Програмчлалын үндэс CS200

Лекц № 12

Си-гийн препроцессор

А. Хүдэр

Агуулга

- Си-гийн препроцессор
  - Том хэмжээтэй програм хөгжүүлэхэд #include -ийг ашиглах
  - #define-г ашиглан аргументтай макро зохиох
  - Програмын зарим хэсгийг (жишээ нь зүгшрүүлэлтэнд туслах хэсгийг) хэзээ хөрвүүлэхийг заах буюу нөхцөлт хөрвүүлэлт хийх
  - Нөхцөлт хөрвүүлэлтийн үед алдааны мэдээлэл гаргах
  - Илэрхийллийн утга зөв байгаа эсэхийг баталгаа (assertion) ашиглан шалгах
- Си-гийн бусад сэдвүүд

Оршил

- Препроцессор
  - Програмыг хөрвүүлэхээс өмнө ажиллана
  - Бусад файлыг оруулах
  - Тэмдэгтэн тогтмол ба макрог тодорхойлох
  - Програмын нөхцөлт хөрвүүлэлт
  - Препроцессорын зааврыг нөхцөлтэйгээр биелүүлэх
- Препроцессорын зааврын хэлбэр
  - Мөр нь #-аар эхэлнэ
  - Препроцессорын заавар бүхий мөрийн эхэнд хоосон тэмдэгтүүд болон заавар л байж болно

#include пропроцессорын заавар

- #include
  - Зааж өгсөн файлын хуулбарыг зааврын байгаа мөрнөөс эхлэн оруулна
  - #include <файлын нэр>
    - Стандарт сангаас файлыг хайна
    - Стандарт сангийн файлуудад хэрэглэнэ
  - #include “файлын нэр”
    - Эхлээд идэвхтэй хавтаснаас хайгаад дараа нь стандарт сангаас хайна
    - Хэрэглэгчийн үүсгэсэн файлд хэрэглэнэ
  - Хэрэглээ:
    - Олон файлыг зэрэг хөрвүүлэхэд хэрэглэгдэнэ
    - Толгой файл – ерөнхий тодорхойлолтууд болон зарлалтуудыг агуулна (классууд, бүтцүүд, функцийн загварууд)

#define препроцессорын заавар: Тэмдэгтэн тогтмол

- #define
  - Тэмдэгтэн тогтмол болон макро үүсгэхэд хэрэглэгдэнэ
  - Тэмдэгтэн тогтмолууд
    - Програм хөрвүүлэгдэхийн өмнө тэмдэгтэн тогтмолуудыг хайж олоод солих текстээр нь орлуулдаг
  - Хэлбэр

    #define идентификатор солих текст
  - Жишээ

    #define PI 3.14159
  - Идентификатор баруун талд байгаа зүйлсээрээ солигдоно

    #define PI = 3.14159
    - PI-г "=3.14159" - өөр солино
  - Тэмдэгтэн тогтмолуудыг зарласан бол дахин зарлаж болохгүй

#define препроцессорын заавар: Макро

- Макро
  - #define-д тодорхойлогдсон үйлдэл
  - Аргументгүй макрог тэмдэгтэн тогтмол гэж үзнэ
  - Аргументтэй макро нь түүнийг хэрэглэх үед орлуулах текстээр солигдох аргументуудыг агуулна
  - Текст солих үйлдлийг гүйцэтгэнэ – өгөгдлийн төрлийг шалгахгүй
  - Дараах макро

```
#define CIRCLE_AREA( x ) ( PI * ( x ) * ( x ) )
```

  - доорх мөрийг

```
area = CIRCLE_AREA( 4 );
```

  - ийм мөр болгон хувиргана

```
area = ( 3.14159 * ( 4 ) * ( 4 ) );
```

#define препроцессорын заавар: Макро

- Хаалт хэрэглэх
  - Хаалт хэрэглээгүй үед:

```
#define CIRCLE_AREA( x ) PI * x * x
area = CIRCLE_AREA(c+2);
```

