GNG1106/GNG1506 C Reference

Data Types/Declarations

Character (1 byte)	`char`
Integer	`int`
Real (single precision)	`float`
Real (double precision)	`double`
Structure (user type)	`struct`
No value	`void`
Create user type	`typedef` *type typename*

Literals

Real – exponential form	ww.yy**e**zz ww.yy**E**zz
Character literal	`'a'`
String literal	`"abc…de"`
Newline, cr, tab, backspace	`'\n' '\r' '\t' '\b'`
Special characters	`'\\' '\?' '\'' '\"'`

Operators (grouped by precedence)

Parentheses: grouping	`()`
Brackets (array subscript)	`[]`
Structure member operator	*name*`.`*member*
Structure pointer member operator	*name*`->`*member*
Minus, logical not, indirection, address	`- ! * &`
Cast	`(`*type*`)`
Determine size in bytes	`sizeof`
Multiply, divide, modulus (remainder)	`* / %`
Addition, subtraction	`+ -`
Comparisons inequality	`> >= < <=`
Comparisons equality	`== !=`
Logical AND	`&&`
Logical OR	`||`
Assignment	`=`

Unary and assignment operators evaluated right-to-left, all others left-to-right

C Preprocessor

Include standard header file	`#include <filename>`
Replacement text (defines)	`#define` *name text*

Flow Control

Statement terminator	`;`
Instruction block delimiters	`{ }`
Return from function	`return` *expr*

Flow Constructions

`if` instruction
```
if(expr) instr_block
else if(expr) instr_block
else instr_block
```

`while` instruction
```
while(expr)
    instr_block
```

`for` instruction
```
for(expr1; expr2; expr3)
    instr_block
```

`do/while` instruction
```
do instr_block
while (expr);
```

Program Structure/Functions

Function prototype: *type fncName*`(`*type1*`, `*type2*`, ...);`

Function definition:
```
type fncName(type1, type2, ...)
{
    Declaraitions
    Instructions
    return value;
}
```

Comment block	`/*` *comments* `*/`
Single line comment	`//`

Pointers, Arrays, Structure

Declare pointer variable to type	*type* `*`*name*
Define function returning address to type	*type* `*f() { }`
Null pointer	NULL
Variable referenced by *pointer*)	`*`*pointer*
Address of variable (simple and structure)	`&`*name*

Array declaration: *type name*`[`*dim*`];`

2D array declaration: *type name*`[`*rdim1*`][`*cdim2*`];`

Structure type definition
```
typedef struct
{
    Member declarations
} TYPE;
```

Member of structure variable	*name*`.`*member*
Member of referenced structure variable	*pointer*`->`*member*

Character Standard Library `<ctype.h>`

Is char alphanumeric	`isalnum(c)`
Is char alphabetic	`isapha(c)`
Is char control character	`iscntrl(c)`
Is char decimal digit	`isdigit(c)`
Is char printing char (incl space)	`isprint(c)`
Is char printing char (not incl space)	`isgraph(c)`
Is char printing char except space, letter, digit	`ispunct(c)`
Is char white space (space, formfeed, newline, cr, tab, vtab)	`isspace(c)`
Is char upper case	`isupper(c)`
Is char lower case	`islower(c)`
Convert to lower case	`tolower(c)`
Convert to upper case	`toupper(c)`

String Standard Library `<string.h>`

Length of string `s`	`strlen(s)`
Copy `srcS` to `dstS`	`strcpy(dstS, srcS)`
Concatenate `srcS` to `dstS`	`strcat(dstS, srcS)`
Compare `srcS` to `dstS`	`strccmp(dstS, srcS)`
Copy `srcS` to `dstS`	`strcpy(dstS, srcS)`
Address of first `c` in `s`	`strchr(s, c)`

Standard Utility Library `<stdlib.h>`

Pseudo-random integer, 0 to RAND_MAX	`rand()`
Set random seed to n	`srand(n)`
Convert string s to double	`atof(s)`
Convert string s to integer	`atoi(s)`

Math Standard Library `<math.h>`

Trig functions	`sin(x) cos(x) tan(x)`
Inverse trig functions	`asin(x) acos(x) atan(x)`
Hyperbolic Trig func.	`sinh(x) cosh(x) tanh(x)`
Exponential, natural log	`exp(x) log(x)`
Log base 10	`log10(x)`
Power x^y, Square root $\sqrt{x}$	`pow(x,y)  sqrt(x)`
Absolute value	`fabs(x)`
Rounding	`ceil(x)  floor(x)`

