ESc 101: Fundamentals of Computing

Lecture 12

Jan 27, 2010

Thoughts on the Addition Program

- There is repetition of code in the program: we read the two numbers using essentially same code.
- This also means that if we change the way of reading numbers (which we must, to handle negative numbers), then we must make the changes at two places.
- The readability of the program is also compromised.

Functions

- Functions segregate pieces of program.
- Using functions, we can separate part of program that reads a number from the part that adds two number.
- Similarly, we can separate the part that writes a number.
- It also allows use to reuse part of programs: we can use the same function to read two numbers.

Syntax of Functions

```
<type> <name>(<type-1> <var-1>, ..., <type-n> <var-n>)
<statement block>
```

Name of the function - follows same convention as name of a variable
Parameter variables supplied to the function – this allows the function to perform different tasks depending on the parameter values
Each function returns a value – this specifies the type of the value

Calling Function in a Program

Suppose a function is declared as:

```
int func_name(char number[])
<statement block>
```

It is called as:

```
char number1[SIZE];
<statements>
value = func_name(number1);
<statements>
```

The names are different! Array declaration does need not to specify size.

Structure of a C Program

```
<preamble>
<function-1>
<function-2>
:
<function-m>
```

Structure of a C Program

- `main` is also a function!
- One of the functions of a program must be `main`.
- The execution of a program always starts from the `main` function.

Execution Sequence

Suppose a program is:

```
int my_func(char number[SIZE])
{
   <statement-1>;
   <statement-2>;
}

main()
{
   char number[SIZE];

   <stmt-1>;
   my_func(number);
   <stmt-2>;
}
```

`main` is the calling function and `my_func` is the called function.

Adding Large Numbers using Functions

```
main()
{
   char number1[SIZE]; /* stores first number */
   char number2[SIZE]; /* stores second number */
   char number3[SIZE]; /* stores the result */

   read_number(number1); /* read first number */
   read_number(number2); /* read second number */
   /* Add the two numbers */
   add_numbers(number1, number2, number3);
   output_number(number3); /* output result */
}
```

Reading a Number

```
int read_number(char number[])
{
   char symbol; /* Stores current input symbol */
   char temp[SIZE]; /* temporary storage for numbers */
   int size; /* stores the number of digits in input */

   printf("Input a number of at most %d digits: ", SIZE);
   symbol = getchar(); /* read first symbol */
   for (size = 0; 1; size++) {
      if ((symbol < '0') || (symbol > '9')) /* not a digit */
         break;
      if (size == SIZE) { /* input too large */
         printf("Input too large: number should be at most %d
         return;
      }
```

Reading a Number

```
      temp[size] = symbol - '0';
      symbol = getchar(); /* read next symbol */
   }
   /* Store number in reverse order */
   int i;

   for (i = 0; i < size; i++)
      number[i] = temp[size-1-i];
   for (i = size; i < SIZE; i++)
      number[i] = 0;
}
```

The `return` Statement

- `return` exits the execution of the current function.
- If it is in `main` function, the execution stops.
- However, if it is in another function, the execution continues from the point the function was called.
- The `return` statement is also expected to provide a value for the function to return.

Modifying the read_number Program

```
int read_number(char number[])
{
   char symbol; /* Stores current input symbol */
   char temp[SIZE]; /* temporary storage for numbers */
   int size; /* stores the number of digits in input */

   printf("Input a number of at most %d digits: ", SIZE);
   symbol = getchar(); /* read first symbol */
   for (size = 0; 1; size++) {
      if ((symbol < '0') || (symbol > '9')) /* not a digit */
         break;
      if (size == SIZE) { /* input too large */
         printf("Input too large: number should be at most %d
         return 0; /* return error */
      }
```

Modifying the read_number Program

```
        temp[size] = symbol - '0';
        symbol = getchar(); /* read next symbol */
    }
    /* Store number in reverse order */
    int i;

    for (i = 0; i < size; i++)
        number[i] = temp[size-1-i];
    for (i = size; i < SIZE; i++)
        number[i] = 0;

    return 1; /* No error */
}
```

Adding Numbers

```
int add_numbers(char num1[], char num2[], char num3[])
{
   int carry; /* Stores the carry value */

   for (i = 0, carry = 0; i < SIZE; i++) {
      num3[i] = num1[i] + num2[i] + carry;
      if (num3[i] > 9) { /* new carry created */
         num3[i] = num3[i] - 10;
         carry = 1;
      }
      else /* no carry created */
         carry = 0;
   }
```

Adding Numbers

```
   if (carry == 1) { /*  sum too large */
      printf("The sum is too large!\n");
      return 0; /* return error */
   }
   return 1; /* No errors */
}
```

Output Number

```
int output_number(char number[])
{
   int i;
   /* Skip the leading zeroes */
   for (i = SIZE-1; i >= 0; i--)
   if (number[i] > 0)
      break;
   if (i == 0) /* the sum is zero! */
      printf("The sum is: 0\n");
   else {
      printf("The sum is: ");
      for (; i >= 0; i--)
         putchar(number[i]+'0');
      printf("\n");
   }
   return 1; /* No errors */
}
```

Modifying main

```
main()
{
   char number1[SIZE]; /* stores first number */
   char number2[SIZE]; /* stores second number */
   char number3[SIZE]; /* stores the result */

   /* Read first number */
   if (read_number(number1) == 0) /* error */
      return;
   /* Read second number */
   if (read_number(number2) == 0) /* error */
   /* Add the two numbers */
   if (add_numbers(number1,number2,number3) == 0) /* error */
      return;
   output_number(number3); /* output result */
}
```