мөрийг

```
area = 3.14159 * c + 2 * c + 2;
```

болгоно

- Олон аргумент

```
#define RECTANGLE_AREA( x, y ) ( ( x ) * ( y ) )
```

макро нь

```
rectArea = RECTANGLE_AREA( a + 4, b + 7 );
```

мөрийг

```
rectArea = ( ( a + 4 ) * ( b + 7 ) );
```

болгоно

#define препроцессорын заавар: Макро

- #undef
  - Тэмдэгтэн тогтмол болон макрог хүчингүй болгоно
  - Тэмдэгтэн тогтмол болон макрог хүчингүй болгосны дараа дахин тодорхойлж болно

Нөхцөлт хөрвүүлэлт

- Нөхцөлт хөрвүүлэлт
  - Програмын код болон препроцессорын зааврууудын биелэлтийг удирдана
  - if-тэй төстэй бүтэцтэй

```
#if !defined( NULL )
    #define NULL 0
#endif
```

    - Дээрх макро нь NULL гэдэг тогтмол тодорхойлогдсон эсэхийг шалгаж байна
      - Хэрэв NULL тодорхойлогдсон бол түүнийг 1-тэй тэнцүү эсэхийг шалгана
      - Хэрэв NULL тодорхойлогдоогүй бол түүнийг 0-тэй тэнцүү гэж тодорхойлно
  - #if бүр #endif-ээр төгсөнө
  - #ifdef нь #if defined(нэр) - ийн товч хэлбэр
  - #ifndef нь #if !defined(нэр) - ийн товч хэлбэр

Нөхцөлт хөрвүүлэлт

- Бусад үйлдлүүд
  - `#elif` нь `if` үйлдэл дэх `else if`-тэй төстэй
  - `#else` нь `if` үйлдэл дэх `else` - тэй төстэй
- Кодын хэсгийг тайлбар болгоно
  - Хэрэв код тайлбартай бол /* болон */ - ыг энэ зорилгоор хэрэглэж болохгүй
  - Оронд нь

    #if 0

    тайлбар болгох код

    #endif - ийг хэрэглэнэ
  - Кодыг идэвхжүүлэхийн тулд 0-ийг 1-ээр солино

Нөхцөлт хөрвүүлэлт

- Зүгшрүүлэлт

```
#define DEBUG 1
#ifdef DEBUG
   printf("Variable x = %d\n", x);
#endif
```

  - DEBUG-ийг 1 гэж тодорхойлсноор кодыг идэвхжүүлнэ
  - Кодыг зассаны дараа #define үйлдлийг хасна
  - Зүгшрүүлэлтийн үйлдлүүдийг одоо алгасна

#error препроцессорын заавар

- Хэлбэр: #error токенууд
  - Токен гэдэг нь хоосон зайгаар тусгаарлагдсан тэмдэгтүүд юм
    - “I like C” нь 3-н токентой байна
- #error заавар ямар нэг системд биелэгдэх үед энэ нь:
  - заасан токенуудыг агуулсан алдааны мэдээллийг хэвлэнэ

```
#if BUFFER_SIZE < 256
#error "BUFFER_SIZE is too small.“
#endif
```

  - препроцессорыг зогсоож, програмыг хөрвүүлэхгүй

ба ## операторууд

- #
  - Орлуулах текстийн токеныг давхар хашилтан дахь тэмдэгт мөр болгон хувиргах
  - Дараах үйлдэл

    #define HELLO(x) printf(“Hello, ” #x “\n”);

    доорх мөрийг

    HELLO(John)

    ийм мөр болгон хувиргана

    printf(“Hello, ” “John” “\n”);
  - printf-ийг хэрэглэхэд хоосон зайгаар тусгаарлагдсан тэмдэгт мөрүүд залгагдана

ба ## операторууд

- ##
  - Хоёр токеныг залгана
  - Доорх илэрхийлэл

    #define TOKENCONCAT(x, y) x ## y

    нь

    TOKENCONCAT(O, K)

    мөрийг

    OK

    болгон хувиргана

Урьдчилан тодорхойлогдсон тэмдэгтэн тогтмолууд

- Дөрвөн урьдчилан тодорхойлогдсон тэмдэгтэн тогтмол
  - #define болон #undef – д хэрэглэгдэхгүй

Зарим тэмдэгтэн тогтмолууд

Symbolic constant	Explanation
__LINE__	The line number of the current source code line (an integer constant).
__FILE__	The presumed name of the source file (a string).
__DATE__	The date the source file was compiled (a string of the form "Mmm dd yyyy" such as "Jan 19 2002").
__TIME__	The time the source file was compiled (a string literal of the form "hh:mm:ss").
__STDC__	The value 1 if the compiler supports Standard C.

