Програмиране в UNIX среда

Интерпретатори, компилатори. Изпълними файлове.
Програмиране под UNIX. Система от компилатори GCC.
Граматика на език за програмиране.

Програмни езици

Програмни езици

- C Programming.

Introduction to the C Programming Language

Introduction to C Programming
Structured Program Development and Program Control
Functions
Pointers and Arrays
Characters and Strings
Data Types Structures
File Processing
Further Topics

Introduction

- Developed late 70's
- Used for development of UNIX
- Powerful
  - If used with discipline
- ANSI Standard C
  - ANSI/ISO 9899: 1990

Program Development

- Edit
  - Create program and store on system
- Preprocessor
  - Manipulate code prior to compilation
- Compiler
  - Create object code and store on system
- Linker
  - Link object code with libraries, create executable output
- Loader
- Execution
  - CPU executes each instruction

Program Structure

- Collection of
  - Source files
  - header files
  - Resource files
- Source file layout
- Function layout

Source file layout

- program.c

```
Pre-processsor directives
Global declarations

main()
{
    …….
}

function1()
{
   ……..
}

function2()
{
   ……..
}
```

Function Layout

- vartype function(vartypes)
- {
- local variables to function
- statements associated with function
- ……..
- ……..
- }

Hello World

```
/*Program1: Hello World*/

#include <stdio.h>

main()
{
        printf("Welcome to the White Rose Grid!\n");

        /*Welcome banner on several lines*/
        printf("Welcome to the \n \t White Rose Grid!\n");
}
```

Features of Hello World

- Lots of comments
  - Enclosed by /* */
- **Statements terminated with a ;**
- Preprocessor statement
  - #include <stdio.h>
    - Enables functions to call standard input ouput functions (e.g. printf, scanf)
    - Not terminated with a ;
- Printf uses escape sequence characters
  - e.g. \n newline
  - \t tab character

Variable Types

Type	Description	Size (Bytes)
int	signed integer	4
float		4
double		8
char	Signed character enclosed in single quotes	1

Variables

- Other types using unsigned and long
  - long double, long int, short int, unsigned short int
- Precision and range machine dependent
- Variables of the same type are compared using the comparison operator ==
- Variable declaration using the assignment operator =
  - float myfloat;
  - float fanother=3.1415927;

Operators

- Arithmetic operations
  - =, -, /, %, *
- Assignment operations
  - =, +=. -=, *=, %=, /=, !
- Increment and decrement (pre or post) operations
  - ++, --
- Logical operations
  - ||, &&, !
- Bitwise operations
  - |, &, ~
- Comparison
  - <, <=, >, >=, ==, !=

Input and Output Using stdio.h

- printf
  - Provides formatted input and output
  - Input for printf is a format specification followed by a list of variable names to be displayed
  - printf("variable %d is %f\n", myint, myfloat);
- scanf
  - Provided an input format and a list of variables
  - scanf("%d", &myint);
  - Note variable name has & in front
- Programarith.c is an example of the arithmetic operaions, printf and scanf.

Escape characters

Escape Sequence	Description
\n	Newline, position cursor at the start of a new line
\t	Horizontal tab, move cursor to the next tab stop
\r	Carriage return. Position cursor to the beginning of the current line; do not advance to the next line.
\a	Alert, sound system warning beep
\\	Backslash, print a backslash character in a printf statement
\”	Double quote print a double quote character in a printf statement.

Format Specifiers for printf and scanf

Data Type	Printf specifier	Scanf specifier
long double	%Lf	%Lf
double	%f	%lf
float	%f	%f
unsigned long int	%lu	%lu
long int	%ld	%ld
unsigned int	%u	%u
int	%d	%d
short	%hd	%hd
char	%c	%c

Compilation

- To compile the program myprog.c using the Portland C Compiler
  - pgcc myprog.c –o myprog
- Example compile arith.c
  - Modify program arith.c to test the effect of the decrement and increment operations
  - Modify program arith.c and test the assignment operations

Control

- Sequence Structures
- Selection Structures
  - if… else statements
  - switch structures
- Repetition Structures
  - for loops
  - while loops

Conditional Statements Using if…else

- The if statement allows decision making functionality to be added to applications.
- General form of the if statement is:
  - *if(condition)*
  - *statement;*

Using else

- An alternative form of the if statement is
  - if(condition)
  - statement;
  - else
  - statement;

If the condition is true the first statement is executed if it is false the second statement is executed.

Repetition Using while

- Execute commands until the conditions enclosed by the while statement return false.

