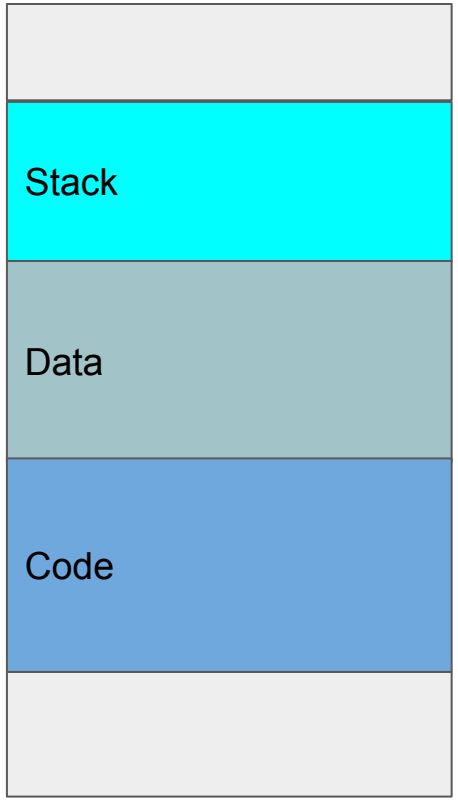
Указатели и ссылки

Память и адресация в программе

- Основная память - это массив ячеек, в которых могут храниться данные
- Каждая ячейка имеет размер 1 байт
- Каждая ячейка имеет свой уникальный **адрес**
  - Адрес - индекс ячейки в массиве

Память процесса (программы)

- Сегмент кода
- Сегмент данных
- Сегмент стека

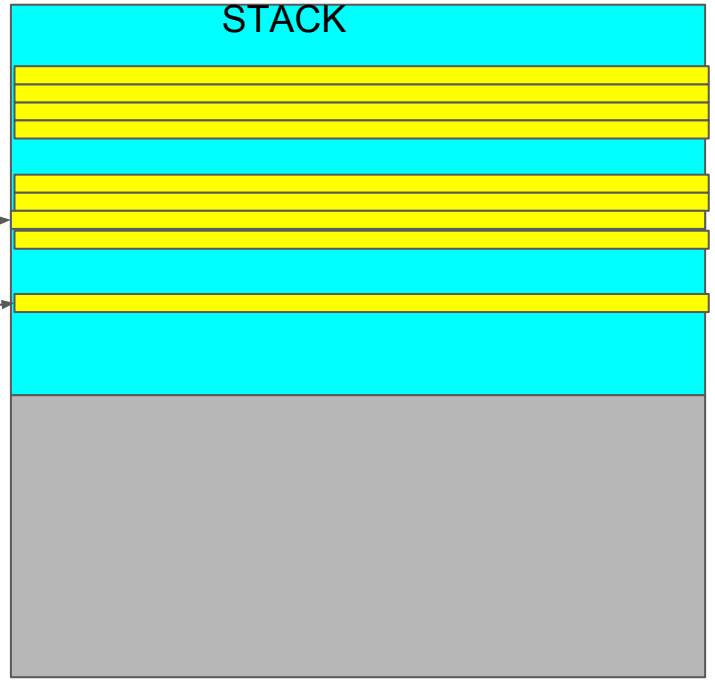


Переменные в памяти

```
int main() {
    int a;
    float b;
    char c;
// Rest of code
}
```

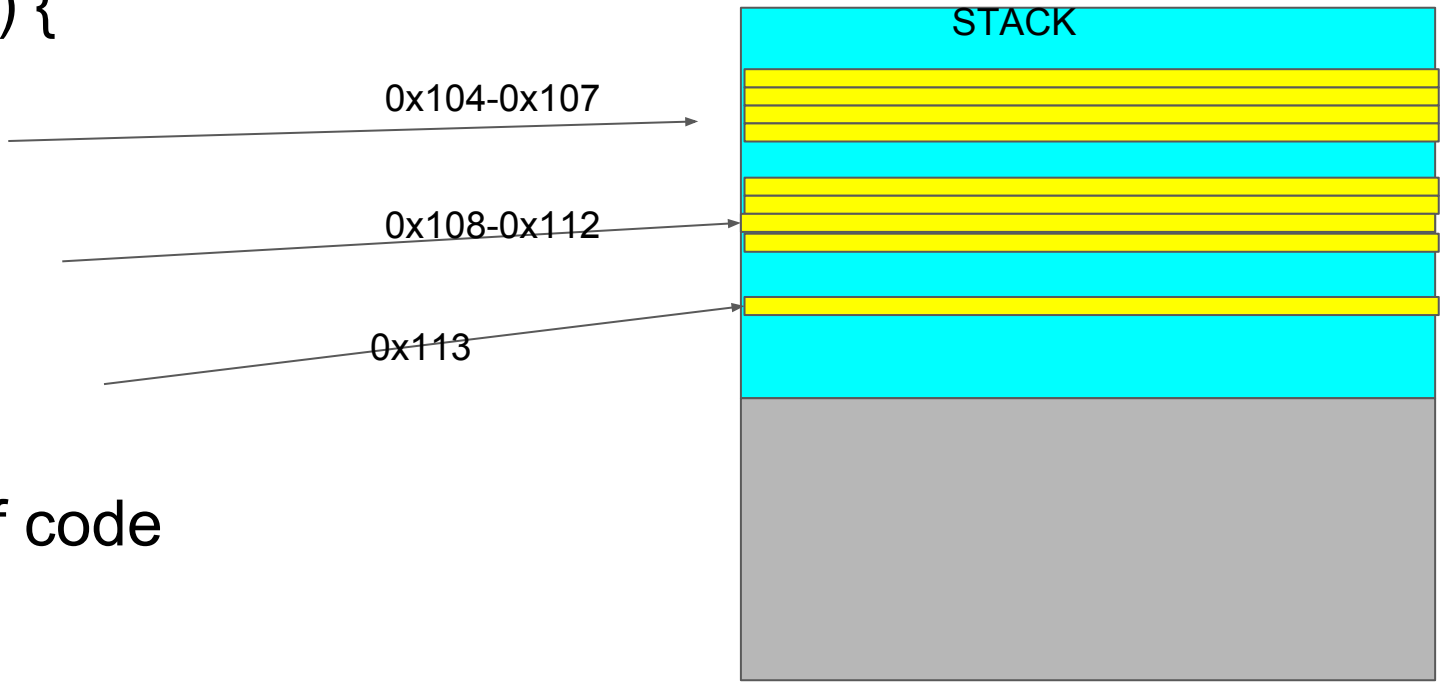
Переменные в памяти

```
int main() {
    int a;
    float b;
    char c;
// Rest of code
}
```