Тайлбар: Эдгээр идентификаторууд болон зарласан идентификаторууд #define болон #undef зааврууддад хэрэглэгдэхгүй

Баталгаа

- assert макро
  - <assert.h> толгой файлд тодорхойлогдсон
  - Илэрхийллийн утгыг шалгана
  - Хэрэв 0 (false) бол алдааны мэдээллийг хэвлээд abort-ыг дуудна
  - Жишээ:

    ```
    assert( x <= 10 );
    ```

    - Хэрэв x нь 10-аас их бол өмнөх үйлдэл дайралдсаны дараа мөрийн дугаар, файлын нэрийг агуулсан алдааны мэдээлэл хэвлэгдэж, програм тасарна.

Програмчлалын үндэс CS200

Лекц № 13

Өгөгдөл боловсруулалт

А. Хүдэр

Си-гийн бусад сэдвүүд

- Гараас орох оролтыг файлаас оруулах
- Дэлгэц рүү гарах гаралтыг файл руу бичих
- Хувьсах урттай аргументын жагсаалт бүхий функц бичих
- Командын мөрийн аргументуудыг боловсруулах
- Тоон тогтмолуудад тусгай төрөл олгох
- Түр зуурын файл хэрэглэх
- Програмд гадны асинхрон үйл явдлуудыг боловсруулах
- Хүснэгтэд зориулан санах ойг динамикаар хуваарилах
- Өмнө нь динамикаар хуваарилсан санах ойн хэмжээг өөрчлөх

Unix/Linux дээр Оролт/Гаралтыг дахин чиглүүлэх

- Стандарт Оролт/Гаралт – гар ба дэлгэц
  - Оролт ба гаралтыг чиглүүлэх
- Чиглүүлэх тэмдэгт (<)
  - Си-гийн боломж биш үйлдлийн системийн боломж
  - $ болон % нь командын мөрийг заана
  - Жишээ

    $ sum < input
  - Утгыг гараас бус файлаас унших
- Хоолой команд (|)
  - Нэг програмын гаралт нөгөө програмын оролт болно

    $ random | sum
  - random-ын гаралт sum-ын оролттой залгагдана

Unix/Linux дээр Оролт/Гаралтыг дахин чиглүүлэх

- Гаралтыг чиглүүлэх (>)
  - Програмын гаралтыг хааш чиглүүлэхийг заана
  - Жишээ:

    $ random > out
    - Гаралтыг out гэсэн файл руу чиглүүлнэ (өмнөх агуулгыг устгана)
- Гаралт дээр нэмэх (>>)
  - Гаралтыг файлын төгсгөлд нэмнэ (өмнөх агуулгыг хадгална)
  - Жишээ:

    $ random >> out
    - Гаралтыг out файлын төгсгөлд бичнэ

Хувьсах урттай аргументын жагсаалт

- Тодорхойгүй тооны аргумент бүхий функц
  - <stdarg.h>-ийг ачаална
  - Параметрын жагсаалтын төгсгөлд гурван цэг (…) бичнэ
  - Дор хаяж нэг параметр тодорхойлсон байх ёстой
  - Жишээ:
    - double myfunction (int i, ...);
  - Хувьсах урттай аргументын жагсаалт бүхий функцийн загвар болон тодорхойлолтонд гурван цэг байна
  - printf нь олон аргумент авч чадах функцийн жишээ юм
  - printf -ийн загвар нь дараах байдалтай байдаг

    int printf(const char* format, ...);