Standard Input/Output Library `<stdio.h>`

Standard I/O

Standard input stream	`stdin`
Standard output stream	`stdout`
End of file	`EOF`
Read a character	`getcahr()`
Print a character	`putchar(c)`
Print formatted data	`printf("`*format*`",arg1, …)`
Print to string	`sprintf(s,"`*format*`",arg1, …)`
Read formatted data	`scanf("`*format*`",&name1, …)`
Read from string	`sscanf(s,"`*format*`",&name1, …)`

File I/O

Declare file pointer	FILE *`fp`
Open file (returns FILE *)	`fopen("`*name*`", "`*mode*`")`
Modes:	`r` (read) `w` (write) `a` (append) `rw` (read binary) `wb` (write binary)
Read a character	`getf(fp)`
Write a character	`putc(c, fp)`
Write formatted string	`putc(c, fp)`
Print to file	`fprintf(fp,"`*format*`",arg1, …)`
Read from file	`fscanf(fp,"`*format*`",&name1, …)`
Close file	`fclose(fp)`
At end of file?	`feof(fp)`
Read line to string `s` (< `max` chars)	`fgets(s, max, fp)`
Read line to string `s` (< `max` chars)	`fgets(s, max, fp)`

Binary File I/O

Write to file	`fwrite(adr,size,num,fp)`
Read from file	`fread(adr,size,num,fp)`

Conversion codes for formatted I/O: **%+- 0w.pmc**

- `-` left justify
- `+` print with sign
- `0` pad with leading zeros
- `w` minimum filed width
- `p` precision (num of digits in fraction)
- `m` conversion character
  - (h short, l long, L long double)
- `c` conversion character
  - d,i integer
  - c single chararter
  - f double (lf required in scanf)
  - s char string
  - e, E exponential
  - g, G same as f or e,E depending on exponent

Integer Type Limits `<limits.h>`

Numbers given in parentheses are typical values for the constants.

Number bits in char	CHAR_BIT	(8)
Min value of char	CHAR_MIN	(-128)
Max value of char	CHAR_MAX	(128)
Min value of int	INT_MIN	(-2147483648)
Max value of int	INT_MAX	(+2147483647)

Real Value Type Limits `<float.h>`

Numbers given in parentheses are typical values for the constants.

For type `float`		
Decimal digits of precision	FLT_DIG	(6)
Smallest x so $1.0 + x \neq 1.0$	FLT_EPSILON	(10^{-5})
Maximum value	FLT_MAX	(10^{37})
Maximum exponent	FLT_MAX_10_EXP	
Minimum value	FLT_MIN	(10^{-37})
Minimum exponent	FLT_MIN_10_EXP	
For type `double`		
Decimal digits of precision	DBL_DIG	(10)
Smallest x so $1.0 + x \neq 1.0$	DBL_EPSILON	(10^{-9})
Maximum value for `double`	DBL_MAX	(10^{307})
Maximum exponent	DBL_MAX_10_EXP	
Minimum value for `double`	DBL_MIN	(10^{-307})
Minimum exponent	DBL_MIN_10_EXP	

GNG1106 – Creating Plots using the PLplot Library

Create a CodeBlocks project using the wizard C Console App/PLplot.
Include the header file #include <gng1106plplot.h> in your program.

HOW TO CREATE A PLOT?

Function parameters	
PLINT	Type integer
PLFLT	Type double
char *	Character string

	Objective	Function	Format
1	Initialise the plot	**plinit**	plinit();
2	Define the range and scale of the plot drawing axes, etc.	**plenv**	plenv(PLFLT xmin, PLFLT xmax, PLFLT ymin, PLFLT ymax, PLINT just, PLINT axis) ; xmin: Sets the left limit of the horizontal scale xmax: Sets the right limit of the horizontal scale. ymin: Sets the lower limit of the vertical scale ymax: Sets the upper limit of the vertical scale. just: Use zero to have each axis is scaled independently. axis: Select -2, -1, 0, 1, 2, 10, 11, 20, 21 or 30: -2: No box or annotation. -1: Draw box only. 0: Draw box, labelled with coordinate values around edge. 1: In addition to box and labels, draw the two axes X = 0 and Y = 0. 2: Same as *axis* = 1, but also draw a grid at the major tick interval. 10: Logarithmic X axis, linear Y axis. 11: Logarithmic X axis, linear Y axis and draw line Y = 0. 20: Linear X axis, logarithmic Y axis. 21: Linear X axis, logarithmic Y axis and draw line X = 0. 30: Logarithmic X and Y axes.
	Add labels to axes and title to the plot.	**pllab**	pllab(char *titleXaxis, char * titleYaxis, char *titlePlot); e.g. : pllab(‘‘string1’’, ‘‘string2’’, ‘‘string3’’); titleXaxis/string1 : label for horizontal axis titleYaxis/string 2 : label for vertical axis titlePlot/string3 : Graph title
3	Trace the curves	**plline**	plline(PLINT n, PLFLT *x, PLFLT *y); n : number of points in the arrays x : pointer to array of x coordinates y : pointer to array of y coordinates
4	Close the plot (returns when plot is closed)	**plend**	plend();

HOW TO CHANGE COLORS?