Переменные в памяти

```
int main() {
    int a;
    float b;
    char c;
// Rest of code
}
```



Операция получения адреса переменной

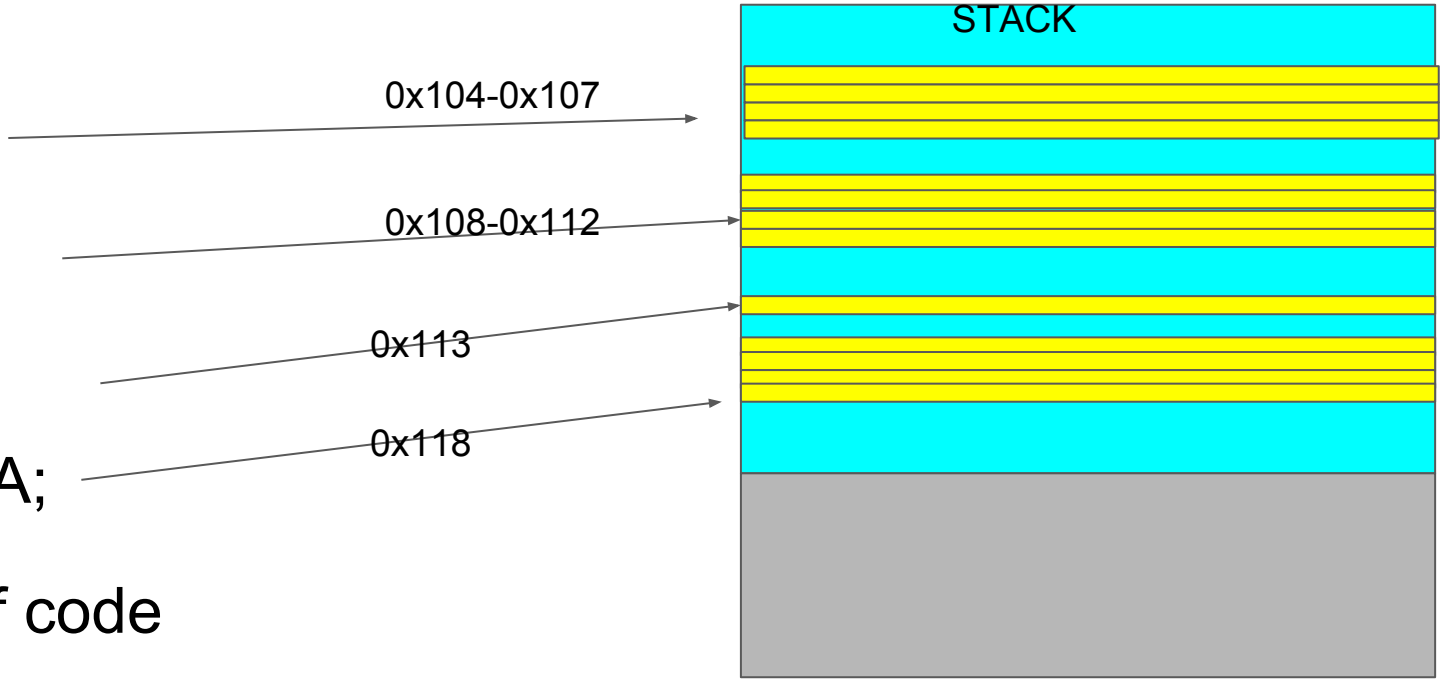
- Оператор &
  - Если есть переменная a, то оператор &a возвращает адрес этой переменной
  - Унарный оператор

Тип указатель

- Если a - переменная типа int, то каков тип выражения &a
  - &a - имеет тип указателя на int
  - int*
  - Аналогично для любого другого типа:
    - <тип>* <имя указателя>
- Если есть такой тип, то мы можем иметь переменные такого типа

