"Advanced C Programming"

GNU toolchain

The **GNU toolchain** is a broad collection of programming tools produced by the GNU Project.

Major components of GNU toolchain are:

- GNU make: an automation tool for compilation and build
- GNU Compiler Collection (GCC): a suite of compilers for several programming languages
- GNU C Library (glibc): core C library including headers, libraries, and dynamic loader
- GNU Binutils: a suite of tools including linker, assembler and other tools
- GNU m4: an m4 macro processor
- GNU Debugger (GDB): a code debugging tool

Source:https://en.wikipedia.org/wiki/GNU_toolchain

GNU make

GNU make

What is a Makefile?

A makefile is a specially formatted text file that a unix/linux program called 'make' can interpret.

Basically, the makefile contains a list of requirements for a program to be 'up to date.' The make program looks at these requirements, checks the timestamps on all the source-files listed in the makefile, and re-compiles any files which have an out-of-date timestamp.

product.c

```
#include <stdio.h>
/* Function Prototype */
int product(int n1,int n2);
int main()
{
        int num1,num2,p;

        printf("Enter a num1:\n");
        scanf("%d",&num1);

        printf("Enter a num2:\n");
        scanf("%d",&num2);

        p=product(num1,num2);
        printf("prod=%d \n",p);

        return 0;
}
```

```
/* Function Definition */
int product(int n1, int n2)
{
                return n1*n2;
}
```

Executable generation:

In Single step:

gcc product.c –o product

In two step:

Step1: object file(product.o) generation

gcc -c product.c

Step2: object file to executable

gcc product.o -o product

prod_main.c

```
#include <stdio.h>
#include “prod_fun.c” // user def file
/* Function Prototype */
int product(int n1,int n2);
int main()
{
        int num1,num2,p;

        printf("Enter a num1:\n");
        scanf("%d",&num1);

        printf("Enter a num2:\n");
        scanf("%d",&num2);

        p=product(num1,num2);
        printf("prod=%d \n",p);
        return 0;
}
```

prod_fun.c

```
/* Function Definition */
int product(int n1, int n2)
{
                return n1*n2;
}
```

Executable generation:

gcc prod_main.c -o prod

prod_main.c

```
#include <stdio.h>
#include “prod_fun.c” // user def file
/* Function Prototype */
int product(int n1,int n2);
int main()
{
        int num1,num2,p;

        printf("Enter a num1:\n");
        scanf("%d",&num1);

        printf("Enter a num2:\n");
        scanf("%d",&num2);

        p=product(num1,num2);
        printf("prod=%d \n",p);
        return 0;
```

prod_fun.c

```
/* Function Definition */
int product(int n1, int n2)
{
        return n1*n2;
}
```

Executable generation:

In two step:

Step1: object file(prod_main.o prod_fun.o) generation

gcc -c prod_main.c

gcc -c prod_fun.c

Step2: object file to executable

gcc prod_main.o prod_fun.o –o prod

Executable generation using Makefile

```
Target: dependencies
      action
-----------------------------------------------------------------------
prod: prod_main.o prod_fun.o
        gcc prod_main.o prod_fun.o -o prod

prod_main.o: prod_main.c
        gcc -c prod_main.c

prod_fun.o: prod_fun.c
        gcc -c prod_fun.c
```

area.c

```
#include <stdio.h>
/* Function Prototype */
int product(int n1,int n2);
int main()
{
        int l,w,area;

        printf("Enter length:\n");
        scanf("%d",&l);

        printf("Enter width:\n");
        scanf("%d",&w);

        area=product(l,w);
        printf("area=%d \n",area);
        return 0;
}
```

prod_fun.c

```
/* Function Definition */
int product(int n1, int n2)
{
                return n1*n2;
}
```

Different Executable generation using Makefile

```
prod: prod_main.o prod_fun.o
	gcc prod_main.o prod_fun.o -o prod

prod_main.o: prod_main.c
	gcc -c prod_main.c

prod_fun.o: prod_fun.c
	gcc -c prod_fun.c


area: area.o prod_fun.o
	gcc area.o prod_fun.o -o area

area.o: area.c
	gcc -c area.c
```

Make for prod executable

make prod

Make for area executable

Make area

multiple Executable generation using Makefile

```
all: prod area
prod: prod_main.o prod_fun.o
	gcc prod_main.o prod_fun.o -o prod
prod_main.o: prod_main.c
	gcc -c prod_main.c
prod_fun.o: prod_fun.c
	gcc -c prod_fun.c

area: area.o prod_fun.o
	gcc area.o prod_fun.o -o area
area.o: area.c
	gcc -c area.c
clean:
	rm *.o prod area
```

Make for all executables

make all

Removing exe and executable

Make clean