Objective	**Function**	**Format**
Select color of pen for labels or color of curves.	**plcol0**	plcol0(PLINT icol) icol : color index (des Table of colors 0)

Table of colors

WHITE	0	BROWN	8
BLACK	1	BLUE	9
YELLOW	2	BLUEVIOLET	10
GREEN	3	CYAN	11
AQUAMARINE	4	TURQUOISE	12
PINK	5	MAGENTA	13
WHEAT	6	SALMON	14
GREY	7	RED	15

Symbolic constants provided in the Table of colors can be used as arguments in the functions for colors. The function `plcol0` must be call before the functions `pllab` and `plline` to modify the pen color for printing labels and tracing curves.

HOW TO CHANGE PEN THICKNESS?

Objective	**Function**	**Format**
Sets the thickness of the pen.	**plwidth**	plwidth (PLINT thickness) thickness: thickness (minimum value is 0)

HOW TO CHANGE LINE STYLE?

Objective	**Function**	**Format**
Sets the line style	**pllsty**	pllsty (PLINT style) style : See the Table for style values

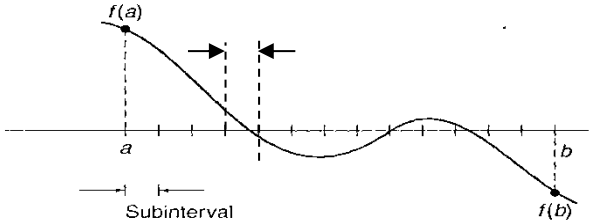
Table for style values

SOLID	1	**Continuous line**
SHRTDASH_SHRTGAP	2	Short dashes and gaps
LNGDASH_LNGGAP	3	Long dashes and gaps
LNGDASH_SHRTGAP	4	Long dashes with short gaps
LNGDASH_SHRTDASH	5	Long dash – short dash

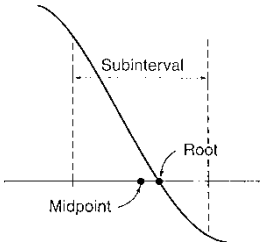
Numerical Methods

Root Finding – Incremental Search

- We delimit a search interval a ≤ x ≤ b where we suspect the existence of a real root (use graphing to find such intervals).
- The interval a≤x≤b is divided into a number of equal subintervals.



- Each subinterval is verified for a change in sign in f(x).
  - For example, over subinterval k, we have $f(a_k)$ which is positive and $f(b_k)$ which is negative; a root is thus present for x in the range $a_k \le x \le b_k$.
- We can now simply approximate this root by the value of x located at the midpoint of the subinterval.
  - This approximation becomes more accurate as the size of the subinterval decreases.



Root Finding – Bisection Method

- Step 1: Choose the search interval: x_L lower edge (left) and x_U upper edge (right) such that the function changes sign over the interval. This can be checked by ensuring that $f(x_L)f(x_U) < 0$.
- Step 2: An estimate of the root x_R is determined by
  $x_R = (x_L + x_U)/2$.
- Step 3: Make the following evaluations to determine in which subinterval the root lies:
  - (*a*) If $f(x_L)f(x_R) < 0$, the root lies in the lower subinterval. Therefore, set $x_U = x_R$ and return to step 2.
  - (*b*) If $f(x_L)f(x_R) > 0$, the root lies in the upper subinterval. Therefore, set $x_L = x_R$ and return to step 2.
  - (*c*) If $|f(x_L)f(x_R)|$ < some small tolerance value (that is, can be considered equal to 0), the root equals x_R; terminate the computation.

Euler's Method

For the equation $\frac{dx}{dt} = f(x)$, the difference equation is:

$$x_{i+1} = x_i + f\left(x_i\right)\Delta t$$

where

time is defined as: $t_{i+1} = t_i + \Delta t$ and Δt is the time step.

And x_0 and t_0 are given, that is, are the initial conditions.

Trapezoidal Rule

Numerical equation for the definite integral is given by: $\int_a^b f(x)dx = \frac{h}{2}\left[f(x_0) + \left[2\sum_{i=1}^{n-1} f(x_i) \right] + f(x_n) \right]$

Where

n is the number of steps that divides the *x* interval $a \le x \le b$

h is the step width = $(b-a)/n = x_i - x_{i-1}$

$x_0 = a$ and x_n = b

Empty Page

Empty Page

Empty Page