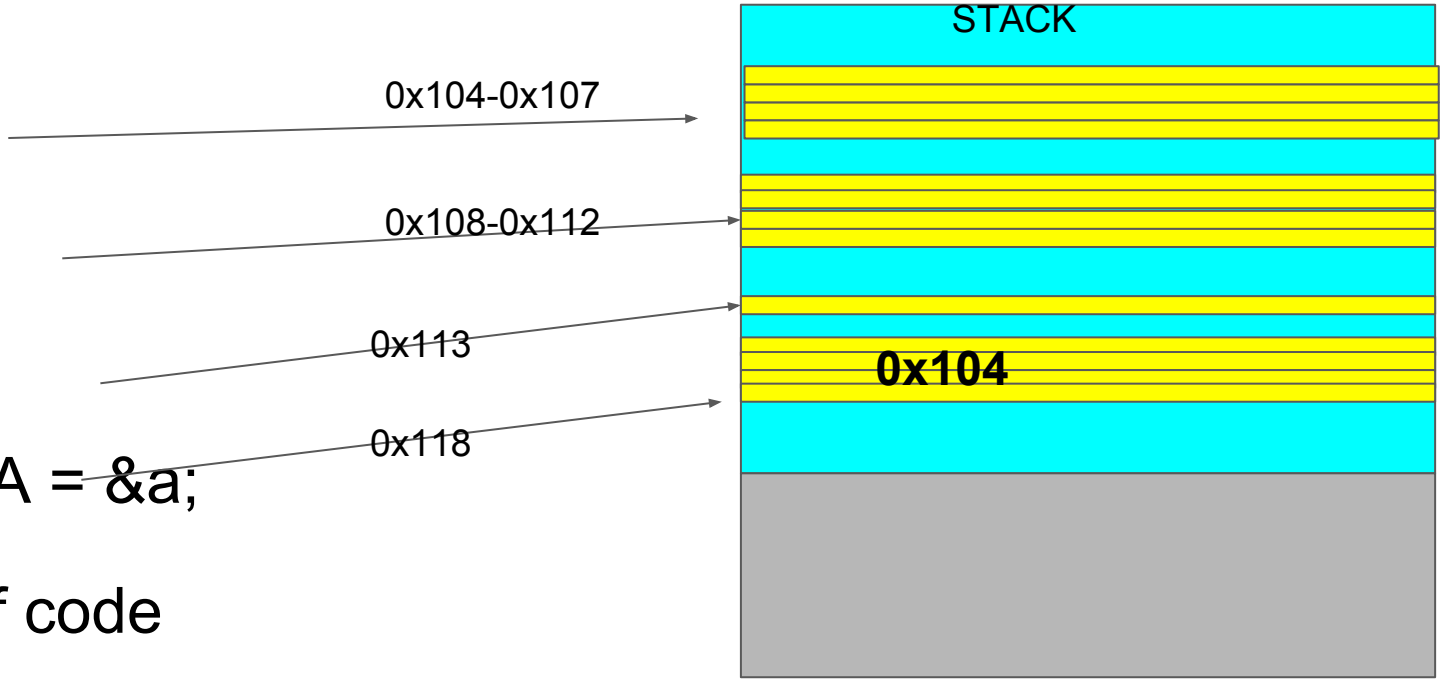
Переменная - указатель

```
int main() {
    int a;
    float b;
    char c;
    int* ptrA;
// Rest of code
}
```



Переменная - указатель

```
int main() {
    int a;
    float b;
    char c;
    int* ptrA = &a;
// Rest of code
}
```

