Лекция 11. Дополнительные конструкции языка C

Директивы препроцессора

Препроцессор — предварительная обработка текста программы, порождает новый текст, который уже и идет на компиляцию.

```
#include "filename"
#include <filename>
```

Вставляет содержимое файла filename в данное место файла. Первый вариант ищет файл в текущем каталоге, второй вариант ищет файл в каталоге include данного компилятора (задается переменной окружения INCLUDE).

```
#define  имя   подстановка
#define  имя(параметры)   текст с параметрами
#define  имя
#undef   имя
```

Далее по тексту это имя будет “‘определено” и также выполнены соответствующие подстановки.

```
#define N 1000
int a[N];       эквивалентно     int a[1000];
общее правило --- все существенные константы должны определяться через define

#define  M_PI  3.14159265358979
#define  M_E   2.718281829459045
#define  NUM_VALUES   12345
#define  SUCCESS   0
#define  ERROR    -1

#define  RES_VALUES   (NUM_VALUES - 1)                (12345 - 1)

int f()
{
    double x[NUM_VALUES];
    ...
    for (i=0; i<NUM_VALUES; i++) x[i] = 12345;
    ...
    if (...) return  ERROR;
    ...
    return SUCCESS;
}
```

```
if (f() == SUCCESS) ...

#define begin {
#define end }

if (...) begin ... end
```

Второй вариант — макроопределение (псевдофункция)

```
#define abs(x)   (((x)>0) ? (x) : -(x))
#define min(a,b) ((a) < (b)) ? (a) : (b))

y = abs(z);                              y = (((z)>0) ? (z) : -(z));
y = abs(r-s);     !!!  скобки !!!        y = (((r-s)>0) ? (r-s) : -(r-s));
w = min(s, t);                           w = ((s) < (t)) ? (s) : (t))
w = min(sin(x), cos(x));    !!! лишние вычисления !!!

#define  SetC (a,n,c)  for(int i=0; i<n; i++) { a[i] = c; }

int x[10];
double y[20];
char z[30];

SetC(x,10,0);
SetC(y,20,100);
SetC(z,30,111);
```

Третий вариант определяет имя (и снимает определение)

```
#define DEBUG
#define TEST_MODE

#undef  DEBUG
#undef  min

#ifdef имя
    .... оставляет текст, если определено
#else
    .... оставляет текст, если не определено
#endif

#ifndef имя
```

```
      ----  оставляет текст, если не определено
   #else
      ----  оставляет текст, если определено
   #endif

   #ifdef DEBUG
      printf(....);
   #endif

   #ifdef TEST_MODE
      if (k < 0) { printf("error K<0 !!!\n"); return; }
   #endif
   res = sqrt(k);

   #ifndef M_PI
   #define M_PI  3.14159265358979
   #endif
```

Прием, чтобы файл вставлялся только 1 раз

```
   файл abc.h
   #ifndef  ABC_H
   #define  ABC_H
   ..... содержательные строки файла
   #endif
---------------------------------
   файл с несколькими включениями
   #include "abc.h"
   #include "abc.h"

   получается так:
   #ifndef  ABC_H
   #define  ABC_H
   .....                      ......   на выходе только один раз
   #endif
   #ifndef  ABC_H
   #define  ABC_H
   .....   это уже не переносится
   #endif
```

Т.е. вставится только 1 раз

```
   #pragma once
   в файле abc.h даст тот же эффект.
```

`#pragma ...` — некоторое специальное указание компилятору (их много разных, описывается в документации).

Параметры функции main

`main()` — без параметров

`main(int argc, char **argv)` — с параметрами командной строки

`main(int argc, char **argv, char **envr)` — с параметрами командной строки и переменными окружения

Пример: main.cpp

Структурные типы и typedef

Структура — объект, состоящий из нескольких полей.

Объединение — объект, способный вместить любое поле из заданного списка

```
struct Vector {
    double x,y,z;
};
struct Student {
    char name[128];
    double rating;
    int  year, group, age;
};
```

Доступ к полю структуры — операция . от объекта и -> от указателя на объект.

Различие в объявлениях

```
C:                                      C++:
struct Vector a,b,c;                    Vector a,b,c;
struct Student  Mary, John;             Student  Mary, John;

a.x = a.y = a.z = 0;
b.x = a.x + a.y;
Mary.rating = 5.0;
John.rating = 3.5;

struct Student * p;
p = & Mary;
p->year = 1;
p->group = 123;
...
printf("%s\n", John.name);
printf("%s\n", p->name);
```

Можно присваивать, создавать массивы, передавать в функции, возвращать.

```
struct Vector  v[10];
v[0] = a;
v[1] = b;

double Length(struct Vector u)
{
    return sqrt(u.x*u.x + u.y*u.y + u.z*u.z);
}
struct Vector Inverse(struct Vector u)
{
    u.x = -u.x;
    u.y = -u.y;
    u.z = -u.z;
    return u;
}
```

Важно !!!
`sizeof (struct ...)` не всегда равняется сумме размеров ее компонент
Объединения — можно рассматривать только одну сущность в каждый момент.

```
union  SomeNumber {
   double d;
   float f;
   int i;
   short s;
};

union  SomeNumber a;

a.d = 3.14;
a.i, a.f, a.s --- непойми что!
a.i = 123;
теперь a.f a.d a.s --- "непонятно какие"
просто размещение в одной памяти и интерпретация по описанному полю

union  DoubleInBytes {
   double x;
   char b[8];
}

union  DoubleInBytes a;
a.x = 123.456;
```

```
теперь можно распечатать байты в шестнадцатеричном формате
и увидеть представление double
printf("%02X", a.b[0]);
printf("%02X", a.b[1]);
printf("%02X", a.b[2]);
...
```

Инструкция typedef позволяет дать более простое обозначение для другого типа, например, сделать объявления структур в C похожим на C++

```
typedef struct V {double x, y;}  Vector2;
Vector2 a,b,c;      struct V a,b,c;

typedef int (*Compare)(const void *, const void *);

void QuickSort(void *a, int n, int elem_len, Compare f);
```