stdarg.h дэх хувьсах урттай аргументын жагсаалт бүхий төрөл болон макро

Identifier	Explanation
`va_list`	A type suitable for holding information needed by macros `va_start`, `va_arg` and `va_end`. To access the arguments in a variable-length argument list, an object of type `va_list` must be defined.
`va_start`	A macro that is invoked before the arguments of a variable-length argument list can be accessed. The macro initializes the object declared with `va_list` for use by the `va_arg` and `va_end` macros.
`va_arg`	A macro that expands to an expression of the value and type of the next argument in the variable-length argument list. Each invocation of `va_arg` modifies the object declared with `va_list` so that the object points to the next argument in the list.
`va_end`	A macro that facilitates a normal return from a function whose variable-length argument list was referred to by the `va_start` macro.

```c
/* Example 100:
   Using variable-length argument lists */
#include <stdio.h>
#include <stdarg.h>

double average( int i, ... ); /* prototype */

int main( void )
{
   double w = 37.5;
   double x = 22.5;
   double y = 1.7;
   double z = 10.2;

   printf( "%s%.1f\n%s%.1f\n%s%.1f\n%s%.1f\n\n",
      "w = ", w, "x = ", x, "y = ", y, "z = ", z );
   printf( "%s%.3f\n%s%.3f\n%s%.3f\n",
      "The average of w and x is ", average( 2, w, x ),
      "The average of w, x, and y is ", average( 3, w, x, y ),
      "The average of w, x, y, and z is ",
      average( 4, w, x, y, z ) );

   return 0; /* indicates successful termination */

} /* end main */
```

average функц і гэсэн бүхэл тоон аргумент болон өөр тодорхойгүй тооны хэдэн аргументтай байна

```
27 /* calculate average */
28 double average( int i, ... )
29 {
30    double total = 0; /* initialize total */
31    int j; /* counter for selecting arguments */
32    va_list ap; /* stores information needed by va_start and va_end */
33
34    va_start( ap, i ); /* initializes the va_list object */
35
36    /* process variable length argument list */
37    for ( j = 1; j <= i; j++ ) {
38       total += va_arg( ap, double );
39    } /* end for */
40
41    va_end( ap ); /* clean up variable-length argument list */
42
43    return total / i; /* calculate average */
44 } /* end function average */
```

```
w = 37.5
x = 22.5
y = 1.7
z = 10.2

The average of w and x is 30.000
The average of w, x, and y is 20.567
The average of w, x, y, and z is 17.975
```

va_list төрлийн хувьсагч нь бусад хувьсах аргументтай макронуудын мэдээллийг агуулна

va_start макро нь **ap**-д **i** элементийг нэмнэ

va_arg макро нь **ap**-аас дараагийн элементийг гарган ирж төрлийг нь **double** болгоно

va_end макро нь функцийг **main** руу хэвийн байдлаар буцахад туслана

Програмчлалын үндэс CS200

Лекц № 14

Сигнал боловсруулалт

А. Хүдэр

Командын мөрийн аргументуудыг хэрэглэх

- UNIX-тэй төстэй үйлдлийн системд main функц руу аргумент дамжуулах
  - main-ийг дараах байдлаар зарлана
    int main(int argc, char *argv[])
  - int argc
    - дамжуулсан аргументын тоо
  - char *argv[]
    - тэмдэгт мөр төрлийн хүснэгт
    - аргументуудын нэрийг дарааллаар нь агуулна
      - argv[0] нь эхний аргумент болно
  - Жишээ: $ mycopy input output
    - argc: 3
    - argv[0]: “mycopy”
    - argv[1]: “input”
    - argv[2]: “output”

```c
/* Example 101:
   Using command-line arguments */
#include <stdio.h>

int main( int argc, char *argv[] )
{
   FILE *inFilePtr;  /* input file pointer */
   FILE *outFilePtr; /* output file pointer */
   int c;            /* define c to hold characters input by user */

   /* check number of command-line arguments */
   if ( argc != 3 ) {
      printf( "Usage: mycopy infile outfile\n" );
   } /* end if */
   else {

      /* if input file can be opened */
      if ( ( inFilePtr = fopen( argv[ 1 ], "r" ) ) != NULL ) {

         /* if output file can be opened */
         if ( ( outFilePtr = fopen( argv[ 2 ], "w" ) ) != NULL ) {

            /* read and output characters */
            while ( ( c = fgetc( inFilePtr ) ) != EOF ) {
               fputc( c, outFilePtr );
            } /* end while */
```

main функц **argc**, **argv** гэсэн аргументуудтай байна

Програмд дараах гурван аргументыг дамжуулсан: програмын нэр, унших файлын нэр, бичих файлын нэр

Програм **argv[1]**-д зааж өгсөн файлыг уншихаар нээхийг оролдоно...