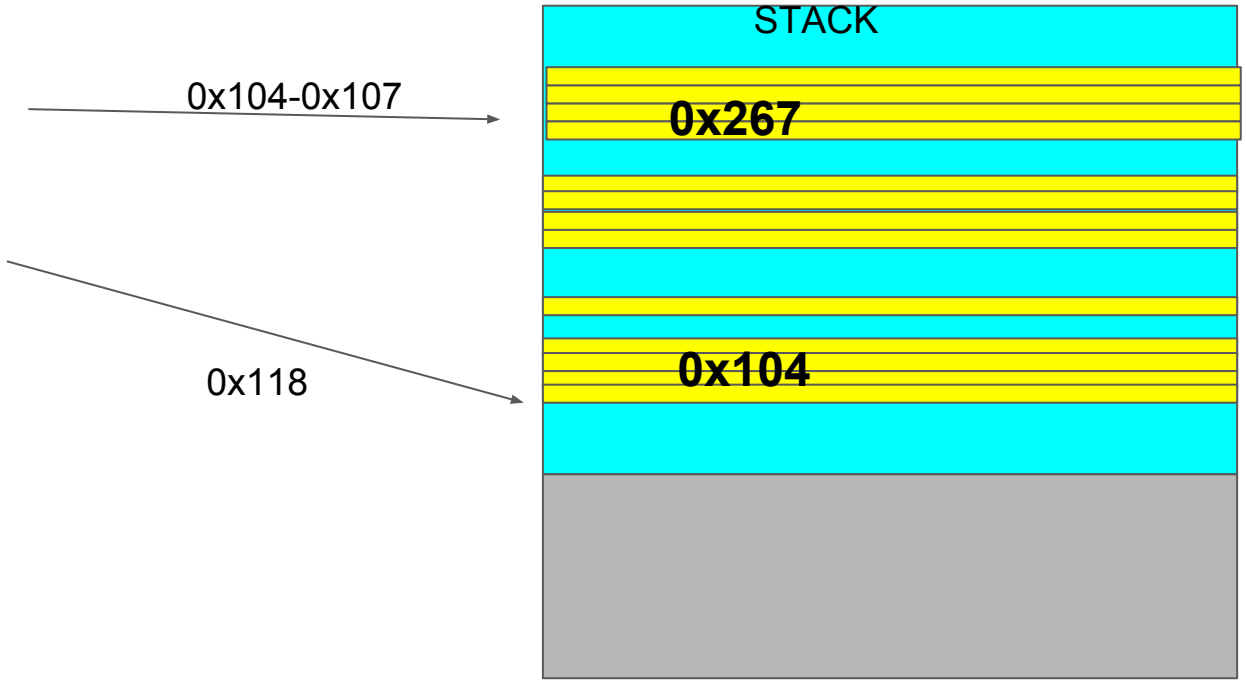


Указатель на указатель

- Каков тип выражения &ptrA ?
  - указатель на int*
  - int**
- Как далеко можно зайти?

Доступ к переменным через указатель

```
int main() {

    int a = 0x267;

    int* ptrA = &a;

// Rest of code

}
```

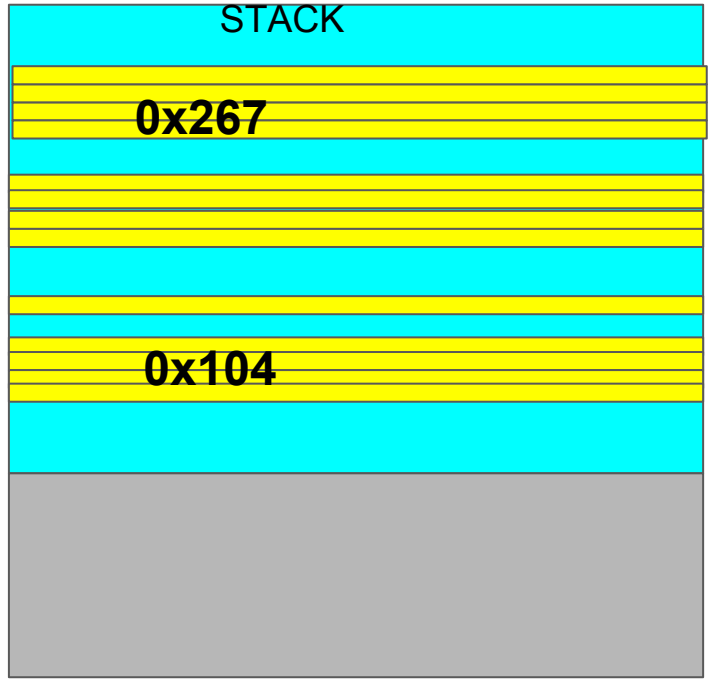


Доступ к переменным через указатель

- Оператор доступа к переменным через указатель - унарный оператор *
  - *ptrA - доступ(запись и чтение) к переменной, на которую указывает ptrA
  - операция разименования указателя
  - Тип указателя нужен, чтобы знать размер(тип) переменной к которой происходит обращение
  - Указатель может быть нулевым (0, NULL, std::null_ptr)

Доступ к переменным через указатель

```
int main() {
    int a = 0x267;
    int* ptrA = &a;
    cout<<*ptrA<<endl;
// Rest of code
}
```



Опасность операции разыменования

- Обращение к памяти, не принадлежащей процессу
  - Крах программы
- Разыменование нулевого указателя 0x0
- Разыменование не инициализированного (утерянного указателя)
  - Неопределенное поведение
  - Крах программы

Указатели и const

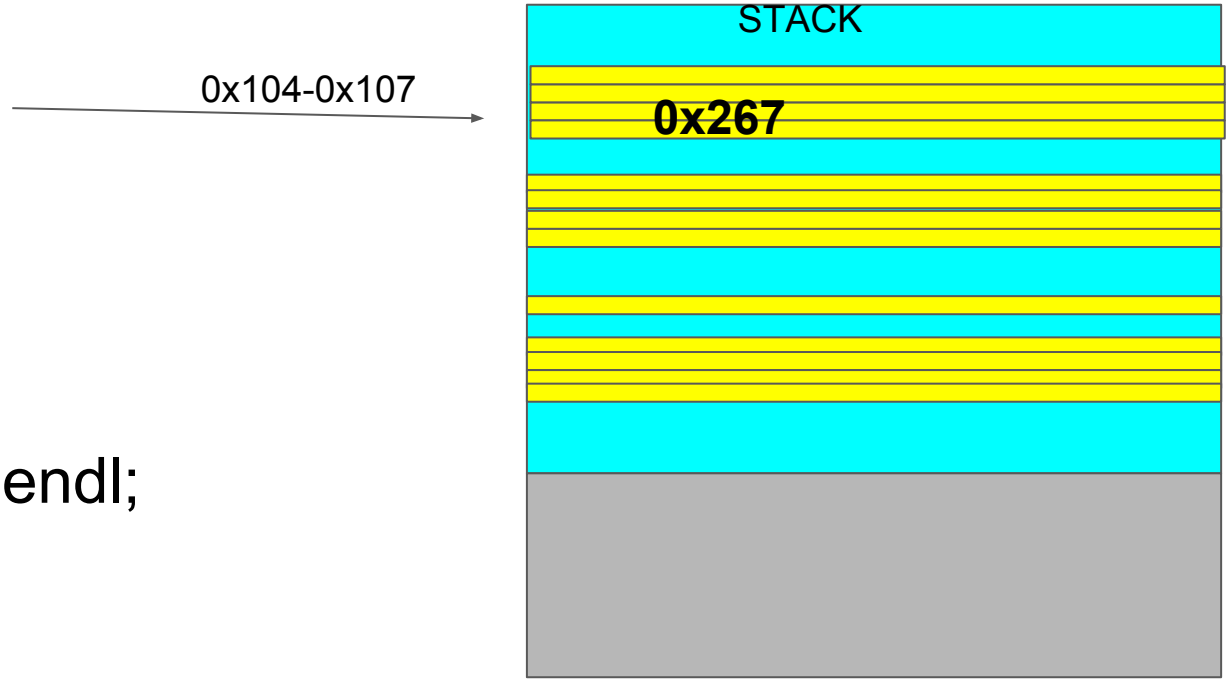
- const int* ptr; - указатель на константу
- int* const ptr = &a; - константный указатель
- const int* const ptr = &a; - константный указатель на константу