  - while(conditions)
  - {
  - statements;
  - }

Do … while

- Good practice to always use {} in a do while loop

```
do
{
        statements…;
        Statements…;
}
while(conditions)
```

Example of while and if statement usage

```
while(files<=5)
 {
        printf("Enter file location(1=Titania, 2=Maxima): ");
        scanf("%d", &result);

        if(result==1)
                ntitania_files = ntitania_files+1;
        else if(result==2)
                nmaxima_files = nmaxima_files+1;
        else
                nother_files=nother_files+1;

        files++;
 }/*End of while file processing loop*/
```

Continue counting until Maximum number of files entered (5)

Request and get user input

Use conditions to update variables

Increment counter

Counter Controlled Repetition

- Components of a typical for loop structure
  - for(expression1; expression2; expression3)
  - statement;
- Пример
  - for(counter=1; counter<=10, counter++)
  - statement;

Multiple selection Structures Using Switch

- Used for testing variable separately and selecting a different action

```
switch(file)
{
        case 'm': case 'M':
                ++nMaxima;
        break;
        case 't': case 'T':
                ++nTitania;
        break;
        default:            /*Catch all other characters*/
                ++nOther;
        break;
} /*End of file check switch */
```

Functions

- Functions enable grouping of commonly used code into a reusable and compact unit.
- In programs containing many functions main should be implemented as a group of calls to functions undertaking the bulk of the work
- Become familiar with rich collections of functions in the ANSI C standard library
- Using functions from ANSI standard library increases portability

Standard Library Functions

Header	Description
<stdio.h>	Functions for standard input and output
<float.h>	Floating point size limits
<limits.h>	Contains integral size limits of system
<stdlib.h>	Functions for converting numbers to text and text to numbers, memory allocation, random numbers, other utility functions
<math.h>	Math library functions
<string.h>	String processing functions
<stddef.h>	Common definitions of types used by C

Functions available with the library math.h

Function	Returns
sqrt(x)	Square root
exp(x)	Exponential function
log(x)	Natural logarithm (base e)
log10(x)	Logarithm (base 10)
fabs(x)	Absolute value
pow(x,y)	X raised to the power of y
sin(x)	Trignometric sine (x in radians)
cos(x)	Trignometric cosine (x in radians)
tan(x)	Trignometric tangent (x in radians)
atan(x)	Arctangent of x (returned value is in radians)

Using Functions

- Include the header file for the required library using the preprocessor directive
  - #include <libraryname.h>
  - Note no semi colon after this
- Variables defined in functions are local variables
- Functions have a list of parameters
  - Means of communicating information between functions
- Functions can return values
- printf and scanf good examples of function calls
- Use the –lm option to compile an application using math library functions e.g.
  - pgcc myprog.c –o myprog -lm

User Defined Functions

- Format of a function definition
  - Return-value-type function-name(parameter-list)
  - {
  - declarations
  - statements
  - }
- A return value of type void indicates a function does not return a value.

Functions: return

- Return control to point from which function called
- 3 Ways to return
  - Function does not return a result (void) control is returned when function right brace } is reached.
  - Execute the statement
    - return;
  - If the statement returns a value the following statement must be executed
    - return expression;

Function Prototypes

- Tells compiler
  - type of data returned by function
  - Number and types of parameters received by a function
- Enable compiler to validate function calls
- Function prototype for a RollDice function
  - int RollDice(int iPlayer);
  - Terminated with ;
  - Placed after pre-processor declarations and before function definitions

Using Functions

- Declare function using prototype
- Define source code for function
- Call the function
- See program functions.c for an example of function declaration, definition, usage

Header Files

- Standard libraries have header files containing function prototypes for all functions in that library
- Programmer can create custom header files
  - Should end in .h e.g. myfunctionlib.h
- Programmer function prototypes declared using the pre processor directive
  - #include “myfunctionlib.h”

Pointers and Arrays

- Pointers are a powerful feature of C which have remained from times when low level assembly language programming was more popular.
- Used for managing
  - Arrays
  - Strings
  - Structures
  - Complex data types e.g. stacks, linked lists, queues, trees

Variable Declaration

- A variable is an area of memory that has been given a name.
- The variable declaration
  - float f1;
  - is command to allocate an area of memory for a float variable type with the name f1.
- The statement
  - f1=3.141
  - is a command to assign the value 3.141 to the area of memory named f1.