...дараа нь **argv[2]**-т зааж өгсөн файлыг бичихээр нээхийг оролдоно

Програм **inFilePtr**-ээс уншсан тэмдэгтүүдээ **outFilePtr** руу бичнэ

```
27
28          } /* end if */
29          else { /* output file could not be opened */
30             printf( "File \"%s\" could not be opened\n", argv[ 2 ] );
31          } /* end else */
32
33       } /* end if */
34       else { /* input file could not be opened */
35          printf( "File \"%s\" could not be opened\n", argv[ 1 ] );
36       } /* end else */
37
38    } /* end else */
39
40    return 0; /* indicates successful termination */
41
42 } /* end main */
```

Олон файлтай програмыг хөрвүүлэх тухай тайлбар

- Олон файлтай програм
  - Функцийн тодорхойлолтууд нэг файлд байх ёстой (хуваагдсан байж болохгүй)
  - Функцуудэд харагдах глобал хувьсагчид файлдаа байна
    - Глобал хувьсагчдыг хэрэглэж байгаа файл бүрд тэдгээрийг зарлаж өгнө
  - Жишээ:
    - flag гэсэн бүхэл тоон хувьсагчийг нэг файлд зарласан
    - Үүнийг өөр файлд ашиглахын тулд дараах үйлдлийг хэрэглэнэ

```
extern int flag;
```

  - extern
    - хувьсагчийг өөр файлд зарлагдсан гэдгийг заана
  - extern үйлдэлгүй файлуудад функцийн загваруудыг хэрэглэж болно
    - Функцийг хэрэглэж байгаа файл бүрд функцийн загварыг бичнэ

Програмыг exit болон atexit – ээр таслах

- exit функц
  - Програмыг таслана
  - Параметрууд-EXIT_SUCCESS эсвэл EXIT_FAILURE гэсэн тэмдэгтэн тогтмол
  - Хэрэгжүүлэлтэд тодорхойлсон утгыг буцаана
  - Жишээ:

    exit(EXIT_SUCCESS);
- atexit функц

  atexit(биелүүлэх функц);
  - Програмыг амжилттайгаар таслах үед дуудагдах “биелүүлэх функц”-ийг бүртгэнэ
    - atexit функц нь өөрөө програмыг таслахгүй
  - 32 хүртлэх функцийг бүртгэнэ (олон atexit функц)
    - Функцуудийг бүртгэснийх нь эсрэг дарааллаар дуудна
  - Дуудагдсан функц аргумент авах юм уу утга буцааж чадахгүй

```c
/* Example 102:
   Using the exit and atexit functions */
#include <stdio.h>
#include <stdlib.h>

void print( void ); /* prototype */

int main( void )
{
   int answer; /* user's menu choice */

   atexit( print ); /* register function print */
   printf( "Enter 1 to terminate program with function exit"
      "\nEnter 2 to terminate program normally\n" );
   scanf( "%d", &answer );

   /* call exit if answer is 1 */
   if ( answer == 1 ) {
      printf( "\nTerminating program with function exit\n" );
      exit( EXIT_SUCCESS );
   } /* end if */

```

atexit функц програмд тасрахдаа **print** функцийг дуудахыг заана

exit функц програмыг таслана

```
23    printf( "\nTerminating program by reaching the end of main\n" );
24
25    return 0; /* indicates successful termination */
26
27 } /* end main */
28
29 /* display message before termination */
30 void print( void )
31 {
32    printf( "Executing function print at program "
33       "termination\nProgram terminated\n" );
34 } /* end function print */
```