Ссылка

- Второе имя для переменной - под ссылку не выделяется память
- Все изменения с ссылкой происходят с самой переменной
- Ссылка не может быть неинициализированной
- Константная ссылка
- Объявление в коде
  - <тип переменной>& <имя ссылки> = <имя переменной на которую ссылается ссылка>
- Ссылка не может быть нулевой, но может указывать на несуществующую переменную
  - Ссылка на автоматически освобожденную память
  - Ссылка на динамически выделенную и затем удаленную память

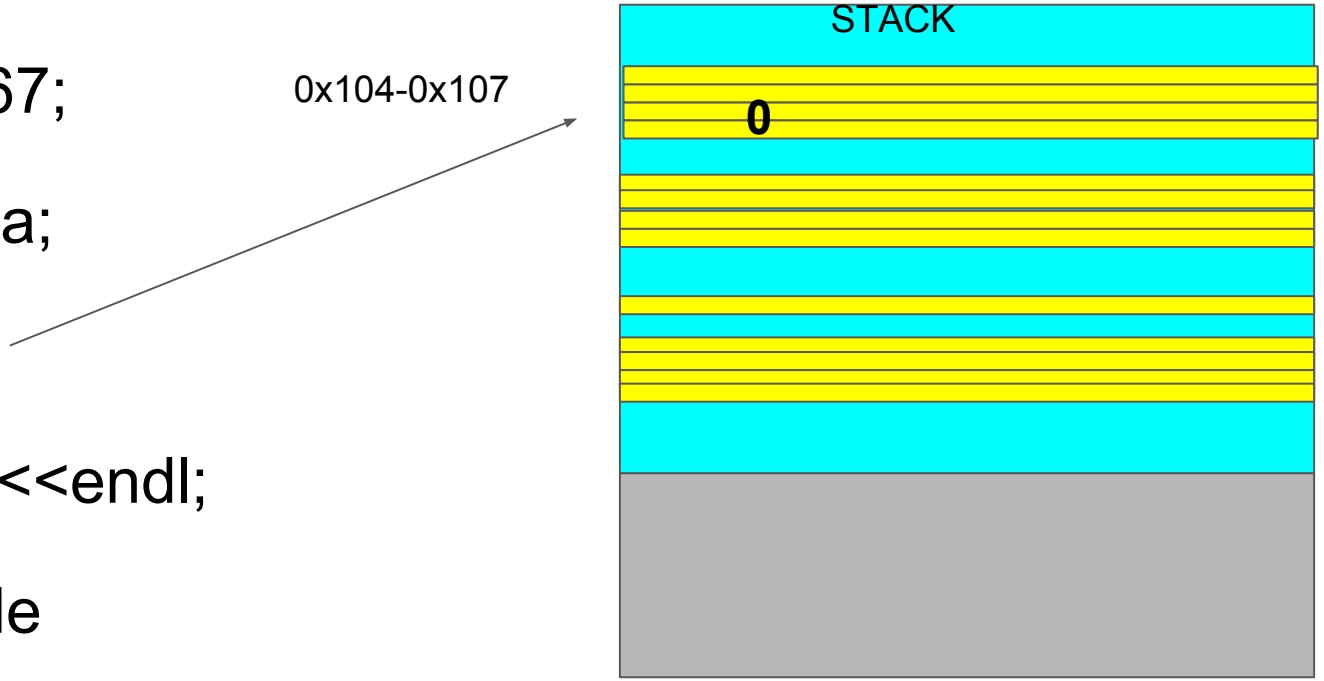
Доступ к переменным через ссылку

```
int main() {
    int a = 0x267;
    int& refA = a;
    refA = 0;
    cout<<*ptrA<<endl;
// Rest of code
}
```