What is a Pointer

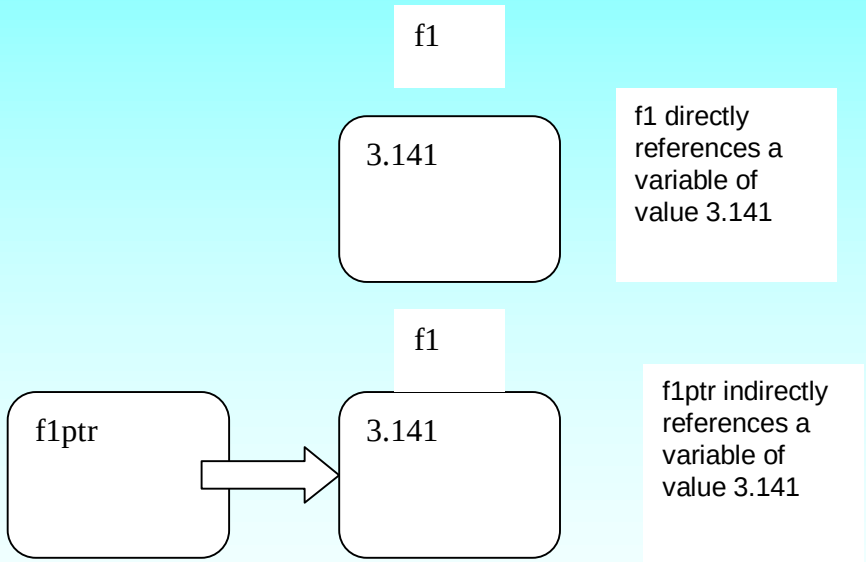
- **Pointers are variables that contain memory addresses as their values.**
- Pointer declared using the indirection or de-referencing operator *.
- Example
  - float *f1ptr;
- f1ptr is pointer variable and it is the memory location of a float variable

Pointer Example

- The redirection operator returns the address of a variable
- & applied to f1 returns the address of f1

  - float f1;
  - float *f1ptr; /* Declare a pointer variable to an integer*/
  - f1=3.141;
  - f1ptr=&f1; /*f1ptr is set to the address of f1*/

Pointer Variables



Function Calls

- Call by value
  - Copy of variable passed to function
  - If that variable is modified within the function then upon return from the function since only the copy has been modified, the actual variable is not modified
- Call by reference
  - Pass the address of a variable (i.e. a pointer) to a function
  - The variable pointed to can be modified within that function

Call By Value Example

- finval=FuncByValue(finval);
- The FuncFyValue function

```
float FuncByValue(float fval)
{
   return fval*fval;
}
```

Call By Reference Example

- FuncByReference(&finref), Use & to pass the address of a variable to the function;
- The FuncByReference function
- Value of the referenced variable passed to the function is modified after returning from the function.

```
void  FuncByReference(float *fvalptr)
{
        *fvalptr = *fvalptr * *fvalptr;
}
```

Arrays

- Initialisation
- int iarray[5]={1,2,3,4,5};
- Or… initialise elements individually
- Note first element is referenced using 0
  - iarray[0]=1;
  - ……..
  - iarray[4]=5;

Multidimensional Array

- The declaration for a multi dimensional array is made as follows:
  - *Type variable[size1][size2];*
- To access or assign an element to an element of a multidimensional array we use the statement:
  - *variable[index1][index2]=avalue;*

Matrix Initialisation example

- Alternatively the bracket initialisation method can be used, for example the integer matrix[2][4] can be initialised as follows:

```
int matrix[2][4]
{
        {1,2,3,4},
        {10,20,30,40}
};
```

Arrays are Pointers

- The array variable is a pointer whose value is the address of the first element of the array.
- For a one dimensional array access a value using the following pointer notation:
- *int ielement= *(iarray+1);*
- This assignment increments the array pointer to the second element in the array (the first element is always index 0)
- uses the * operator to dereference the pointer

Pointer Arrays

- A string is a pointer to an array of characters
- An array of strings is an array of pointers
- Multidimensional array is essentially an array of pointer arrays.

Memory Leaks

- TAKE VERY SPECIAL CARE IN USE OF POINTERS AND MANAGEMENT OF ARRAYS
- A common problem when using arrays is that the program might run off the end of the array particularly when using pointer arithmetic.
- When passing an array to a function it is good practice to pass the size of that array making the function more general.

