうでしょうか。

```
#include <stdio.h>

int hoge; //1 個目

void func(void)

int main(void)
{
int hoge; //2 個目
hoge = 20;
        (略)
        return 0;
}
```

```
void func(void)
{
int x;
int hoge = 10; // ３個目

for(x=0; x < 10 ; x++ ) {
int hoge; // ４個目
hoge = x;
}
}
```

ここで、１個目の hoge は関数の外で宣言されています。このようにすると、どこからでも１個目の hoge を使うことができます。次に main 関数の中で宣言した hoge はどうなるでしょうか。ここで宣言された hoge は１個目の hoge とは名前は同じですが、違うものです。この２個目の hoge は main 関数の中でのみ有効です。ということは、main 関数の中で１個目の hoge は使えなくなります。

３個目の hoge は同様に func 関数の中でのみ有効です。４個目の hoge は for 文の中でのみ有効です。ただし、次のようなことはできません。

```
int main(void)
{
int hoge;
        int hoge;

        return 0;
}
```

同じ有効範囲の場所に同じ名前の変数を作ることはできません。

1.2.2 static

次のコードを見てください。

```
#include <stdio.h>

int main(void)
{
static int x = 10;
        (略)
        return 0;
}
```

ここで見慣れないstaticというものが付いています。このstaticのことを記憶域クラス指定子といいます。

staticを付けて宣言された変数は、プログラム開始時に変数が作られ、プログラムが終了するときに消滅します。次のコードを見てください。

```
#include <stdio.h>

void func(void);

int main(void)
{
func();
        func();
        func();
        return 0;
}

void func(void)
{
static int hoge = 10;
hoge++;
        printf("hoge = %d\n",hoge);
}
```

このコードの実行結果は以下のようになります。

実行結果

hoge = 11
hoge = 12
hoge = 13

また、func 関数を通るたびに 10 が代入されているわけではありません。static int hoge = 10　の部分は初期化を表しています。static 変数では初期化はプログラム開始の前に一度行われるだけです。

1.3 プリプロセッサ

1.3.1 #define

#define とは、何らかの文字を何か別の文字に置き換える時に使うものです。わかりにくいので、例をあげて説明します。

書式

```
#define 文字列 A 文字列 B
```

#define は、A を B に置き換えるという役割を果たします。

```
#include <stdio.h>
#define NUMBER 10

int main(void)
{
printf("%d",NUMBER);
        return 0;
}
```

上記のコードは、NUMBER という文字列を 10 という定数に置きかえるという意味です。また、NUMBER と全部大文字の名前をつけましたが、#define を使って作った文字列に名前をつけるときには、全部大文字にするようにしましょう。

また、よくある間違いですが、#define で定義したあと、セミコロンをつけないようにしましょう。そのセミコロンまで置き換えられてしまいます。

1.3.2 #undef

#undef とは、一度定義したマクロを無効にしたい時に使います。

書式

#undef 文字列 A

たとえば、次のようにして使います。

```
#undef NUMBER
#define NUMBER 11
```

これは、もし#undef 以前に NUMBER というマクロがすでに存在していた場合、今までのマクロを無効にして、新しく１１という数字で置き換えろという意味です。

1.3.3 引数付マクロ

define のようなマクロ定義においても、関数と同じように引数を持つことができます。

```
#include <stdio.h>

#define SUM( x, y ) (i+j)

int main(void)
{
int sum;

        sum = SUM(3,5);
        printf("%d",sum);

        return 0;
}
```

これは、 SUM(3,5) の部分を (3+5) に置き換えるという意味です。
引数付きマクロは使い方に注意しないと、予期せぬ結果がでてくることがあります。例えば次のような場合です。

```
#define SQR(x) x*x
```

このマクロは、ある値を引数として受け取り、その二乗の値に置き換える役割を果たします。では、このマクロを使ってみましょう。

```
int x = SQR( a + b );
int y = SQR( a ) / SQR( b );
```

a と b には何か値が入っているとします。ここで、このマクロを展開してみると、次のようになります。

```
int x = a+b * a+b;
int y = a*a / b*b;
```

これではおかしな事になってしまいます。例えば、a が 3 で b が 4 だとしましょう。x の結果は本来、7 の二乗の 49 にならないといけません。しかし、3+4*3+4 は 3+12+4 となり 19 となってしまいます。また、y の値もおかしな事になってしまいます。これを回避するには、どのようにしたらよいでしょうか。答えは次のようになります。

```
#define SQR(x) ((x)*(x))
```

このようにカッコをつける事によって回避することができます。

1.3.4 #include

#include という命令は、指定したファイルを取り込むということです。たとえば、いままで printf 関数などの関数を使ってきましたが、printf 関数はどこで定義

されているでしょうか。printf 関数は stdio.h という名前のファイルに宣言されています。この stdio.h というファイルを取り込むことによって、printf や scanf などの関数を自分で作らなくても使うことができます。このように、標準で用意されている関数のことを、標準関数と呼びます。

使い方

```
#include <ファイル名>
#include "ファイル名"
```

#include には 2 通りの宣言の仕方があります。まず、はじめの< >で囲む宣言の仕方ですが、これは標準で用意されているヘッダファイルを読み込む時に使います。” ”で囲む宣言は、自分で作ったファイルを読み込む時に使います。ヘッダファイルは自分で作ることができますが、ここでは説明を省略します。

1.4　標準関数

ここで、標準で用意されている関数のほんの一部を紹介します。

1.4.1　数学関係の処理

数学で用いる関数は、math.h に用意されています。

sin 関数

```
double sin( double arg );
```

sin 関数は引数 arg の sin の値を返します。引数の単位はラジアンです。

sin 関数の使用例

```
#include <stdio.h>
#include <math.h>

int main(void)
{
double value = -1.0;
printf("%f の sin の値は%f です\n",value,sin(value));

return 0;
}
```

1.4.2 文字列関係の処理

文字列の処理に用いる関数は、string.h に用意されています。

strcpy 関数

```
char *strcpy(char* str1, char* str2);
```

strcpy 関数は文字列をコピーする際に使います。一つ目の引数に二つ目の引数の内容をコピーします。

sin 関数の使用例

```
#include <stdio.h>
#include <string.h>

int main(void)
{
char str[20];

strcpy( str , "hello" );
printf("コピーされた文字列は %s です\n",str);
return 0;
}
```

1.5　本日の講義のおさらい

- 有効範囲

  変数には寿命がある

- プリプロセッサ

  #define や #include などの事

- 標準関数

  標準で用意されている関数がたくさん存在する

参考文献

[1] Brian W. Kernighan, Dennis M. Ritchie: The C Programming Language Second Edition.

[2] ハーバートシルト著 トップスタジオ訳: 独習 C 第三版.

[3] 田中 敏幸著 C 言語によるプログラミングの基礎

[4] 柴田 望洋 明解C言語

[5] 松本 健一 上田 悦子 安室 喜弘 ２００６年度プログラミング演習（初級コース）課題

http://chihara.naist.jp/people/STAFF/yasumuro/Pub/c-ensyu2006/

[6] Ryo Kawahara C/C++プログラミング初心者講座

http://www.stat.phys.kyushu-u.ac.jp/~ryokawa/cbegin2_3/pdf/cbegin.pdf

[7] TOMOJI 初心者のためのポイント学習 C 言語

http://www9.plala.or.jp/sgwr-t/

[8] 小出 俊夫 C言語入門インターネット版

http://homepage1.nifty.com/toshio-k/prog/c/

[9] 赤坂 玲音 C 言語入門

http://wisdom.sakura.ne.jp/programming/c/index.html