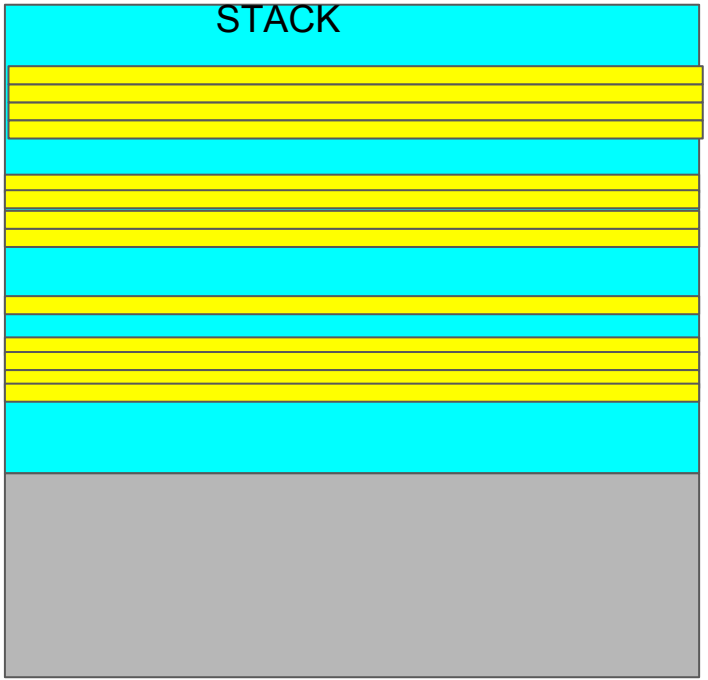
Доступ к переменным через ссылку

```
int main() {
    int a = 0x267;
    int& refA = a;
    refA = 0;
    cout<<refA<<endl;
// Rest of code
}
```



Доступ к переменным через ссылку

```
int main() {
   int a = 0x267;
   int& refA = a;
   {
      int b = 1;
      refA = b;
    }
   refA = 0;  <- Ошибка!!!
   cout<<refA<<endl;
// Rest of code
}
```



Передача переменных в функцию по ссылке

- Если мы хотим изменить переданную переменную внутри функции
- Если не хотим копировать большой объект
- Чтобы нельзя было изменять используем константную ссылку

Передача указателей в функции

- По значению - как обычную переменную
- По ссылке - тоже можно, если мы хотим чтобы значение самого указателя изменилось внутри функции

Пример

```
void swapByPtr(int* ptrX, int* ptrY);
int main() {
  int m;
  int n;
  cout << "Give m and n : ";
  cin >> m >> n;
  swapByPtr(&m, &n);
  cout << "m : " << m << endl;
  cout << "n : " << n << endl;
  return 0;
}
```

```
void swapByPtr(int* ptrX, int* ptrY) {
  int temp;
  temp = *ptrX;
  *ptrX = *ptrY;
  *ptrY = temp;
  return;
}
```

Передача указателей в функции

- Передавая указатели в функцию - даем доступ к локальным переменным (возможность их изменять)
- Другой способ достичь этого - использовать ссылки
  - На самом деле, в этом случае используются указатели
  - Код выглядит проще

Пример

```
void swapByPtr(int* ptrX, int* ptrY);
int main() {
  int m;
  int n;
  cout << "Give m and n : ";
  cin >> m >> n;
  swapByPtr(&m, &n);
  swapByRef(m, n);
  cout << "m : " << m << endl;
  cout << "n : " << n << endl;
  return 0;
}
```

```
void swapByPtr(int* ptrX, int* ptrY) {
  int temp;
  temp = *ptrX;
  *ptrX = *ptrY;
  *ptrY = temp;
  return;
}
```

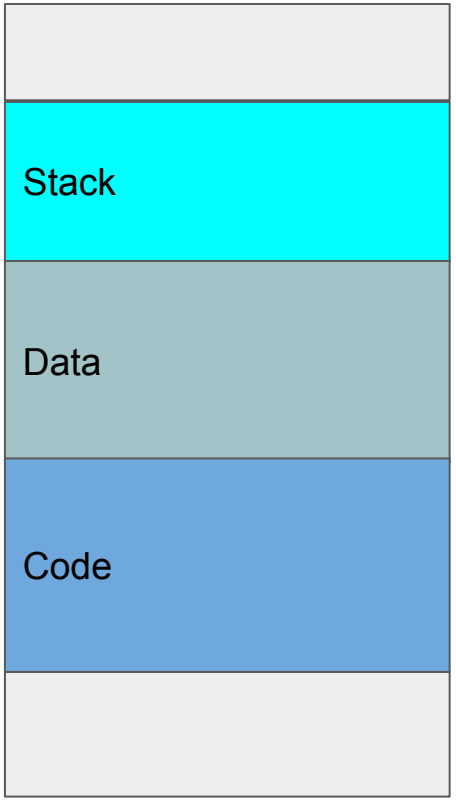
```
void swapByRef(int& X, int& Y) {
  int temp;
  temp = X;
  X = Y;
  Y = temp;
  return;
}
```

Возврат указателя из функции

- Необходима осторожность - нельзя возвращать указатель на локальные переменный функккции
  - После выхода из функции - стек функции (activation record) очищается - все локальные переменные удаляются
  - Разыменование указателя несуществующей переменной - ошибка

Память процесса (программы)