Examples

- Compile and run the following programs
  - Program array.c initialising and using arrays with pointers
  - Program bubblesort.c is a bubble sort example, using call by reference to manipulate data passed into a function
  - Program arrayref.c uses pointer notation to manipulate arrays

- Characters and Strings
- Data Types Structures
- File Processing
- Further Topics

Characters and Strings

- A single character defined using the char variable type
- Character constant is an int value enclosed by single quotes
  - E.g. ‘a’ represents the integer value of the character a
- A string is a series of characters
  - String, string literals and string constants enclosed by double quotes

Defining characters and strings

- Declaring and assigning a single character
  - char c='a';
- Strings are arrays of characters
  - A pointer to the first character in the array
  - The last element of the string character array is the null termination charcter '\0'
  - '\0' Denotes theend of a string

Defining Strings

- char node[]="iceberg";
- char *nodeptr="iceberg";
- char nodename[180];
- For the first two definitions the null termination is added by the compiler

Character Input and Output

- include <stdio.h>
- int getchar(void)
  - Input the next character from standard input, return it as an integer.
- int putchar(int c)
  - Display character stored in c
- Also use printf and scanf with the %c format specifier

String Input and Output

- char *gets(char *s)
  - Input characters from standard inout in to the array s until newline or EOF character is reached. A NULL termination character is placed at the end of the string.
- int puts(char *s)
  - Display array of characters in s follow with a newline.
- Also use printf and scanf with the %s format specifier

Code Example Using puts and getchar

```
char c, nodename1[80], nodename2[80];
int i=0;

puts("Enter a line of text");
while((c=getchar())!='\n')
                nodename1[i++]=c;
nodename1[i]='\0';
```

Formatted String input and output

- sprintf(char *s, const char *format, …..)
  - Equivalent to printf with the exception that its output is stored in the array s specified in the sprintf function. The prototype for sscanf is ;
- sscanf(char *s, const char *format, …).
  - Equivalent to scanf reads input from the string s specified in the sscanf function.

sprintf and sscanf examples

- char node[20], s2[80];
- char s1[] ="Titania 3.78 7";
- float fload, floadout;
- int nusers, nusersout;

- /*Using sscanf to read data from a string*/
- sscanf(s1, "%s%f%d", node, &floadout, &nusersout);
- sprintf(s2, "%s %f %d", node, fload, nusers);

Functions for Character Manipulation

- library **ctype.h**
- **isdigit**, **isalpha**, **islower**, **isupper**, **toupper, tolower** and **isspace**.
- These functions can be used to perform conversions on a single character or for testing that a character is of a certain type.

String Conversion Functions

- String conversion functions from the general utilities library **stdlib**
- convert strings to float, int long int, double, long, and unsigned long data types respectively.
- **atof**, **atoi**, **atol**, **strtod**, **strtol**, **strtoul**

String Manipulation

- The string handling library **string.h**
- provides a range of string manipulation functions for copying, concatenating, comparison, tokenizing and for identifying the occurrence and positions of particular characters in a string.
- E.g. **strcpy**, **strlen**, **strcmp** and **strtok**.
- See the examples

Data Types and Structures

- Features for representing data and aggregations of different data types.
  - structures,
  - type definitions,
  - enumerations and
  - unions.

Data Structures

- Arrays and structures are similar
  - pointers to an area of memory that
  - aggregates a collection of data.
- Array
  - All of the elements are of the same type and are numbered.
- Structure
  - Each element or field has its own name and data type.

Format of a data structure

```
struct structure-name {
        field-type field-name; /*description*/
        field-type field-name; /*description*/
        .......
}  variable-name;
```

Declaring structures and Accessing Fields

- *struct structure-name variable-name;*
- A pointer to a structure
  - *struct structure-name *ptr-variable-name;*
- Accessing a field in a structure
  - *variable-name.field-name*
- For a pointer to a structure a field is accessed using the indirection operator ->
  - *ptr-variable-name->field-name*

Structure Example

```
struct node {
        char *name;
        char *processor;
        int  num_procs;
        };
```

Declaring and Initialising Structures

```
struct node n1;
struct node *n1ptr;

n1.name="Titania";
n1.processor ="Ultra Sparc III Cu";
n1.num_procs = 80;
n1ptr = &n1;
```

Accessing Structure data

- Direct access
  - printf("The node %s has %d %s processors\n",
  - n1.name, n1.num_procs, n1.processor);
- Access using a pointer
  - printf("The node %s has %d %s processors\n",
  - n1ptr->name, n1ptr->num_procs, n1ptr->processor);
- Dereferencing a pointer
  - printf("The node %s has %d %s processors\n",
  - (*n1ptr).name, (*n1ptr).num_procs, (*n1ptr).processor);