програм тасрах үед **print** функц дуудагдана

```
Enter 1 to terminate program with function exit
Enter 2 to terminate program normally
1

Terminating program with function exit
Executing function print at program termination
Program terminated

Enter 1 to terminate program with function exit
Enter 2 to terminate program normally
2

Terminating program by reaching the end of main
Executing function print at program termination
Program terminated
```

Бүхэл болон бутархай тоон тогтмолын төгсгөлийн тэмдэглэгээ

- Си хэл тогтмолуудад зориулан төгсгөлийн тэмдэглэгээг хэрэглэдэг
  - тэмдэггүй бүхэл тоо U эсвэл u
  - long бүхэл тоо L эсвэл l
  - тэмдэггүй long бүхэл тоо ul, lu, UL эсвэл LU
  - float – f эсвэл F
  - long double – l эсвэл L
  - Жишээ:

    174u
    467L
    3451ul
  - Хэрэв бүхэл тоон утгын төгсгөлд тэмдэглэгээ байхгүй бол түүнийг хадгалж чадах эхний төрлийг нь олгоно (int, long int, unsigned long int)
  - Хэрэв бутархай тоо төгсгөлийн тэмдэглэгээгүй бол double төрөлтэй гэж үзнэ

Файл

- Си хэл бинар файлуудыг боловсруулж чадна
  - Бинар файлууд зарим нэг системд байдаггүй
    - хэрэв систем бинар файл дэмждэггүй бол тэдгээрийг текст хэлбэрээр нээдэг
  - Их хурд, санах ой болон дэмжих нөхцөл шаардлагатай үед бинар файлуудыг хэрэглэдэг
  - Эсрэг тохиолдолд текст файлыг хэрэглэхэд хангалттай
    - Системээс хамаарахгүй, стандарт хэрэгслийг ашиглан өгөгдлийг шалгаж болно
- tmpfile функц
  - Түр зуурын файлыг “wb+” горимд нээнэ
  - fclose функцээр хаагдах эсвэл програм дуусах хүртэл нээлттэй байна
- rewind функц
  - Байрлалын заагчийг файлын эхэнд аваачна

Бинар файлыг нээх горимууд

Mode	Description
rb	Open an existing binary file for reading.
wb	Create a binary file for writing. If the file already exists, discard the current contents.
ab	Append; open or create a binary file for writing at end-of-file.
rb+	Open an existing binary file for update (reading and writing).
wb+	Create a binary file for update. If the file already exists, discard the current contents.
ab+	Append; open or create a binary file for update; all writing is done at the end of the file.

```c
/* Example 103:
   Using temporary files */
#include <stdio.h>

int main( void )
{
   FILE *filePtr;     /* pointer to file being modified */
   FILE *tempFilePtr; /* temporary file pointer */
   int c; /* define c to hold characters read from a file */
   char fileName[ 30 ]; /* create char array */

   printf( "This program changes tabs to spaces.\n"
      "Enter a file to be modified: " );
   scanf( "%29s", fileName );

   /* fopen opens the file */
   if ( ( filePtr = fopen( fileName, "r+" ) ) != NULL ) {

      /* create temporary file */
      if ( ( tempFilePtr = tmpfile() ) != NULL ) {
         printf( "\nThe file before modification is:\n" );

```

tmpfile функц түр зуурын файл үүсгэнэ

```
23          /* read characters from file and place in temporary file */
24          while ( ( c = getc( filePtr ) ) != EOF ) {
25             putchar( c );
26             putc( c == '\t' ? ' ' : c, tempFilePtr );
27          } /* end while */
28
29          rewind( tempFilePtr );
30          rewind( filePtr );
31          printf( "\n\nThe file after modification is:\n" );
32
33          /* read from temporary file and write into original file */
34          while ( ( c = getc( tempFilePtr ) ) != EOF ) {
35             putchar( c );
36             putc( c, filePtr );
37          } /* end while */
38
39       } /* end if */
40       else { /* if temporary file could not be opened */
41          printf( "Unable to open temporary file\n" );
42       } /* end else */
```

Уг програм **filePtr**-ээс тэмдэгтүүдийг авч **tempFilePtr** руу бичихдээ **tab**-уудыг хоосон зай болгоно