- Сегмент кода
- Сегмент данных
- Сегмент стека

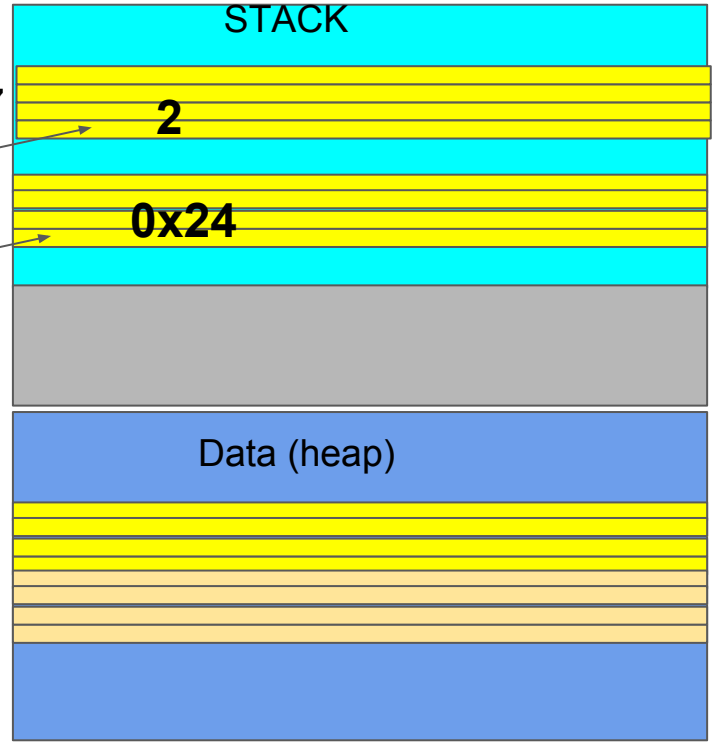


Динамическое выделение памяти

- Заранее (на этапе компиляции) неизвестен требуемый размер памяти
- Необходимо выделить память, которая будет существовать после возврата из функции

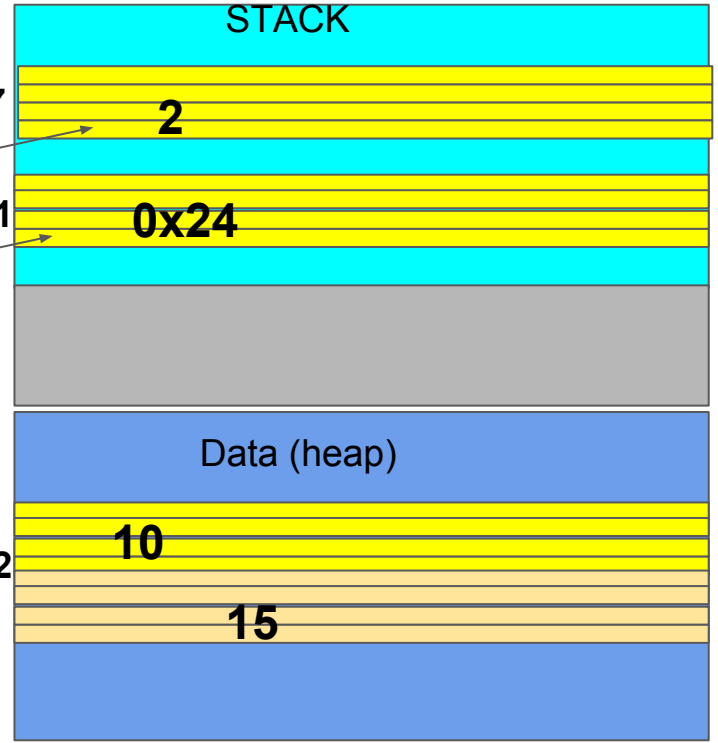
Динамическое выделение памяти

```
int main() {

    int numStudents;

    cin>>numStudents;

    int* marks;

    marks = new int [numStudents];

// Rest of code

}
```



Динамическое выделение памяти

```
int main() {
    int numStudents;
    cin>>numStudents;
    int* marks;
    marks = new int [numStudents];
    marks[0] = 10; marks[1] = 15;
}
```



Выделение памяти в общем случае

- Выделение памяти для переменной типа T
  - T* ptrT;
  - ptrT = new T;
  - Доступ к переменной через оператор ‘*’: *ptrT = 0;
- Выделение памяти для массива из переменных типа T
  - T* arr;
  - arr = new T [size]
  - Доступ к элементам через индекс arr[0], arr[i]
  - Доступ к элементам через * arr[i] эквивалентно *(arr + i)

Особенности работы с динамической памятью

- Оператор new может вернуть нулевой указатель (память может кончиться)
  - Необходима проверка
- Выделенную память необходимо освобождать
  - Оператор delete
  - delete[]
  - Проверка на нулевой указатель