Type definitions

- *typedef float vec[3];*

- Defines an array of 3 float variables a particle position may then be defined using:
  - *vec particlepos;*
- Defined structure types
  - *typedef struct structure-name mystruct;*
  - *mystruct mystructvar;*

Enumerations

- enum enum-name {tag-1, tag-2, ….} variable-name;
- enum months {JAN=1, FEB, MAR,APR,MAY,JUN,JUL,AUG,SEP,OCT,NOV,DEC};
- Same as
- int JAN=1;
- int FEB=2;
- ..
- int DEC=12;

Using an Enumeration

- enum months month;
- for(month=JAN; month<=DEC; month++)
- statement;

File Processing

- file as a sequential stream of bytes with each file terminated by an end-of file marker
- When a file is opened a stream is associated with the file
- Streams open during program execution
  - stdin
  - stdout
  - stderr

Sequential file management

- Streams
  - channels of communication between files and programs.
- Range of functions for streaming data to files
  - fprintf
  - fscanf
  - fgetc
  - fputc

Opening a FILE

- When opening a file it is necessary to declare a variable that will be used to reference that file, the standard library provides the FILE structure.
- So a pointer to a FILE is declared using:
  - FILE *myfile;
- File opened using the function fopen
  - returns a pointer to the opened file

fopen usage

```
if((myfile=fopen("myfilename", "w"))==NULL)
                        printf("The file could not be opened!\n");
else
{
                file was opened and is read or written here
}
```

File open modes

Mode	Description
r	Open for reading
w	Open for writing
a	Append, open or create a file for writing at the end of the file
r+	Open a file for update (reading and writing)
w+	Create a file for update. If the file already exists discard the contents
a+	Append, open or create a file for update, writing is done at the end of the file

Writing data using fprintf

- *fprintf(fileptr, “format specifiers”, data list);*
  - fprintf(mfptr, "%6d %20s %6d\n", iRunid, sName, iNode);
- Closing the file
  - fclose(mfptr);

Reading data using fscanf

- *fscanf(fileptr, “format specifiers”, data list);*

```
while(!feof(mfptr))
{
        printf("%6d %20s %6d\n", sim.id, sim.name, sim.node);
        fscanf(mfptr, "%d%s%d", &sim.id, sim.name, &sim.node);
}
```

Dynamic Memory Allocation

- Allocate Memory
  - malloc
  - calloc
- Free memory
  - Free
- Size of memory used by variabletype
  - sizeof

Using malloc

- struct node *newPtr;
- newPtr = (struct node *)malloc(sizeof(struct node));

- The *(struct node *)* before the malloc statement
  - used to recast the pointer returned by malloc from (void *) to (struct node *).

Free me!

- *free(newPtr);*

Reserving memory for an array

- int n=10;
- struct node *newPtr;
- newPtr = (struct node *)calloc(n, sizeof(struct node));

- **Avoid memory leaks Free the memory!**

- free(newPtr);

Further Topics

- Bitwise operations
- The preprocessor
- Data structures
  - Linked lists
  - Stacks
  - Queues
  - Trees, oct trees, bsps etc….
- Libraries
- Make utilities

References

- http://www.cs.cf.ac.uk/Dave/C/CE.html Features examples of UNIX utility usage and network programming
- http://www.le.ac.uk/cc/tutorials/c/ccccintr.html
- http://www.shef.ac.uk/uni/academic/N-Q/phys/teaching/phy225/index.html
- http://www.bobbooth.staff.shef.ac.uk/hpcs/materials/material.html

Литература:

- http://www.wylug.org.uk/talks/2003/04/unix.pdf
- http://ce.sharif.edu/courses/ssc/unix/resources/root/Slides/unixhistory.pdf
- http://www.cs.uga.edu/~eileen/1730/Notes/intro-UNIX.ppt
- http://remus.rutgers.edu/cs416/F01
- http://www.cs.virginia.edu/~cs458/
- http://www.bobbooth.staff.shef.ac.uk/hpcs/materials/material.html
- http://www.comm.utoronto.ca/~jorg/teaching/ece461
- http://home.iitk.ac.in/~navi/sidbilinuxcourse/
- http://www.cs.washington.edu/homes/bershad/Mac/ssh/practicalmagic.pdf
- http://www.cs.cf.ac.uk/Dave/C/CE.html
- http://www.le.ac.uk/cc/tutorials/c/ccccintr.html
- http://www.shef.ac.uk/uni/academic/N-Q/phys/teaching/phy225/index.html

http://www.bobbooth.staff.shef.ac.uk/hpcs/materials/material.html