Програм дараа нь **tempFilePtr**-ээс тэмдэгтүүдийг авч **filePtr** руу бичнэ

```
43
44     } /* end if */
45     else { /* if file could not be opened */
46        printf( "Unable to open %s\n", fileName );
47     } /* end else */
48
49     return 0; /* indicates successful termination */
50
51 } /* end main */
```

```
This program changes tabs to spaces.
Enter a file to be modified: data.txt

The file before modification is:
0       1       2       3       4
        5       6       7       8       9


The file after modification is:
0 1 2 3 4
 5 6 7 8 9
```

Дохио боловсруулах

- Дохио
  - Гэнэтийн үйл явдал, програмыг таслаж болно
    - тасалдал (<ctrl>c), буруу команд, сегмент зөрчих, таслах хүсэлт, хөвөх таслалын онцгой тохиолдол (тэгд хуваах, том тоонууд үржүүлэх)
- signal функц
  - Гэнэтийн үйл явдлуудыг барих
  - <signal.h> толгой файл
  - Хоёр аргумент авна: дохионы дугаар болон дохио боловсруулах функцийн заагч
- raise функц
  - Дохионы дугаар болох бүхэл тоог авч дохиог үүсгэнэ

signal.h-ийн стандарт дохионууд

Signal	Explanation
SIGABRT	Abnormal termination of the program (such as a call to function `abort`).
SIGFPE	An erroneous arithmetic operation, such as a divide by zero or an operation resulting in overflow.
SIGILL	Detection of an illegal instruction.
SIGINT	Receipt of an interactive attention signal.
SIGSEGV	An invalid access to storage.
SIGTERM	A termination request set to the program.

```c
/* Example 104:
   Using signal handling */
#include <stdio.h>
#include <signal.h>
#include <stdlib.h>
#include <time.h>

void signalHandler( int signalValue ); /* prototype */

int main( void )
{
   int i; /* counter used to loop 100 times */
   int x; /* variable to hold random values between 1-50 */

   signal( SIGINT, signalHandler ); /* register signal handler */
   srand( clock() );

   /* output numbers 1 to 100 */
   for ( i = 1; i <= 100; i++ ) {
      x = 1 + rand() % 50; /* generate random number to raise SIGINT */

      /* raise SIGINT when x is 25 */
      if ( x == 25 ) {
         raise( SIGINT );
      } /* end if */

```

signal функц нь **SIGINT** дохио үүсэх үед **signalHandler** – ыг дуудахыг заана

raise функц **SIGINT** дохиог үүсгэнэ

```
27          printf( "%4d", i );
28
29          /* output \n when i is a multiple of 10 */
30          if ( i % 10 == 0 ) {
31             printf( "\n" );
32          } /* end if */
33
34       } /* end for */
35
36       return 0; /* indicates successful termination */
37
38 } /* end main */
39
40 /* handles signal */
41 void signalHandler( int signalValue )
42 {
43       int response; /* user's response to signal (1 or 2) */
44
45       printf( "%s%d%s\n%s",
46          "\nInterrupt signal ( ", signalValue, " ) received.",
47          "Do you wish to continue ( 1 = yes or 2 = no )? " );
48
49       scanf( "%d", &response );
50
51       /* check for invalid responses */
52       while ( response != 1 && response != 2 ) {
53          printf( "( 1 = yes or 2 = no )? " );
54          scanf( "%d", &response );
55       } /* end while */
```

```
56
57    /* determine if it is time to exit */
58    if ( response == 1 ) {
59
60       /* reregister signal handler for next SIGINT */
61       signal( SIGINT, signalHandler );
62    } /* end if */
63    else {
64       exit( EXIT_SUCCESS );
65    } /* end else */
66
67 } /* end function signalHandler */
```

signal функцийг дохио илэрсний дараа дахин дуудна

```
   1   2   3   4   5   6   7   8   9  10
  11  12  13  14  15  16  17  18  19  20
  21  22  23  24  25  26  27  28  29  30
  31  32  33  34  35  36  37  38  39  40
  41  42  43  44  45  46  47  48  49  50
  51  52  53  54  55  56  57  58  59  60
  61  62  63  64  65  66  67  68  69  70
  71  72  73  74  75  76  77  78  79  80
  81  82  83  84  85  86  87  88  89  90
  91  92  93
Interrupt signal ( 2 ) received.
Do you wish to continue ( 1 = yes or 2 = no )? 1
  94  95  96
Interrupt signal ( 2 ) received.
Do you wish to continue ( 1 = yes or 2 = no )? 2
```

