**Build Process**  
Build Process là một quá trình xử lý với Input là các file source code (.c, .cpp, .h, .s, ...) và đầu ra là các file nhị phân để xử dụng (.exe, .elf, .hex, .bin, ...).

Build Process trong một chương trình C sẽ trải qua các bước sau:

A diagram of a program

Description automatically generated

**Step 1. Preprocessor**

    Step này sẽ gộp các file source với file thư viện tương ứng như là thay #include <stdio.h> bằng toàn bộ nội dung file stdio.h, cùng với đó là xử lý các chỉ thị tiền xử lý như là xóa bỏ tất cả chú thích, comments của chương trình và xử lý **#define**đã được thay thế.

  Câu lệnh GCC cho Step này:

**gcc -E main.c -o main.i**

**Step 2. Compilation**

    Quá trình này sẽ biến file main.i trên thành file mã máy - Assembly.

Câu lệnh:

**gcc -S main.i -o main.s**

Output là file **main.s** , có kiến thức về ngôn ngữ Assembly thì có thể đọc hiểu được

**Step 3. Assembler**

    Step này sử dụng trình Assembler thay thì Compiler, nó gộp các file .s vừa tạo với các file .s có sẵn (nếu có) tạo thành các file object.

 Câu lệnh:

**as main.s -o main.o**

Output là file main.o, với nội dung và định dạng mà chúng ta không đọc được

**Step 4. Linking**

    Sau khi có các file object, chúng sẽ được Link cùng với các static library (nếu có).

    Câu lệnh:

**gcc -v -o main main.o**

    Output của quá trình này là file **main.exe** dùng để run.

**Macro**

* Macros là từ hay chỉ thị dùng để chỉ những thông tin được xử lý ở quá trình tiền xử lý.
* Macros chia làm 3 nhóm chính:
  + Chỉ thị bao hàm tệp (**#include**)
  + Chỉ thị định nghĩa, hủy định nghĩa (**#define, #undef**)
  + Chỉ thị biên dịch có điều kiện (**#if, #elif, #else, #ifdef, #ifndef, #endif**)

**Các nhóm Macro chính:**

**a. Chỉ thị bao hàm file (#include):**

* Sử dụng để thêm nội dung của file khác vào chương trình.
* Phân biệt hai kiểu dấu ngoặc:
  + #include <stdio.h> (tìm kiếm trong thư viện hệ thống)
  + #include "abc.txt" (tìm kiếm trong thư mục hiện tại trước, rồi mới tìm trong thư viện hệ thống)

Mục đích:

* Tái sử dụng mã nguồn.
* Phân chia chương trình thành các phần nhỏ dễ quản lý.

**b. Chỉ thị định nghĩa (#define):**

Sử dụng để định nghĩa các hằng số hoặc macro thay thế chuỗi mã nguồn trước khi biên dịch.

**Ví dụ định nghĩa hằng số:**

#define PI 3.14

double area = PI \* radius \* radius;

**Ví dụ định nghĩa hàm đơn giản bằng macro:**

#define SQUARE(x) ((x)\*(x))

int result = SQUARE(5); // Kết quả là 25

**Ví dụ định nghĩa macro in nhiều lệnh:**

#define DISPLAY\_SUM(a,b) \

printf("This is macro to sum 2 number\n"); \

printf("Result is: %d", a + b);

**Ví dụ macro so sánh:**

#define MAX(x,y) ((x) > (y) ? (x) : (y))

**c. Chỉ thị hủy định nghĩa (#undef):**

Hủy bỏ định nghĩa của một macro đã được định nghĩa trước đó.

**Ví dụ:**

#define SENSOR\_DATA 42

#undef SENSOR\_DATA

#define SENSOR\_DATA 50

**d. Chỉ thị biên dịch có điều kiện (#if, #elif, #else):**

* Xử lý điều kiện tiền biên dịch, chỉ biên dịch các đoạn mã thỏa mãn điều kiện.
* Hỗ trợ viết code tương thích nhiều môi trường, nền tảng khác nhau.

**Ví dụ:**

#define MCU STM32

#if MCU == STM32

// code cho STM32

#elif MCU == ESP32

// code cho ESP32

#else

// code khác

#endif

**e. Chỉ thị biên dịch có điều kiện kiểm tra Macro (#ifdef, #ifndef):**

* #ifdef: kiểm tra macro đã được định nghĩa chưa.
* #ifndef: kiểm tra macro chưa được định nghĩa.

**Ví dụ tránh định nghĩa nhiều lần:**

#ifndef \_\_ABC\_H

#define \_\_ABC\_H

int a = 10;

#endif

**Một số toán tử đặc biệt trong Macro:**

**Toán tử # (Stringizing Operator)**

🔹 **Khái niệm:**

* Toán tử # dùng để biến một tham số của macro thành một chuỗi ký tự (string).
* Quá trình này gọi là stringizing.

🔹 **Cú pháp:**

#define STRINGIZE(x) #x

**🔹 Ví dụ minh họa:**

#include <stdio.h>

#define STRINGIZE(x) #x

int main() {

printf("Result: %s\n", STRINGIZE(HelloWorld));

printf("Result: %s\n", STRINGIZE(12345));

printf("Result: %s\n", STRINGIZE(3.14));

return 0;

}

**Kết quả đầu ra:**

Result: HelloWorld

Result: 12345

Result: 3.14

**Toán tử ## (Token-Pasting Operator)**

**🔹 Khái niệm:**

* Toán tử ## dùng để **ghép nối** (concatenate) hai token lại thành một token mới trong quá trình tiền xử lý.
* Token ở đây có thể là tên biến, tên hàm, hoặc từ khóa.

**🔹 Cú pháp:**

#define CONCAT(a, b) a##b

**🔹 Ví dụ minh họa:**

#include <stdio.h>

#define DECLARE\_VARIABLE(prefix, number) int prefix##number

int main() {

DECLARE\_VARIABLE(var, 1); // int var1;

DECLARE\_VARIABLE(myVar, 2); // int myVar2;

var1 = 10;

myVar2 = 20;

printf("var1: %d\n", var1);

printf("myVar2: %d\n", myVar2);

return 0;

}

**Kết quả đầu ra:**

var1: 10

myVar2: 20

Macro DECLARE\_VARIABLE(var, 1) được tiền xử lý thành khai báo biến int var1;.

Toán tử ## ghép prefix và number thành một tên biến hoàn chỉnh.

**Variadic Macro (...)**

**🔹 Khái niệm:**

* Variadic Macro là loại macro có thể nhận số lượng tham số thay đổi.
* Dùng dấu ba chấm ... để biểu thị rằng macro có thể nhận từ 0 đến nhiều tham số.

**🔹 Cú pháp:**

#define MACRO\_NAME(...) \_\_VA\_ARGS\_\_

* Trong đó, \_\_VA\_ARGS\_\_ là tên đặc biệt đại diện cho toàn bộ tham số truyền vào macro.

🔹 **Ví dụ minh họa rõ ràng:**

#include <stdio.h>

#define PRINT\_MENU(...) \

do { \

const char \*items[] = {\_\_VA\_ARGS\_\_}; \

int n = sizeof(items)/sizeof(items[0]); \

for (int i = 0; i < n; i++) \

printf("%d. %s\n", i + 1, items[i]); \

} while(0)

int main() {

PRINT\_MENU("Option 1", "Option 2", "Option 3", "Exit");

return 0;

}

**Kết quả đầu ra:**

1. Option 1

2. Option 2

3. Option 3

4. Exit