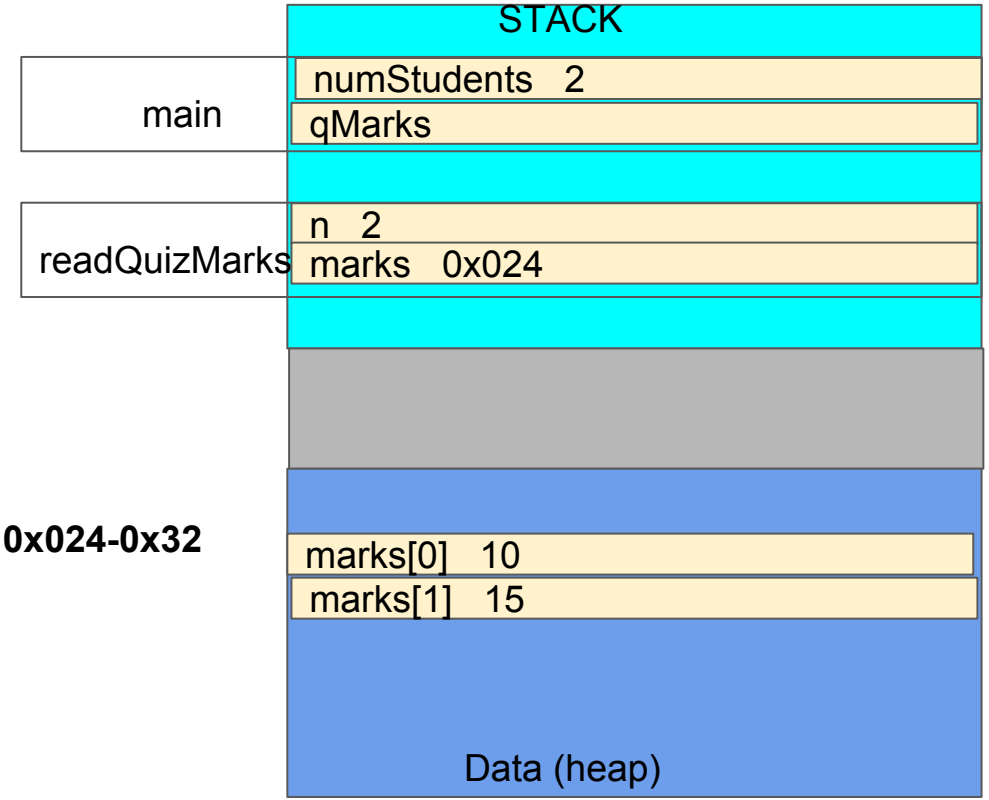
Выделение памяти в функции

```
int * readQuizMarks(int n);
int main()
{
    int numStudents;
    int * qMarks;
    cout << "Given student count:"<< endl;
    cin >> numStudents;
    cout << "Give marks of students"<< endl;
    qMarks = readQuizMarks(numStudents);
    //использование qMarks

    delete[] qMarks;
```

```
int * readQuizMarks(int n)
{
    int * marks, i;
    marks = new int[n];
    if (marks == NULL)
    {
        return NULL;
    }
    for (i = 0; i<n; i++)
    {
        cin >> marks[i];
    }
    return marks;
}
```

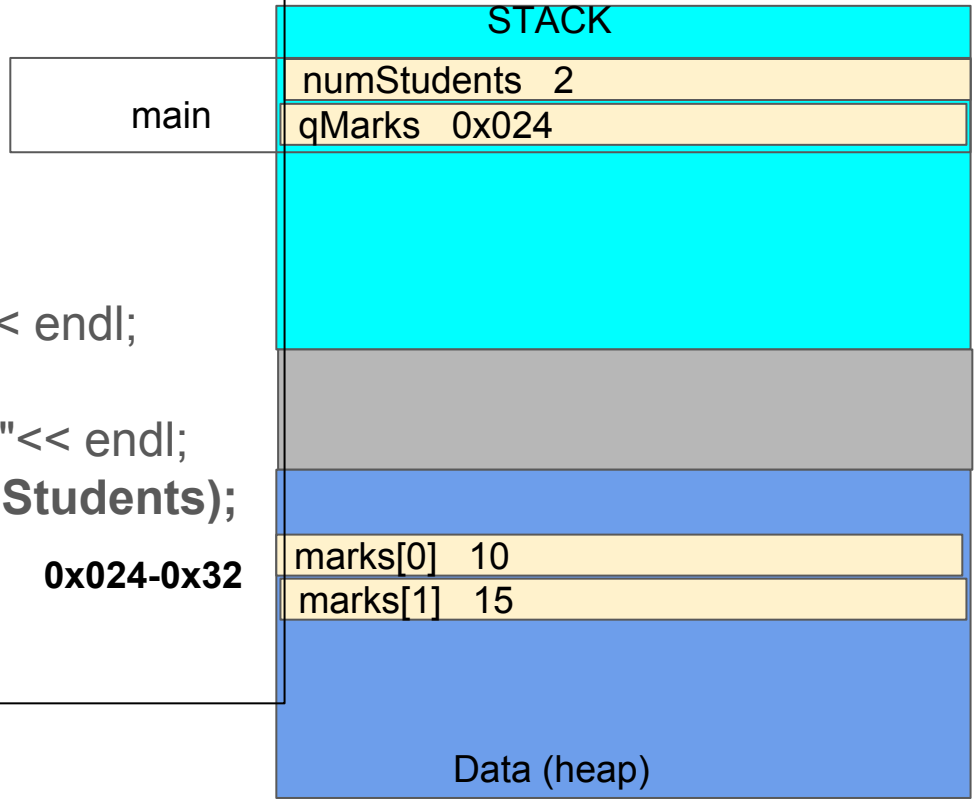
Выделение памяти в функции

```
int * readQuizMarks(int n)
{
    int * marks, i;
    marks = new int[n];
    if (marks == NULL)
    {
        return NULL;
    }
    for (i = 0; i<n; i++)
    {
        cin >> marks[i];
    }
    return marks;
}
```



Выделение памяти в функции

```
int * readQuizMarks(int n);
int main()
{
    int numStudents;
    int * qMarks;
    cout << "Given student count:"<< endl;
    cin >> numStudents;
    cout << "Give marks of students"<< endl;
    qMarks = readQuizMarks(numStudents);
    //использование qMarks
    delete[] qMarks;
```



Выделение памяти в функции

```
int * readQuizMarks(int n);
int main()
{
    int numStudents;
    int * qMarks;
    cout << "Given student count:"<< endl;
    cin >> numStudents;
    cout << "Give marks of students"<< endl;
    qMarks = readQuizMarks(numStudents);
    //использование qMarks
    delete[] qMarks;
    qMarks = NULL;
```