Санах ойг динамикаар хуваарилах: calloc болон realloc функцууд

- Санах ойг динамикаар хуваарилах
  - Динамик хүснэгт үүсгэж чадна
- calloc(nmembers, size)
  - nmembers-элементийн тоо
  - size-элемент бүрийн хэмжээ
  - динамик хүснэгтийн хаягийг буцаана
  - malloc ба calloc – ийн гол ялгаа нь calloc хуваарилсан санах ойгоо цэвэрлэдэгт оршино
- realloc(pointerToObject, newSize)
  - pointerToObject-дахин хуваарилж буй объектийн заагч
  - newSize-объектийн шинэ хэмжээ
  - Дахин хуваарилсан санах ойн заагчийг буцаана
  - Хэрэв зай хуваарилж чадахгүй бол NULL-ыг буцаана
  - Хэрэв newSize нь тэгтэй тэнцүү бол зааж байгаа объектыг чөлөөлнө
  - Хэрэв pointerToObject нь NULL-тай тэнцүү бол malloc-тай адил ажиллана

goto ашиглан нөхцөлгүйгээр салаалах

- Бүтэцлэгдээгүй програмчлал
  - Гүйцэтгэл маш чухал үед хэрэглэнэ
  - Давталтаас нөхцөл шалгалгүйгээр гарах
- goto үйлдэл
  - Програмын удирдлагыг тэмдэглэгээний дараах үйлдэл рүү шилжүүлнэ
  - Тэмдэглэгээ нь тодорхойлох цэгээр төгсөнө (жишээ нь start:)
  - Олон давхар давталтаас хурдан гарах
    - goto start;

Зөвөлгөө

- goto үйлдлийг олон давхар бүтцээс хурдан гарахад хэрэглэнэ

```c
/* Example 105:
   Using goto */
#include <stdio.h>

int main( void )
{
   int count = 1; /* initialize count */

   start: /* label */

      if ( count > 10 ) {
         goto end;
      } /* end if */

      printf( "%d  ", count );
      count++;

      goto start; /* goto start on line 9 */

   end: /* label */
      putchar( '\n' );

   return 0; /* indicates successful termination */

} /* end main */
```

```
1   2   3   4   5   6   7   8   9   10
```

goto үйлдэлд хэрэглэгдэх тэмдэглэгээ

goto үйлдэлд програмыг заасан тэмдэглэгээ руу шилжүүлнэ

Агуулга

- Том хэмжээтэй програм хөгжүүлэхэд #include –ийг ашиглах
- #define-г ашиглан аргументтай макро зохиох
- Програмын зарим хэсгийг (жишээ нь зүгшрүүлэлтэнд туслах хэсгийг) хэзээ хөрвүүлэхийг заах буюу нөхцөлт хөрвүүлэлт хийх
- Нөхцөлт хөрвүүлэлтийн үед алдааны мэдээлэл гаргах
- Илэрхийллийн утга зөв байгаа эсэхийг баталгаа (assertion) ашиглан шалгах

Си-гийн бусад сэдвүүд

- Гараас орох оролтыг файлаас оруулах
- Дэлгэц рүү гарах гаралтыг файл руу бичих
- Хувьсах урттай аргументын жагсаалт бүхий функц бичих
- Командын мөрийн аргументуудыг боловсруулах
- Тоон тогтмолуудад тусгай төрөл олгох
- Түр зуурын файл хэрэглэх
- Програмд гадны асинхрон үйл явдлуудыг боловсруулах
- Хүснэгтэд зориулан санах ойг динамикаар хуваарилах
- Өмнө нь динамикаар хуваарилсан санах ойн хэмжээг өөрчлөх